Université Paris Cité

Reichman University

Université Paris Cité
Ecole Doctorale Sciences Mathématiques de Paris Centre – ED386
Institut de Recherche en Informatique Fondamentale (UMR 8243)

et

Reichman University
Efi Arazi School of Computer Science
Foundations and Applications of Cryptographic Theory – FACT Center

Calcul multipartite sécurisé avec communication sous-linéaire

Par Pierre MEYER

Thèse de doctorat d'Informatique

Dirigée par Elette BOYLE
et par Geoffroy COUTEAU

Présentée et soutenue publiquement le 14 septembre 2023 devant un jury composé de:

Elette BOYLE	Associate Professor	Reichman University and NTT Research.	*Directrice*
Geoffroy COUTEAU	Chargé de Recherche	CNRS, Université Paris Cité, IRIF.	*Directeur*
Claudio ORLANDI	Full Professor	Aarhus Universitet.	*Rapporteur*
David POINTCHEVAL	Directeur de Recherche	DIENS, ENS, CNRS, Inria, Université PSL.	*Rapporteur*
Yuval ISHAI	Full Professor	Technion.	*Examinateur*
Tal MALKIN	Full Professor	Columbia University.	*Examinatrice*
Tal MORAN	Associate Professor	Reichman University.	*Examinateur*

Résumé

Le *calcul multipartite sécurisé* (en anglais, MPC) [Yao82,GMW87a] permet à des agents d'un réseau de communication de calculer conjointement une fonction de leurs entrées sans avoir à n'en rien révéler de plus que le résultat du calcul lui-même. Une question primordiale est de savoir dans quelle mesure le *coût en communication* entre les agents dépend de la *complexité calculatoire* de la fonction en question. Un point de départ est l'étude d'une hypothétique *barrière de la taille du circuit*. L'existence d'une telle barrière est suggérée par le fait que tous les protocoles MPC fondateurs, des années 80 et 90, emploient une approche "porte-logique-par-porte-logique" au calcul sécurisé: la communication d'un tel protocole sera nécessairement au moins linéaire en le nombre de portes, c'est-à-dire en la taille du circuit. De plus ceux-ci constituent moralement l'état de l'art encore de nos jours en ce qui concerne la sécurité dite "par théorie de l'information".

La barrière de la taille du circuit a été franchie pour le MPC avec *sécurité calculatoire*, mais sous des hypothèses structurées impliquant l'existence de *chiffrement totalement homomorphe* (en anglais, FHE) [Gen09] ou de *partage de secret homomorphe* (en anglais, HSS) [BGI16a]. De plus, il existe des protocoles avec sécurité *par théorie de l'information* dont la communication *en-ligne* (mais pas la communication totale) est sous-linéaire en la taille du circuit [IKM$^+$13, DNNR17, Cou19].

Notre méthodologie de recherche consiste à s'inspirer des techniques developpées dans le modèle de l'aléa corrélé—dans lequel tout résultat pourra être considéré comme plus "fondamental" que le modèle calculatoire (de par le type de sécurité obtenue) mais qui est néanmoins un modèle inadapté à comprendre la complexité de communication du MPC (puisque que l'on s'autorise à ne pas compter toute quantité de communication qui peut être reléguée à une phase "hors-ligne", c'est-à-dire avant que les participants au calcul ne prennent connaissance de leurs entrées)—pour développer de nouvelles méthodes dans le modèle calculatoire. Avec cette approche, nous obtenons des protocoles franchissant la barrière de la taille du circuit sous l'hypothèse de la sécurité quasipolynomiale de LPN [CM21] ou sous l'hypothèse QR+LPN [BCM22]. Ces hypothèses calculatoires n'étant pas précédement réputées impliquer l'existence de MPC sous-linéaire, la pertinence de notre méthodologie est, dans une certaine mesure, validée *a posteriori*. Plus fondamentalement cependant, nos travaux empruntent un nouveau paradigme pour construire du MPC sous-linéaire, sans utiliser les outils "avec de fortes propriétés d'homomorphisme" que sont le FHE ou du HSS. En combinant toutes nos techniques héritées de l'étude du modèle de l'aléa corrélé, nous parvenons à briser la *barrière des deux joueurs pour le calcul sécurisé avec communication sous-linéaire*, sans FHE [BCM23]. Spécifiquement, nous présentons le premier protocole à plus de deux participants dont la communication est sous-linéaire en la taille du circuit et qui ne soit pas fondé sur des hypothèses sous lesquelles on sait déjà faire du FHE.

Parallèlement à ces travaux centrés sur la sécurité calculatoire, nous montrons [CMPR23] comment adapter les approches à deux joueurs utilisant du HSS, à la [BGI16a], pour gurantir une sécurité "théorie de l'information" à l'un des deux joueurs et une sécurité calculatoire à l'autre. Ceci est, de façon prouvable, la notion de sécurité la plus forte que l'on puisse espérer en présence de seulement deux joueurs (sans aléa corrélé). Nous obtenons le premier protocole de ce type avec communication sous-linéaire, qui ne soit fondé sur des hypothèses sous lesquelles on sait déjà faire du FHE.

Mots-Clefs: Cryptographie · Fondations · Calcul Multipartite Sécurisé · Partage de Fonction Secrète

Résumé Substantiel

La cryptographie. Jusqu'aux années 1970, la cryptographie pouvait être décrite comme l'art de communiquer de façon privée au travers de canaux a priori non sécurisés. Un art, puisque les "codes secrets" depuis l'Antiquité au milieu de la Guerre Froide (le code de César ou celui de la machine Enigma par exemple) ne sont pas invulnérables, et peuvent être cassés avec suffisament d'astuce. Tout l'enjeu du travail d'un cryptographe était alors de prédire les vecteurs d'attaque potentiels, et de trouver un moyen d'y remédier. Dès la Seconde Guerre Mondiale, cette faible notion de sûreté basée sur l'espoir qu'aucune faille n'émerge se montre obsolète avec l'apparition des premiers ordinateurs, explosant la capacité de calcul d'attaquants. La cryptographie moderne se caractérise au contraire par la notion de *sécurité prouvable*—bien qu'il eût été préférable que fût retenu par l'usage le terme de *sûreté prouvable*—, et permet d'accomplir des tâches plus complexes que la communication sécurisée. Si l'on établit un cadre formel définissant les capacités des adversaires et les propriétés de securité que l'on souhaite guarantir, il devient en effet possible de fournir une preuve mathématique qu'*aucun* tel adversaire ne peut briser un schéma ou protocole donné, et de fait exclure l'existence de toute astuce ou vecteur d'attaque efficace. Il est cependant primordial de garder à l'esprit que la cryptographie n'est qu'un moyen de guarantir la sécurité et la sûreté des systèmes d'information, et qu'en pratique elle est à combiner avec des moyens physiques (issus de l'ingénierie électronique par exemple, ou même plus simplement par des moyens "humains"). En effet, la cryptographie théorique permet d'établir des théorèmes, accompagnés de leurs *preuves de sécurité*, mais ceux-ci ne sauraient être appliqués si leurs hypothèses ne sont pas réunies. À titre d'exemple, la sécurité de la plupart des schémas cryptographiques supposent que les utilisateurs honnêtes—c'est-à-dire ceux auquels ont veut pouvoir donner des guaranties de sécurité—ont accès à une source d'aléa parfaite et que cet aléa pourra être stocké à l'abri de l'adversaire contre lequel on cherche à se prémunir. Il s'agira donc d'assurer ces guaranties par d'autres moyens, ou à défaut de retravailler le théorème pour se passer de ces hypothèses. L'art d'appliquer la cryptographie dans le but d'assurer la sûreté des systèmes d'information consiste à trouver un équilibre subtil entre les modèles théoriques et les situations pratiques. Plus celles-ci permettront à ceux-là de faire des hypothèses fortes, et plus les guaranties pourront être élevées.

Une fois qu'un modèle théorique adéquat a été choisi, on peut espérer prouver qu'un protocole donné satisfait effectivement telle ou telle propriété de sécurité. Pour appréhender ce concept, considérons que deux personnes, que nous nommerons selon l'usage Alice et Bob, souhaitent échanger un message privé par le biais de canaux de communication public, épiés par un espion passif (c'est-à-dire qui ne fait qu'enregistrer les messages, sans les modifier) que nous appelerons Eve[1]. Si on arrive à établir que l'ensemble des informations accessibles à l'adversaire, soit la transcription intégrale des messages échangés entre les participants par des canaux publics, est une variable aléatoire (où l'espace de probabilité est pris relativement à l'entropie de l'aléa local de Alice et Bob) qui soit indépendente d'un message à caractère privé échangé par les participants, alors clairement les messages ne seront d'aucune utilité à l'adversaire qui n'aura donc aucun vecteur d'attaque. Ce type d'argument donne lieu à ce que l'on appelle la *sécurité de type théorie de l'information*. Sans définir cette dernière notion, l'idée est que nul adversaire, même avec un temps de calcul illimité, ne pourra récupérer

[1]Le jeu de mots sur "eavesdropper" étant proprement intraduisible, nous admettrons sans démonstration la pertinence du nom.

d'information sensible puisque ce qui lui est donné n'encode aucune information. C'est cependant une très forte notion de sécurité qui est prouvablement inatteignable dans de nombreuses circonstances. Lorsque nous considérons des adversaires avec une capacité de calcul bornée, il suffit que les informations auquelles ils ont accès leur "paraissent indépendantes". Cette fois-ci les messages contiendront bien—au sens de la théorie de l'information—des informations sensibles, mais qui sont inaccessibles avec les ressources calculatoires de l'adversaires. Idéalement, nous aimerions pouvoir prouver l'existence de deux distributions qui soient statistiquement éloignées (très abusivement, "elles correspondent à des chiffrés de deux messages différents") mais calculatoirement proches (c'est à dire qu'elle ne peuvent pas être distinguées "efficacement"), et s'en servir comme base pour la cryptographie. Malheureusement un tel théorème est bien au-delà de l'état actuel des mathématiques, et il nous devons nous contenter pour l'heure de travailler avec des *hypothèses de sécurité* supplémentaires. Heureusement, la communauté mathématique a su produire des candidats pour des problèmes "cryptographiquement durs", ayant résisté à de nombreux efforts de cryptanalyse. Admettant qu'un tel problème soit effectivement dur, il est alors possible de prouver la sécurité d'un schéma ou protocole via une *réduction de sécurité*. En substance, on montre que s'il existait une attaque contre le protocole, alors il existerait une attaque contre le problème sous-jacent, pourtant réputé dur. Réduire la sécurité de tous nos protocoles à une liste restreinte d'hypothèses a, entre autres, le mérite de permettre de concentrer les efforts de cryptanalyse.

Le calcul multipartite sécurisé. Historiquement, la cryptographie était employée exclusivement dans le but de sécuriser la communication, et c'est encore de nos jours son application principale. Ainsi, Alice et Bob cherchent à communiquer de façon fiable et privée en la présence d'un adversaire avec a minima la faculté d'épier tous les messages qu'ils s'échangent, voire même celle de les modifier. L'attaquant est ici externe au système, et tous les participants, considérés honnêtes, se font confiance. Cependant, depuis la fin des années soixante-dix la cryptographie est capable de fournir des solutions pour assurer la sûreté des systèmes distribués en présence d'un adversaire qui soit interne au système. Admettons qu'un ensemble de deux entités ou plus, détenant chacune une entrée, souhaitent calculer conjointement une fonction de leurs entrées respectives. Formellement, les N participants détiennent respectivement x_1, x_2, x_3, etc. et cherchent à calculer la valeur de $f(x_1, \ldots, x_N)$, où f est une fonction entendue de tous. Si les entrées relèvent d'un caractère privé et que les agents ne peuvent pas se faire confiance, il semblerait a priori—sans la cryptographie—que la seule solution serait de choisir un tiers de confiance qui serait parfaitement incorruptible et de lui déléguer le calcul. Cette solution a un certain nombre d'avantages, comme celui d'empêcher les agents de "tricher", par exemple en choisissant leur propre entrée en fonction de celles des autres. Le *calcul multipartite sécurisé* (en anglais, *Secure Multi-Party Computation*, abbrégé MPC) permet de remplacer l'hypothèse (souvent irréaliste) de l'existence d'un tel tiers par un protocole distribué, tout en préservant les guaranties. Spécifiquement, le MPC permet à des agents de calculer une fonction de leurs entrées conjointes *sans avoir à n'en rien révéler qui ne puisse être déduit de la sortie de l'adversaire*. Cette quantité d'information est le strict minimum qu'il soit nécessaire de révéler tout en réalisant correctement le calcul. En effet, l'adversaire étant interne au système, on ne peut espérer prouver qu'il n'apprenne *rien* des entrées des agents honêtes: si par exemple la fonction calculée et de determiner l'intersection des données des agents, il est inévitable que l'adversaire pourra déduire que sa sortie est un sous-ensemble de l'entrée de tous les participants. Une question naturelle est celle de

comment on peut espérer prouver un tel théorème. La solution retenue par la communauté MPC est la notion de *preuve par simulation*: si l'on produit un algorithme (que l'on appelle un simulateur) qui, étant donnée les entrées et sorties des joueurs corrompus par l'adversaire, est capable de génerer une "fausse transcription" d'une exécution du protocole qui soit indistinguable de la vraie du point de vue de l'adversaire, alors essentiellement par définition il est impossible à l'adversaire d'apprendre plus dans une véritable exécution du protocole que ce qui est donné en entrée au simulateur dans l'expérience de pensée.

Le MPC trouve son utilité principale dès lors qu'il est nécessaire d'interagir avec des parties ayant des objectifs opposés ou concurrents. La diplomatie (et plus généralement, toute forme de négociation ou d'enchères) repose souvent sur des informations secrètes telles que ce chaque partie apprécie le plus, et ce que chaque partie sait des intentions de l'autre partie. Même—ou peut-être surtout—dans la diplomatie informelle, toutes les informations pertinentes ne peuvent pas être mises sur la table ouvertement (Le prisonnier à échanger est-il en fait un espion, et si oui, que sait-il ? Quelle est sa valeur ?). Un protocole de calcul sécurisé peut être utilisé pour "émuler les négociations" (avec une preuve mathématique rigoureuse que cette émulation s'est faite "honnêtement", sans chercher à avantager qui que ce soit) et de telle sorte que ne soit révélée aucune information intermédiaire au-delà des termes de l'accord final. Le MPC est un outil versatile qui peut même éventuellement permettre aux participants de nier l'existence de toutes négociations jusqu'au moment de l'accord final! Nous remarquons toutefois que les applications ne se limitent pas au domaine traditionnel de la cryptographie militaire, et qu'il s'agit d'un outil qui peut tout à fait trouver sa place dans la vie quotidienne. Un exemple classique, et léger, est son application à outrepasser la peur du rejet. Ainsi deux personnes (ou plus) ne seraient peut-être pas disposées à se faire une déclaration d'intérêt directement, de peur que leurs sentiments ne soient pas réciproques. La cryptographie leur permet alors de se passer du besoin d'amis de confiance pour agir en tant qu'intermédiaires.

Une barrière à la taille du circuit? Une question fondamentale dans l'étude du MPC est de comprendre le *coût de la sécurité*. En effet, apporter des quaranties de sécurité requiert que les participants dépensent un certain nombre de ressources telle que des bits de calcul, d'aléa, et de communication. Cette dernière peut être considérée comme la plus "chère", motivant tout particulièrement la question de chercher à minimiser la *complexité de communication* de nos protocoles. De façon intriguante, tandis que le calcul distribué d'une fonction ne nécessite que l'échange de ses entrées, tous les protocoles séminaux utilisent une quantité de communication qui croît linéairement avec le nombre de portes de sa représentation sous forme de circuit. Cette observation nous mène à considérer la question suivante.

> *La* complexité de communication *du calcul multipartite sécurisé est-elle fortement correlée avec la* complexité de calcul *de la fonction calculée?*[2]

Le premier pas est de déterminer la nature de cette "barrière à la taille du circuit" suggérée par les protocoles fondateurs du domaine: s'agit-il d'une limitation inhérente au problème, ou simplement d'un artefact historique? Il aura fallu une trentaine d'années

[2]Cette élégante formulation de la question (à traduction près) est à attribuer à Yuval Ishai, et est issue de sa présentation "Private Information Retrieval, Part I" à la 10ème école d'Hiver sur la Cryptography de Bar-Ilan.

de recherche en cryptographie pour un premier élément de réponse. En 2009, Gentry [Gen09] construisit un schéma de chiffrement totalement homomorphe (en anglais, *Fully Homomorphic Encryption*, abbrégé FHE) qui mènerait ensuite à des constructions de MPC avec communication indépendente de la taille du circuit sous des hypothèses calculatoires standard, quoique fortes et spécifiques [DFH12, AJL+12]. Plus récemment, une famille de protocoles à deux joueurs, dont la communication est tout juste *sous-linéaire en la taille du circuit*, a vu le jour avec l'introduction du *partage de secret homomorphe* (en anglais, *Homomorphic Secret Sharing*, abbrégé HSS) [BGI16a]. D'abord sous l'hypothèse DDH [BGI16a], puis sous DCR [FGJS17, OSY21, RS21]. Ces percées prometteuses rendent envisageable la perspective de briser la barrière de la taille du circuit sous des hypothèses de moins en moins structurées, voires idéalement sous des hypothèses minimales.

Malgré ces avancées dans le modèle de la sécurité calculatoire, aucun tel résultat n'existe pour la sécurité de type "théorie de l'information", que l'on pourra pourtant considérer comme le modèle le plus fondamental. Une série de travaux a bien réussi à franchir la barrière de la taille du circuit [IKM+13, DNNR17, Cou19], mais dans le modèle de l'aléa corrélé. Bien que celui-ci permette de comprendre la complexité de communication *en-ligne*, c'est-à-dire après que les participants aient connaissance de leurs entrées, il ne saurait apporter une réponse satisfaisante à notre question dans la mesure où il s'autorise à ne pas prendre en compte toute communication *hors-ligne*.

Nos résultats relatifs au MPC avec communication sous-linéaire. Notre méthodologie de recherche consiste à s'inspirer des techniques developpées dans le modèle de l'aléa corrélé pour développer de nouvelles méthodes dans le modèle standard, en commençant par viser la sécurité calculatoire.

- *Calcul sécurisé avec communication sous-linéaire sous de "nouvelles" hypothèses.* À mesure que nous développons de nouvelles approches pour franchir la barrière de la taille du circuit, il nous faudra un moyen concret de juger de l'originalité d'un protocole. Nous nous proposons de suivre le critère de déterminer si l'on obtient un protocole sous une hypothèse calculatoire standard qui n'était pas précédement réputée impliquer l'existence de MPC sous-linéaire. Nous insistons cependant sur le fait qu'il ne s'agisse que d'un critère, et qu'obtenir des protocoles sous une multiplicité d'hypothèses n'est pas une fin en soit, mais plutôt une étape dans l'appréhension du problème. Nous espérons qu'en s'éloignant des spécificités de telle ou telle hypothèse structurée nous verrons émerger des principes plus fondamentaux, que nous pourrions extraire.

- *Converger au cœur de la barrière de la taille du circuit.* Une fois que plusieurs approches ont été trouvées pour briser la barrière de la taille du circuit, les différents protocoles pourront être classés selon leur "puissance". Bien qu'il soit en général hautement désirable qu'un protocole présente de fortes propriétés de sécurité ou ait une complexité de communication bien inférieure à la taille du circuit, ces propriétés peuvent se montrer "parasitiques" si notre but est de comprendre la barrière de la taille du circuit. À mesure que nous développons des méthodes différentes et en extrayons des principes communs, nous devrions nous attendre à ce que ces propriétés (telles que le fait d'obtenir de la sécurité statistique pour l'un des joueurs, ou celle d'avoir une communication indépendente de la taille du circuit) disparaissent, à moins bien sûr que celles-ci ne soient inhérentes. Ainsi, comprendre les différences entre les différents protocoles que nous pouvons établir est primordial.

Nos résultats se divisent en quatre grandes parties.

2PC avec communication sous-linéaire sous "quasi-polynomial LPN" [CM21]. Nous présentons un schéma de partage de secret homomorphe singleton (en anglais *single-circuit HSS*) pour n'importe quel circuit log / log log-local. Les parts d'entrées peuvent être générées avec une quantité de communication linéaire en la largeur du circuit (et une quantité polynomiale de calcul), et la sécurité du schéma se réduit à la sécurité super-polynomiale de l'hypothèse LPN. Le cœur de notre construction est une nouvelle famille de PCG pour toute correlation additive et log / log log-locale.

Notre application principale, et la motivation pour ces travaux, est la construction d'un protocol de calcul bipartite sécurisé pour tout circuit—arithmétique ou booléen—dont les portes peuvent être partitionnées en couches ordonnées de telle sorte que chaque arête du circuit raccorde un nœud d'une couche à un nœud de la couche suivante. Ce résultat étend la liste des hypothèses calculatoires sous lesquelles on sait briser la "barrière de la taille du circuit", pour une classe de circuit expressive. Par ailleurs, la force de l'hypothèse est très liée au facteur de sous-linéarité. Ainsi, pour tout $k(s) \leq (\log \log s)/4$, nous obtenons un protocole à communication $\mathcal{O}(s/k(s))$ sous l'hypothèse de la $s^{2^{k(s)}}$-sécurité de LPN.

Seules les hypothèses LWE, DDH, et une variante de la sécurité circulaire de DCR étaient précédemment réputées impliquer l'existence d'un PCG pour des corrélations de degrée super-constant ou du calcul sécurisé "générique" avec communication sous-linéaire.

Fonctions pseudo-aléatoires contraintes (en anglais, CPRF) à partir de partage de secret homomorphe [CMPR23]. Nous proposons et analysons une stratégie simple pour obtenir une fonction pseudo-aléatoire contrainte (CPRF) à partir de partage de secret homomorphe. Nous contributions intermédiaires sont les suivantes. D'abord, nous identifions des propriétés utiles qu'un schéma de HSS doit posséder pour que notre approche aboutisse. Ensuite, nous montrons que la plupart des constuctions connues de HSS satisfont effectivement ces propriétés, ce qui nous permet entre autre d'obtenir des CPRF (pour diverses classes de contraintes) sous une multitude d'hypothèses calculatoires. En particulier, nous produisons la première CPRF (à une clef, avec sécurité selective, et avec contrainte privée) pour des contraintes de type "produit scalaire" ainsi que la première CPRF (à une clef, avec sécurité selective) pour des contraintes dans NC^1 sous l'hypothèse DCR. Enfin, nous démontrons l'utilité des propriétés de HSS que nous avons identifiées en revisitant deux applications classiques du HSS au MPC. Premièrement, nous obtenons du 2PC dans le modèle du pré-calcul silencieux, où l'une des deux parties peux pré-calculer l'intégralité de son aléa correlé avant même d'intéragir avec l'autre partie. Deuxièmement, nous obtenons du 2PC avec communication sous-linéaire avec sécurité statistique pour l'un des deux joueurs (avec toutefois certaines restrictions sur les calculs que l'on peut tolérer).

Calcul sécurisé avec communication sous-linéaire sous de "nouvelles" hypothèses [BCM22]. Le calcul multipartite sécurisé permet à des parties se méfiant mutuellement de néanmoins calculer une fonction de leurs entrées sans avoir à n'en rien révéler de plus que (ce qui peut être déduit de) la sortie du calcul elle-même. Comprendre la quantité de communication que requiert un tel protocole est un problème ouvert majeur, et un premier pas consiste à determiner quand celle-ci peut être *sous-linéaire* en la taille de tout circuit représentant la fonction à calculer. Pour certains fonctions

spécifiques, telle que le PIR, la question s'étant même à la sous-linéarité en la taille des entrées.

Nous développons ici de nouvelles techniques permettant entre autres d'étendre l'ensemble des hypothèses calculatoires sous lesquelles nous savons obtenir du calcul sécurisé avec communication sous-linéaire, à la fois dans le cas du calcul général et dans celui du PIR.

- *Sous-linéarité en la taille du circuit.* Nous présentons un protocole pour calculer de façon sécuriser tout circuit "layered", avec communication sous-linéaire en la taille du circuit. Ce protocole est basé sur l'existence d'un protocole de transfert inconscient en deux tours et un rendement asymptotiquement optimal, et possédant de plus une certaine propriété de "décomposabilité". Cette propriété n'est pas purement théorique, mais est satisfaite par les protocoles de transfert inconscient de Brakerski-Branco-Döttling-Pu [BBDP22].

  En particulier, nous obtenons ainsi du calcul sécurisé avec communication sous-linéaire sous la conjonction des hypothèses QR et LPN, ce qui n'était pas connu précédemment. De plus s'agit d'une approche fondamentalement nouvelle, dans la mesure où c'est le premier protocole qui ne soit basé ni sur le chiffrement totalement homormophe ni sur le partage de secret homomorphe.

- *Sous-linéarité en la taille des entrées.* Nous construisons un protocole de PIR calculatoire avec communication polylogarithmique sous l'hypothèse CDH. Tous les protocoles précédents sous cette hypothèse utilisent une communication linéaire (en la taille de la base de données).

Calcul multipartite sécurisé avec communication sous-linéaire, sans FHE [BCM23]. Des progrès significatifs ont été faits dans la quête d'obtenir sous une multitude d'hypothèses des protocoles de calcul sécurisé avec communication sous-linéaire en la taille du circuit, mais presque exclusivement dans le cas *bipartite*. Cependant, dans le cas du calcul *multipartite* (à plus de deux parties), la seule approche connue est basée sur le FHE, même s'il n'y a que trois parties (dont deux corrompues).

Nous présentons un paradigme pour obtenir du calcul sécurisé $(N+1)$-partite avec communication sous-linéaire, basé sur une forme de partage de fonction secrète qui soit seulement N-partite. Ceci nous permet d'obtenir des protocoles tri-, quadra-, ou penta-partites pour diverses classes de fonctions sous lesquelles on ne sait pas construire de FHE.

Liste des Articles Présentés dans cette Thèse

En adéquation avec les instructions de l'Université Paris Cité, nous recensons ici les articles présentés dans cette thèse, et présents en texte (quasi-)intégral.

1. Breaking the Circuit Size Barrier for Secure Computation under Quasi-Polynomial LPN.
    - Co-auteur: Geoffroy Couteau.
    - Statut: Publié dans Canteaut, A., Standaert, FX. (eds) Advances in Cryptology – EUROCRYPT 2021. EUROCRYPT 2021. Lecture Notes in Computer Science, vol 12697. Springer, Cham.
    - DOI: `https://doi.org/10.1007/978-3-030-77886-6_29`
    - Accès ouvert: `https://eprint.iacr.org/2021/943`
2. Sublinear Secure Computation from New Assumptions.
    - Co-auteurs: Elette Boyle et Geoffroy Couteau.
    - Statut: Publié dans Kiltz, E., Vaikuntanathan, V. (eds) Theory of Cryptography. TCC 2022. Lecture Notes in Computer Science, vol 13748. Springer, Cham.
    - DOI: `https://doi.org/10.1007/978-3-031-22365-5_5`
    - Accès Ouvert: https://eprint.iacr.org/2023/513
3. Constrained Pseudorandom Functions from Homomorphic Secret-Sharing.
    - Co-auteurs: Geoffroy Couteau, Mahshid Riahinia, et Alain Passelègue.
    - Statut: Publié dans Hazay, C., Stam, M. (eds) Advances in Cryptology – EUROCRYPT 2023. EUROCRYPT 2023. Lecture Notes in Computer Science, vol 14006. Springer, Cham.
    - DOI: `https://doi.org/10.1007/978-3-031-30620-4_7`
    - Accès Ouvert: `https://eprint.iacr.org/2023/387`
4. Sublinear-Communication Secure Multiparty Computation does not require FHE.
    - Co-auteurs: Elette Boyle et Geoffroy Couteau.
    - Statut: Publié dans Hazay, C., Stam, M. (eds) Advances in Cryptology – EUROCRYPT 2023. EUROCRYPT 2023. Lecture Notes in Computer Science, vol 14005. Springer, Cham.
    - DOI: `https://doi.org/10.1007/978-3-031-30617-4_6`

Nous recensons de plus les articles dont seul un résumé est présent dans cette thèse.

5. Topology-Hiding Communication from Minimal Assumptions.
    - Co-auteurs: Marshall Ball, Elette Boyle, Ran Cohen, Lisa Kohl, Tal Malkin, et Tal Moran.

- Version Conférence: Publié dans Pass, R., Pietrzak, K. (eds) Theory of Cryptography. TCC 2020. Lecture Notes in Computer Science, vol 12551. Springer, Cham.
- DOI: `https://doi.org/10.1007/978-3-030-64378-2_17`
- Version Journal: Accepté pour publication dans *Journal of Cryptology*.
- Accès Ouvert: `https://eprint.iacr.org/2021/388.pdf`

6. On Low-End Obfuscation and Learning.

   - Co-auteurs: Elette Boyle, Yuval Ishai, Robert Robere, et Gal Yehuda
   - Statut: Publié dans Yael Tauman Kalai, editor, 14th Innovations in Theoretical Computer Science Conference (ITCS 2023), volume 251 of Leibniz International Proceedings in Informatics (LIPIcs), pages 23:1–23:28, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz- Zentrum für Informatik.
   - DOI: `https://doi.org/10.4230/LIPIcs.ITCS.2023.23`

7. Towards Topology-Hiding Computation from Oblivious Transfer.

   - Co-auteurs: Marshall Ball, Alexander Bienstock, et Lisa Kohl.
   - Statut: Accepté pour publication à TCC 2023.
   - Accès Ouvert: `https://eprint.iacr.org/2023/849.pdf`

8. New Random Oracle Instantiations from Extremely Lossy Functions.

   - Co-auteurs: Chris Brzuska, Christoph Egger, Geoffroy Couteau, et Pihla Karanko.
   - Statut: Manuscrit en soumission.
   - Accès Ouvert: `https://eprint.iacr.org/2023/1145.pdf`

Université Paris Cité

Ecole Doctorale Sciences Mathématiques de Paris Centre – ED386

Institut de Recherche en Informatique Fondamentale (UMR 8243)

and

Reichman University

Efi Arazi School of Computer Science

Foundations and Applications of Cryptographic Theory – FACT Center

Doctoral Thesis

Sublinear-communication secure multiparty computation

or: *How I Stopped Communicating and Started Computing*

Candidate: Pierre Meyer

Advisors:
Prof. Elette Boyle
Dr. Geoffroy Couteau

A thesis submitted in fulfillment of the requirements for the Degree of Doctor of Philosophy (Ph.D.) in Computer Science

And defended publicly on September 14th 2023 before a jury comprised of:

Elette Boyle	Associate Professor	Reichman University and NTT Research.	*Advisor*
Geoffroy Couteau	Chargé de Recherche	CNRS, Université Paris Cité, IRIF.	*Advisor*
Claudio Orlandi	Full Professor	Aarhus Universitet.	*Reviewer*
David Pointcheval	Directeur de Recherche	DIENS, ENS, CNRS, Inria, Université PSL.	*Reviewer*
Yuval Ishai	Full Professor	Technion.	*Examiner*
Tal Malkin	Full Professor	Columbia University.	*Examiner*
Tal Moran	Associate Professor	Reichman University.	*Examiner*

Abstract

Secure Multi-Party Computation (MPC) [Yao82, GMW87a] allows a set of mutually distrusting parties to perform some joint computation on their private inputs without having to reveal anything beyond the output. A major open question is to understand how strongly the *communication complexity* of MPC and the *computational complexity* of the function being computed are correlated. An intriguing starting point is the study of the *circuit-size barrier*. The relevance of this barrier is a historical, and potentially absolute, one: all seminal protocols from the 1980s and 1990s use a "gate-by-gate" approach, requiring interaction between the parties for each (multiplicative) gate of the circuit to be computed, and this remains the state of the art if we wish to provide the strongest security guarantees.

The circuit-size barrier has been broken in the computational setting from specific, structured, computational assumption, via *Fully Homomorphic Encryption* (FHE) [Gen09] and later *Homomorphic Secret Sharing* [BGI16a]. Additionally, the circuit-size barrier for *online communication* has been broken (in the correlated randomness model) information-theoretically [IKM$^+$13, DNNR17, Cou19], but no such result is known for the *total communication complexity* (in the plain model).

Our methodology is to draw inspiration from known approaches in the correlated randomness model, which we view simultaneously as fundamental (because it provides information-theoretic security guarantees) and inherently limited (because the best we can hope for in this model is to understand the *online* communication complexity of secure computation), in order to devise new ways to break the circuit-size barrier in the computational setting. In the absence of a better way to decide when concrete progress has been made, we take extending the set of assumptions known to imply sublinear-communication secure computation as "proof of conceptual novelty". This approach has allowed us to break the circuit-size barrier under quasipolynomial LPN [CM21] or QR and LPN [BCM22]. More fundamentally, these works constituted a paradigm shift, away from the "homomorphism-based" approaches of FHE and HSS, which ultimately allowed us to break the *two-party barrier for sublinear-communication secure computation* and provide in [BCM23] the first sublinear-communication protocol with more than two parties, without FHE. Orthogonally to this line of work, purely focusing on computational security, we showed in [CMPR23] that [BGI16a] could be adapted to provide information-theoretic security for one of the two parties, and computational security for the other: these are provably the strongest security guarantees one can hope to achieve in the two-party setting (without setup), and ours is the first sublinear-communication protocol in this setting which does not use FHE.

Keywords: Cryptography · Foundations · Secure Multiparty Computation · Function Secret Sharing

To my parents, my sisters, and my grandfather.

Foreword

When a section of this thesis describes results obtained as part of joint work with other researchers, we emphasise that fact in order to assign proper credit for what we describe as "our results"; however even within such sections the meaning of "we" will vary from "my co-authors and I" to *pluralis majestatis*[3] (in particular, when providing retrospective analyses and subjective interpretations of jointly authored works). This should not be taken to mean named parties necessarily endorse any claims made within this document.

Acknowledgments

Thanks to David Pointcheval and Claudio Orlandi for taking time ouf of their busy schedules to review this thesis, and in doing so became the first, and probably last, people to read this document. Thanks to Yuval Ishai, Tal Malkin, and Tal Moran for agreeing to serve on the committee as examiners. Thanks to Elette Boyle and Geoffroy Couteau for advising me these past few years. I am honored to have such a distinguished committee assess my PhD proposal.

It always struck me as odd that cryptographers, who work so hard to guarantee others' privacy, would be willing to broadcast sensitive information in this section of their manuscript. One way to achieve perfect privacy would be to simply copy and paste someone else's acknowledgements (with bonus points if they happened to therein thank me by name...). Unfortunately, this would fail to achieve any reasonable notion of "correctness", so I set out to design a cryptosystem which would allow me to have this section be a long pseudo-random string (to be generated with a corresponding backdoor) which could be opened to each reader's own personalised message.

```
D'‘‘@p>J[|4X2VUSe@?bO<Ln,+k#i'g2|A/R~}|_):xwYotsrk1oQPlkd*)JIedcb[!BXWV
[ZSwWPONMRQPImM/KDhHGFED=%;_?>=6|:32Vwv432+O/.-&+$H('&feBz@~}|^]yxq7utV
rkji/mOkdcb(‘ed]#a‘_A@\[TxXQ9UTMqQJ2NMLKDhHA)E>b%A@?87[;{z810/S-,+0/('&
J*#('~Dedzyx>_{tyr8vuWslkj0hPfkdc)afe^]ba‘Y}@VUyS;QPOsMRKJONGkEJCBGFE>b
%A@?87[5:921U5u-,10/(Lm+*)"!&%${Ab~}vut:xwpXn4Ukji/mleM*hgfH%cb[ZY}]\U=
SwWVUN6Lp3ONMLEDh+*F?D=BA@?8\<;432765.R210)o'&Jk)"F&f$#"!~}v<zy[qpunml2
pohg-kjihJI_%cba‘_A@VzZ<XWPUTMqQJnNMLEJCgG@E>CB;_?>~<5Y3y705.-Q1qp('&Jk
)"!&}C{c!x>_{tyr8Yutsl2pRhmfkjiha'eGcba‘Y}]\>=SwWVUTMqQPImG/EJCHAeE'=<A
@?8\}54321U/u-,+Op.'K+$)(!EfeB"y~}v{zyr8punm3qpihg-kdcha'Hd]\[ZY}]VUZSR
vVUTMLpP2NGFEiC+A@EDCB;_?>=6|:32V6/.32+O/.-,+*#(!E%${"y?}|utyxwpo5srkpi
/mlkjchg‘&dFE[‘_X]VzTSRQu8NMRKPIHGkKJCgGF(DCBA@9]~<5:381U/4-,10/(L,+$#G
!~%|{A!~w_{zs9wYunmrqpi/mfN+ib(‘H^]\[!1
```

When I realised this would require me to implement one of my own protocols, I decided to instead resort to the traditional format of acknowledgements, albeit in a somewhat reserved form. I shall confine more involved expressions of sentiment to secure point-to-point channels, but know that many of you mean much, much, more to me than you probably realise (and certainly more than I am capable of expressing).

My journey as a researcher up to this point has been such a wonderfully natural sequence of events it is nigh impossible to describe in purely secular terms. First and foremost, I thank my parents for my upbringing and allowing each of their children to

[3] To be understood here as "my ego and I".

develop and express themselves in their own unique way, and for nurturing my calling as a mathematician. I thank my grandfather M.J.P. for being such a great source of inspiration, and one of the most remarkable people I know. I would like to thank "le Maître" Abehsira, whose pedagogical genius (a term I use very sparingly) kept alive the spark of mathematical research in a young mind sometimes prone to self-doubt. I would like to thank Binh-Minh Bui-Xuan and Clémence Magnien for providing such a positive first experience of academic research: the fact they always kept an open door, and were so willing to listen to and accept ideas coming from a student has shaped the way I view research today. I would like to thank Damien Stehlé and Alain Passelègue for pointing a Masters' student in search for the right internship in the perfect direction. I thank Daniel Tschudi and the Aarhus crypto group for being so welcoming into the cryptographic community. Finally, I thank Elette Boyle and Geoffroy Couteau for being such exceptional advisors, and everything I had hoped for when I came to them.

None of this would have been possible without the kind support of my friends over the years. I cannot congratulate you on your choice of friend, but I am deeply grateful for it. Antoine, Idan, Louis-Henri, Timothée, Gauthier, Adrien, Anne-Lise, Elizabeth, Audrey, Yannick, Solveig, Maximilien, Blandine P., Enguérand, Blandine F., Léonard, Julul, HélèH, Josol, DeneƆ, and "Chef".

Thanks to Storm Storm and Hund for being there when I needed you. Thanks to the members of the "IRIF coffee break team", and especially Dancsi, Conker, Avi, and Simona[4]. Thanks to Cecilia, Simon, Divya, Amos, Hila, Dor, Matan, Shuli, Jessica, Hai for the many fun times in the Tel Aviv area. Also thanks to FACT's Ran, Tal, Adi, Rita, Alon. Thanks to Geoffroy, Elette, Lisa, Marshall, Ran, Tal Ma., Tal Mo., Yuval, Gal, Robert, Mahshid, Alex, Dung, Alain, Chris, Christoph, and Pihla for the wonderful research.

Finally, my thanks to all those of you who have brightened one of my days.

Organisation of this Thesis

In this thesis, chapter 1 contains an introduction to the topic and a detailed summary of our results, chapter 2 preliminarily regroups standard notions from the literature, chapter 3 recalls the state of the art on low-communication secure computation (excluding our own results presented in this thesis), chapter 4 describes our construction of sublinear-communication secure two-party computation from quasipolyomial LPN via single-circuit HSS, chapter 5 provides our two-party construction with one-sided statistical security from DCR or DDH, chapter 6 presents our protocol for sublinear-communication secure two-party computation from QR+LPN via correlated SPIR, chapter 7 expands on our breakthrough to the multiparty setting (i.e. beyond two parties) without fully homomorphic encryption, and finally chapter 8 lists a few immediate open questions raised by our work.

A tutorial on skimming through this manuscript. The expert reader looking for a ten-page comprehensive overview, exclusive to this thesis, of our published works [CM21,CMPR23,BCM22,BCM23] may wish to start with section 3.2.3 and then read section 1.3 (except section 1.3.2). Furthermore, each of the technical chapters (chapters 4 to 7) starts with a high-level section which is akin to a technical overview (sections 4.1, 5.1, 6.1 and 7.1).

[4]If you are currently reading this in book format instead of the traditional (and impractical) A4, you have Simona and her own thesis to thank for inspiring this great idea!

Contents

Chapter 1

Introduction

1.1 Cryptography

One typically thinks of cryptography as the art of designing or breaking secret codes, used by intelligence agencies to achieve secure communication over public channels, and this was an apt description up until the mid 1970s. Over the past fifty years however, its use has been democratised with the rise of everyday global communication over the Internet, and furthermore the nature of cryptography has evolved in two significant ways.

From an art to a science. The main drawback of what was historically referred to as a "secret code" is that its security relies solely on the hope an attacker will not find a flaw the designer overlooked. In contrast, the main feature of modern cryptography is that it provides *provable security* guarantees and rules out the existence of *any* attack. This is done by first establishing a formal framework modelling an adversary's power and defining desirable security properties, and then providing a rigorous mathematical proof that a scheme satisfies these security definitions. Still, it is important to keep in mind that cryptography is only one way to guarantee the security of information systems, and in practice it is crucial it be combined with other crafts and sciences. For instance, cryptography can allow two parties Alice and Bob to agree on a secret value even in the presence of an eavesdropper given access to the complete transcript of their communication. However, the security proof guaranteeing that the agreed-upon value remains secret may not—and should not be expected to—hold if the adversary is given extra power, such as the ability to surreptitiously modify the messages between Alice and Bob. What if the eavesdropper is able to gain additional information beyond the communication between them? For instance, monitoring the electrical consumption of Alice's computer might give away information about her local computation. Bob could be tricked through social engineering into revealing part of the secret. Furthermore, most security proofs only hold provided the parties use a cryptography-grade source of randomness. These vulnerabilities and many more can be handled by cryptography—provided they are factored into the model—but it will often be preferable to also use a combination of engineering and other physical methods to safeguard information systems.

Once a satisfactory model for the capabilities of users (*e.g.* Alice and Bob in the above paragraph) and adversaries has been established, we can prove that a given protocol satisfies a formal security definition. One might hope to prove that the transcript of the communication between Alice and Bob is a sequence of uniformly random messages which is *independent* (in the mathematical sense of formal random variables)

of the underlying private information they are exchanging. Of course these messages, when considered in conjunction with either of the parties' local state (i.e. "the randomness they used"), are not independent of the payload since both parties should be able to retrieve this information: it is only the adversary's partial view of the transcript which can be truly random. This strong notion of security is referred to as *perfect*, as informally the adversary's view carries no information which allows them to determine the users' secrets. Unfortunately perfect security may be provably impossible to achieve in some settings, and even when it is possible, it can be prohibitively expensive in terms of the users' resources. When we consider adversaries with bounded computational resources, it will be sufficient that their views only "seem independent" to them because of their restricted power.

Ideally we would want *unconditional*[1] security guarantees (against a given class of adversaries, *e.g.* computationally bounded). This is a tall order, and in most cases far beyond the scientific community's current state of knowledge. Most often the best we can hope for is a *security reduction*: the ability to break a given scheme is *reduced* to the ability to solve an underlying problem, which is presumed to be hard (but not necessarily proven to be so). In other words, *assuming* the underlying problem cannot be broken (by some class of adversaries), the scheme is secure. If an assumption is not broken despite being extensively studied by the community it will be considered *standard*, although it should be noted there is a perpetual debate in the community as to which assumptions should be considered standard and which should not. There are many sources of "potential hardness" used to provide candidates for *computational assumptions*, such as integer factorisation or lattice and code-based problems.

The security of a cryptographic scheme can alternatively be reduced to the security of another. Should the second scheme be sufficiently "elementary" we will talk about *a generic assumption*. These include *(the existence of) one-way functions* (OWF), which are "easy to compute, but hard to invert", or *(the existence of a) key-agreement (protocol)* (KA), which allows parties to establish a shared secret over a public channel. If a converse reduction exists (i.e. object A can be reduced to object B and object B can be reduced to object A), then we will talk about *minimal assumptions*. In the same way not everyone agrees on which computational assumptions are standard, there is some level of subjectivity involved in deciding which assumptions deserve to be called *generic* or *minimal*[2]. The main advantage of constructions from generic assumptions rather than from specific computational assumptions is modularity: we have complete freedom over the choice of implementation of the starting point of the reduction.

From secret codes to complex protocols. Typically, one thinks of cryptography as a tool used to achieve communication which is secure against external adversaries. In its modern form however, cryptography can be achieved to achieve more complex tasks. In section 1.2, we describe *secure multiparty computation*, which can be used to perform complex computations (beyond secure communication) while guarding against

[1]This is the first and last time we will use this term (in the context of security proofs) in this thesis, or indeed in any of our works. The issue is it obfuscates the fact that the model itself (*e.g.* if the reader is familiar with these technical terms, the fact an adversary is *passive* or that there is an *honest-majority*) should arguably be seen as an assumption. If these assumptions are overlooked when applying cryptographic tools to a problem, consequences might be devastating. More concretely, the term *unconditionally secure* is often used in the context of MPC even though the scheme protocol requires the parties communicate over *secure point-to-point channels*, which can itself be a challenge of (software and hardware) engineering to implement.

[2]Since everything can be reduced to itself—whether it even exists or not—one has to be careful in which assumptions are labelled "minimal" least the concept become meaningless.

threats which are internal to the information system.

1.2 Secure Multiparty Computation

Two or more parties may wish to perform some joint computation of their private inputs, but they may not trust each other enough to simply share their inputs. If all parties could agree on a trusted third party, they could simply delegate this computation: everyone would send their inputs to this trusted party, then wait to receive the result. The existence of such a third party is of course often an unrealistic assumption, and the goal of *secure multiparty computation* (MPC) is to instead provide the parties with a protocol they can run amongst themselves. Before describing how it might be achieved, we start by listing a few scenarios where secure computation may be desirable.

Usecases. MPC has a range of applications in settings when one is required to interact with a party with opposing or competing goals. Diplomacy (and more generally, any form of negotiation or auction) often relies on secret information such as what each party values most, and what each party knows about the other party's intentions. Even—or perhaps especially—in backchannel diplomacy, not every piece of relevant information can be put on the table openly (Is the prisoner to be exchanged in fact a spy, and if so what do they know? How much are they worth?). A secure computation protocol can be used to "emulate the negotiations" (with a rigorous mathematical proof that this emulation was done "honestly", without trying to advantage any party) in a garbled way which reveals no intermediary information beyond the final agreement. As an additional benefit, this can also be used to hide the fact any negotiation is even taking place, up until the point of final agreement. The classic example for this is *private dating*. Alice and Bob may not be willing to declare interest to each other directly, and so—without 2PC—would have to rely on trusted friends to act as intermediaries. We leave it to the reader to extend the above example to the multiparty setting.

As a second example, let's say a patient is brought to the hospital with unusual symptoms. In order to find the best treatment, the doctors in this hospital—hereafter considered a collective party we refer to as "the hospital"—needs to know if other patients presented the same symptoms, possibly in different hospitals, and if so which treatment worked for them. The problem is that previous patients' privacy is very important, so the different hospitals cannot simply share their databases with each other in the clear. However, if the different hospitals run a secure multiparty computation protocol, they can use the data without violating patient privacy.

Simulation-based security. When an adversary is external (for instance an eavesdropper), then we can hope to guarantee they learn "no information". However, when an adversary is internal to the communication system, they can corrupt parties who hold some information as inputs and are entitled to learn their own outputs. Therefore, intuitively, the security notion we can hope to capture is that the adversary, who corrupts some of the parties and gains access to their view of the protocol's execution, should learn nothing about the honest parties' inputs beyond what they could infer from the corrupted parties' inputs and outputs. Proving this statement more formally can be done by providing a *simulator*, which is given this exact information (the corrupted parties' inputs and outputs) as input and outputs the joint view of the corrupted parties. If no adversary can distinguish between the simulated view (which,

by definition, cannot possibly leak any information beyond what is allowed) and the real view of a protocol's execution, we will call the protocol *standalone secure*. We refer to Lindell's [Lin16] tutorial on the simulation proof technique for a more in-depth introduction to the topic.

Standalone simulation security does not necessarily provide any guarantees as multiple instance of a protocol are composed, sequentially or in parallel. While a in-depth explanation is beyond the scope of this introduction, we mention that in Canetti's [Can01] *Universal Composability* (UC) framework, in order to compose protocols one must be able to simulate the *behaviour of the adversary*, as opposed to simply simulating its view. The simulator is then called an *ideal adversary* which, by definition, cannot be passing on sensitive information to a malicious environment.

The cost of security. Once secure multiparty computation is defined, we can ask whether it is even possible to achieve. We will assume that n parties, who are fully connected by secure and authenticated point-to-point channels, wish to perform secure computation (of arbitrary functions). As mentioned in section 1.1, proving formal security claims requires modelling adversaries and parties. An adversary is allowed to corrupt some of the parties running a protocol, and the principle parameters governing the strength of the adversaries are the type and number of corruptions, as well as the computational power of the adversary. An adversary is *passive* (or *semi-honest*) if corrupt parties follow the protocol (and only try to "passively" gain information by recording their transcript), *semi-malicious* if corrupt parties follow the protocol but the adversary sets the value of their random tape, and *malicious* if the adversary can control the behaviour of corrupt parties arbitrarily. A threshold-t adversary can corrupt up to t parties out of n; if $t < n/2$ we will say there is an *honest majority* and if $t < n/3$ that there is an *honest super-majority*.

In the presence of a computationally unbounded adversary, it is known [BGW88, CCD88] that any functionality can be securely computed with perfect correctness and security, against a passive adversary if there is an honest majority of parties (threshold $t < n/2$), and against an active adversary if there is an honest two-thirds super-majority of parties ($t < n/3$). If the parties additionally have access to a secure broadcast channel, then statistical correctness can be achieved against active adversaries even if we only assume a simple honest majority [RBO89].

In the cryptographic setting, where the adversary is limited to run in polynomial-time, and still under the assumption that the parties have access to a full set of pairwise connected channels, then it is known that any function can be securely computed while tolerating all-but-one corruptions ($t < N$) from *Oblivious Transfer* (OT) [Yao86, GMW87b, GMW87a, Kil91].

However, the communication complexity of all these seminal protocols is linear in the circuit size of the function computed. Fundamentally, this is because all these works can be seen as reducing the ability to compute a function by providing the parties the means to compute a single (multiplication) gate, then compile up to general circuits in a "gate-by-gate" fashion. In contrast, (non-private) distributed computation simply requires the parties to exchange their inputs, and perform the computation locally, with no further interaction. This begs the question of whether a high amount of communication is inherently part of *the cost of security*.

Is the communication complexity *of secure computation strongly correlated with the* computational complexity *of the function being computed?*[3]

[3]This elegant phrasing of the question is shamelessly lifted from Yuval Ishai's talk entitled "Private

We present in chapter 3 an overview of the state of the art in regards to providing an answer to this question, excluding our own results which we summarise in section 1.3.

1.3 Our Results on Sublinear-Communication Secure Multiparty Computation

This section describes results previously communicated in [CM21, BCM22, BCM23, CMPR23].

Based on joint work with Elette Boyle, Geoffroy Couteau, Alain Passelègue, and Mahshid Riahinia.

Our goal in this thesis is to make a step towards establishing the communication complexity of secure multiparty computation by understanding in which circumstances the *circuit-size barrier* can be broken. That is to say, we seek to know when "general purpose" secure multiparty computation can be done using an amount of communication which grows only *sublinearly in the circuit-size*. The relevance of this barrier is a historical, and potentially absolute, one: all seminal protocols from the 1980s and 1990s require linear communication, and this remains the state of the art if we wish to provide the strongest security guarantees, i.e. information-theoretic security in the plain model.

Previous works have already demonstrated that communication need not scale linearly with the size of the circuit, under specific computational assumptions or in the correlated randomness model. Our methodology is to draw inspiration from known approaches in the correlated randomness model, which we view simultaneously as fundamental (because of information-theoretic security guarantees) and inherently limited (because the best we can hope for in this model is to understand the *online* communication complexity of secure computation), in order to devise new ways to break the circuit-size barrier. We put forward two ways to track our progress.

> *Sublinear-communication secure computation from "new" assumptions.* As we develop various approaches to breaking the circuit-size barrier, we will need a way to decide how novel each one is. We propose to take extending the set of (standard) assumptions known to imply sublinear-communication secure computation as concrete "proof of conceptual novelty".

This first metric has the merit of being of an objective measure (up to how we decide which assumptions are "standard") by which to quantify our progress. Note however that achieving sublinear-communication secure computation from *new* assumptions[4] is not our direct goal *per se*. Rather, our hope is that by studying protocols from a multitude of structured computational assumptions, we can start abstracting out generic methods, and ultimately converge on minimal assumptions.

Once we develop several ways to break the circuit-size barrier, we can start classifying protocols into different categories depending one how well they solve the problem. In the high-end regime, we consider protocols which provide the strongest security

Informal Retrieval, Part I" at the 10$^{\text{th}}$ Bar-Ilan Winter School on Cryptography.

[4]We emphasise that all the computational assumptions we work with are standard and well-founded; "new" should be taken to mean "not previously known to imply sublinear-communication secure computation".

guarantees[5] or achieve the lowest communication complexity. In the low-end regime, we consider protocols which "only just qualify" as sublinear communication. Beyond simply understanding the landscape of protocols, we hope this classification can help us identify the more fundamental techniques (and ideally minimal assumptions), as opposed to the "heavy hammers", which overshoot the problem.

> *Converging to the circuit-size barrier.* It is usually desirable for a protocol to present extra features (*e.g.* low round or computational complexity, parallelisation, or pre-processability). However, since our goal is to understand the significance of the circuit-size barrier, such additional properties can be viewed as parasitic. As we develop new approaches to breaking the circuit-size barrier, we will try and identify the different ways in which each one overshoots the problem. If we can isolate the source of these additional properties within the assumptions we make, we can (hopefully, heuristically) gain insight into more "minimal" techniques.

We note however that this second way to determine progress is less of a metric and more of a helpful, albeit informal, guideline. Indeed, it would be fairly straightforward to "sabotage" any given sublinear-communication protocol so that it only barely qualifies as such. Still, provided each protocol's conceptual novelty is demonstrated by its reliance on a new assumption, this should remain a good heuristic way to converge on more fundamental methods.

Despite concrete progress, our work is somewhat open-ended in nature[6]. For this reason, we split the presentation of our results in two. To borrow from more experimental sciences, we start by explaining the theory we find best explains our datapoints. Then, to safeguard against the insidious dangers of wishful-thinking-driven overfitting, we present the "raw data", i.e. the main theorems, our conclusions are based on. After all, one can learn from the unsound nearly as well as from the sound, but it is easier when claims are precise enough that one can determine when their author left the path of reason.

[5]For two-party computation, the best we can hope for (in the plain model) is *one-sided statistical security*, providing statistical security for one party and computational security for the other.

[6]Three years being a very short amount of time in the grand scheme of things, we candidly see this as empirically evidence we work on not-too-uninteresting questions.

1.3.1 The Power of Retrospection: A Romanticised Story

> "– Unsound?
> – Well that's a general opinion. Sloppily argued from some highly dubious data.
> – Well then please take [the book] back.
> – Why?
> – Well I have no wish to clutter my mind with useless information.
> – My dear sir, your mind may not have elastic walls, but it does at least possess both an entrance and an exit. Read the book, decide for yourself what to retain. One can learn from the unsound as well as the sound you know. "
>
> "The Saviour of Cripplegate Square", *The Further Adventures of Sherlock Holmes*, written by Bert Coules.

In 2009, Gentry [Gen09] demonstrated the feasibility of fully homomorphic encryption, which led to optimal-communication secure multiparty computation in the computational setting [DFH12, AJL+12]. For seven years, this remained the only avenue for breaking the circuit-size barrier, and in particular instantiations were restricted to lattice-based assumptions. That is, until Boyle, Gilboa, and Ishai [BGI16a] demonstrated that sublinear-communication secure two-party computation does not require FHE. The introduction of homomorphic secret sharing opened the way for breaking the circuit-size barrier under DDH [BGI16a] and later DCR [FGJS17, OSY21, RS21] (but using essentially te same blueprint). However it seems that a fundamentally new approach is required if we want to move away from computational assumptions with a rich algebraic structure.[7] In search for new insights, we turn to the correlated randomness model for inspiration, as it is the only model in which we can achieve low-communication secure multiparty computation with information-theoretic security guarantees.

First Insight: Feasibility of offline-online sublinear-communication secure multiparty computation. Couteau [Cou19], building chiefly upon the works of Ishai, Kushilevitz, Meldgaard, Orlandi, Paskin-Cherniavsky [IKM+13] and Damgård, Nielsen, Nielsen, Ranellucci [DNNR17], designed a protocol for securely computing any layered[8] circuit with a sublinear amount of communication while only using a polynomial amount of computation and correlated randomness. However, the model "cheats" by only counting online communication, and sweeping under the rug resources involved in generating the setup in the absence of a trusted dealer. Fortunately, Couteau also showed that this setup could be generated from Boyle, Gilboa, and Ishai's [BGI16a] HSS (in the two-party setting), or from FHE (in the multiparty setting). Still, since these assumptions were already known to imply sublinear-communication secure com-

[7] A concrete first goal is to achieve sublinear-communication secure computation from assumptions *e.g.* not known to imply linearly homomorphic encryption. However, the relevance of this barrier is unclear beyond the obvious, and somewhat misleading, semantic link between fully homomorphic encryption and homomorphic secret-sharing.

[8] A circuit is *layered* [GJ11] if all gates and inputs are arranged into layers, such that any wire only connects one layer to the next, but each input may occur multiple times at different layers. A layered circuit is *locally synchronous* [Bel84] if each input occurs exactly once (but at an arbitrary layer). A locally synchronous circuit is *synchronous* [Har77] if all inputs are in the first layer.

putation, one may wonder as to the actual novelty of this approach, when transposed into the plain model (with computational security).

First Contribution: Offline-online sublinear-communication two-party computation "does not require homomorphism". As our first contribution, we show that, in the two-party setting, the correlated randomness used by Couteau's [Cou19] protocol can be generated with sublinear communication under the *quasipolynomial Learning Parity with Noise (LPN)* assumption. Very concretely, this extends the set of assumptions under which the circuit-size barrier is known to be broken. Furthermore, while our techniques could in retrospect be used to build a special-purpose form of HSS, and therefore fall into Boyle et al.'s [BGI16a] framework, we argue that the tools we use are conceptually much weaker. Indeed, from quasipolynomial LPN, we only build a "single-circuit HSS scheme", for which it is crucial at the time of sharing the secret input that the parties know the topology of the circuit they want to evaluate homomorphically. The parties must first agree of a loglog-depth circuit, then share the input, and finally generate shares of that circuit's outputs. In contrast Boyle et al.'s [BGI16a] HSS from DDH (as well as subsequent construction from DCR [FGJS17, OSY21]) allows the parties to first generate input shares and then generate output shares for any (and even many!) log-depth circuits.

Second Contribution: Sublinear-communication two-party computation with one-sided statistical security does not require FHE. It may seem unclear at first just how much our LPN-based single-circuit HSS is different from the DDH- or DCR-based HSS for branching programs. We now put forward evidence of a "conceptual separation" by showing that the latter yields sublinear-communication protocols with strong properties, previously believed to be only achievable from FHE.

From initial inspection, FHE enjoys two competitive advantages over HSS. The first is that the communication complexity of HSS-based protocols (even with HSS for $\mathsf{P/poly}$) is inherently *symmetric* in that communication must scale with the size of both inputs[9], whereas FHE-based protocols can be allowed to scale only with the size of the smaller of the two parties' inputs. The second advantage is that provided the fully homomorphic encryption scheme is *statistically function-hiding*, which many schemes are, then the resulting sublinear-communication secure two-party computation protocol provides *one-sided statistical security* and one-sided computational security.

Our second contribution is to bridge the gap between FHE and HSS by showing that most known HSS schemes for NC^1 enjoy limited *programming* properties on the HSS shares[10], which mean that one of the two HSS shares of an input x can be sampled before knowing x. In turn, this yields sublinear-communication secure two-party computation with one-sided statistical security. Concretely, this is the first positive result for sublinear-communication secure two-party computation with one-sided statistical security from assumptions not known to imply FHE.

Second Insight: Break a circuit into small chunks, then run correlated instances of communication-optimal computation. Using computational tools to instantiate protocols in the correlated randomness model has allowed us to break the

[9]We thank Yuval Ishai for bringing this fact to our attention during the defence of this PhD's initial proposal, thereby planting the seeds for some of this work.

[10]This is an eminently non-black box statement. We observe that most constructions of HSS schemes for branching programmes follow a common "template", which we show to yield these additional properties.

circuit-size barrier in a fundamentally new fashion, but this approach is inherently limited if the goal is to identify the minimal assumptions for doing so. Indeed, by definition the resulting protocol must follow the offline-online paradigm, which does not seem it should be an inherent property of sublinear-communication protocols. To make further progress, we propose to identify the key ideas of existing protocols which achieve low *online* communication (in the correlated randomness model), and try and exploit them *directly* in the plain model.

In the correlated randomness model, the "one-time truth table" protocol of Ishai, Kushilevitz, Meldgaard, Orlandi, Pasking-Cherniavsky [IKM+13] achieves optimal-communication secure computation, but requires doubly exponential (in the input size) computation and correlated randomness. In order to bring the amount of computation down to a polynomial amount—which is of paramount importance if we want to instantiate this protocol with computational security—(but at the cost of increasing communication from optimal to "barely sublinear") Couteau's protocol [Cou19] and Damgaard, Nielsen, Nielsen, Ranellucci's "Tiny Tables" protocol [DNNR17] in the correlated randomness model relies on breaking down a (layered, boolean or arithmetic) circuit into tiny computations, each one small enough that Ishai et al.'s [IKM+13] protocol (or an arithmetic alternative) can be used with polynomial computation (in the overall circuit's size). We note that the gates of the layered circuits are not partitioned into smaller circuits: rather, due to the way the circuit is broken into chunks, there is some overlap. As a result it is not enough to perform these smaller computations independently, as even with optimal-communication subroutines, the overall protocol would not be sublinear. Instead, these smaller computations must be treated as *correlated*, and redundancies in communication across them must be exploited. Without going into any detail here, the correlated randomness model is well equipped to handle this issue.

Third Contribution: Sublinear-communication two-party computation from Correlated Symmetric PIR. Our third contribution is to introduce and instantiate the notion of *correlated symmetric PIR* (correlated SPIR), and show that it yields sublinear-communication secure two-party computation. Correlated SPIR is a form of batch SPIR[11] with correlated queries. The server holds k databases, and the client wishes to query one element per database, but these queries are not independent: rather, the client holds a small input w (which we refer to as the "entropy" used to generate the correlated queries) and each query is a public function of w. We impose that the total upload communication, from client to server, scale with the size of $|w|$, which may be significantly smaller than the total size $k \cdot n$ of all the queries. Without going into too much detail here, the way we use correlated SPIR to get low-communication 2PC of loglog-depth boolean circuits (and in turn sublinear-communication 2PC of layered boolean circuits) is analogous to Couteau's protocol in the randomness model, except using the eminently computational tool of single-server PIR as a substitute for one-time truth tables: Alice hardcodes her input in the circuit and exploits log-locality to break the resulting circuit into polynomial-size truth tables, and Bob then uses correlated SPIR to retrieve the appropriate values from these truth

[11] Recall that a (single-server) *private information retrieval* (PIR) [CGKS95, KO97] is a protocol which allows a client to privately retrieve an item from a database, held by a server, using an amount of communication which is sublinear in the size of the database. Privacy here means the server should not learn the index of the element retrieved by the client; we refer the *symmetric private information retrieval* (SPIR) if there is an additional privacy requirement that the client should not learn anything about the rest of the database.

tables. The main technical difficulty in realising correlated SPIR is keeping upload communication proportional to Bob's "query-entropy", in this case Bob's input to the overall computation.

We then show how to achieve correlated SPIR (for the specific class of correlations needed for our application to go through) from a new primitive we coin as *decomposable rate-1 batch OT*. This is rate-1 batch OT where the sender message for the batch is equipped with a certain *decomposability* property: roughly speaking any subset of the rate-1 sender message for the batch can be correctly decoded to the corresponding subset of the payload. Fortunately, Brakerski-Branco-Döttling-Pu's [BBDP22] recent construction of rate-1 batch OT from LPN as well as any of QR, DDH, DCR, or LWE, can be shown to be decomposable. In turn, this means we have found a new set of standard assumptions under which we break the circuit-size barrier for secure two-party computation, namely QR+LPN. Note that the parameter regime of this flavour of LPN, while technically incomparable to quasipolynomial LPN, is significantly more standard.

From a certain point of view, correlated SPIR is an optimal-communication special-purpose protocol for securely computing the function

$$\begin{array}{rcl} ((\{0,1\}^n)^k \times \{0,1\}^\ell) & \to & \{0,1\}^k \\ ((x_1,\ldots,x_k),w) & \mapsto & (x_1[Q_1(w)],\ldots,x_k[Q_k(w)]) \end{array}$$

where the functions $Q_1,\ldots,Q_k$ (used to generate correlated queries from a common "entropy" w) are public. As such, our protocol can be seen as achieving a trade-off between functionality (from special-purpose computation to general computation of layered circuits) and communication-complexity (from optimal to "barely sublinear"). This constitutes an intriguing first step towards establishing the minimal assumptions for sublinear-communication secure two-party computation.

Fourth Contribution: Breaking the Two-Party Barrier for Sublinear-Communication Secure Computation, without FHE. Summarising, our two low-communication secure two-party computation protocols follow the template of breaking a loglog-depth circuit into tiny computations [DNNR17, Cou19]. These computations are correlated in the sense that each party's input to the computation is a subset of some global input to the entirety of the computation. Where our protocols diverge is in how these tiny computations are performed (while exploiting the correlated nature of these computations to avoid redundancies in communication):

- In our first protocol, the parties are able to generate the correlated randomness used in Couteau's protocol [Cou19] by using a pseudorandom correlation generator (PCG) for a circuit-dependent correlation (which can be seen as a dual form of single-circuit HSS).

  In order to continue this presentation of our results, we need to provide more details of Couteau's protocol at this point. We refer the reader to section 3.2.3 for a more in-depth description of both this protocol and the underlying paradigm of *circuit randomisation*.

  More precisely each output gate of the loglog-depth circuit is computed by a function with a polynomial-size truth table, whose input is a log-sized subset of the inputs: $C(x) = (y_1,\ldots,y_m) = (f_1(x[S_1]),\ldots,f_m(x[S_m]))$, where $|S_1|,\ldots,|S_m| \leq \log|C|$. The (pseudo)random correlated material the parties generate is comprised of an additive shares of a (pseudo)random mask r for their joint input x to the loglog-depth computation, as well as additive shares of the truth tables of

the functions $g_i(\cdot) \coloneqq f_i(\cdot - r[S_i])$ for $i \in [m]$, which are of polynomial size. To perform theses computations, the parties reconstruct the shifted value $c \coloneqq x + r$, and finally locally compute a share of $C(x)$ by selecting the entry indexed by $c[S_i]$ in the share of the truth table of g_i (which can be seen as the truth table of a "share of the function" g_i) for $i \in [m]$. The parties can then optionally reconstruct the output $C(x)$ after broadcasting these shares.

- In our second protocol, one party first hardcodes their input into the f_i and prepares the corresponding truth tables, then the other party retrieves the entry in each table corresponding to its own input by using correlated SPIR.

These approaches seem *a priori* stuck at the two-party barrier, but for different reasons. The bottleneck of the first approach is the existence of a sufficiently expressive PCG or HSS (with security against all-but-one corruptions) for more than two parties (from assumptions not already known to imply FHE). In fact, no instantiation of such a primitive was known until late into this candidate's PhD studies. Thankfully, Chillotti, Orsini, Scholl, Smart, van Leeuwen [COS+22] provided a construction of four-party HSS for constant-depth circuits from DCR. While this is not enough to yield sublinear-communication four-party computation on its own, we showed that their techniques can be adapted to build single-circuit HSS for any loglog-depth circuit from DCR or from DDH. In particular, this yields the first construction of sublinear-communication four-party computation without FHE. However, this result is a relatively minor contribution, and not what we refer to as *breaking the two-party barrier for sublinear-communication secure multiparty computation.*

The bottleneck of the second approach is the fact that correlated SPIR is a two-party primitive, and it appears unclear how it could be generalised to more parties. Intriguingly, our two seemingly orthogonal approaches in the two-party setting can be combined to obtain sublinear communication secure multiparty computation (for more than two parties). More precisely we provide a protocol for sublinear-communication secure $(N+1)$-party computation for loglog-depth circuits from single-circuit N-party HSS for any loglog-depth circuit and (the two-party primitive of) correlated SPIR. The idea is to have N parties break the loglog-depth circuit C into tiny functions $f_1, \ldots, f_m$ with polynomial-size truth tables. They then use N-party HSS to generate shares of the truth tables of these functions but with all their N inputs hardcoded. The last party, who did not participate in this first procedure, then uses correlated SPIR with each of the parties to retrieve the appropriate output shares, corresponding to its own input (observe that once the first N parties hardcode their inputs into the functions f_i, the resulting function only expects the last party's input). This approach can be instantiated for the three-party setting (i.e. $N = 2$) from quasipolynomial LPN+QR, DDH+LPN, DCR+LPN, or LWE+LPN. The last three sets of assumptions can also be used to instantiate the (four or) five-party setting (i.e. $N = 4$).

To a certain extent our final protocol can be seen as extending HSS-based techniques for breaking the circuit-size barrier, which were mostly stuck at the two-party barrier, to the multiparty setting, which was previously achieved only from the heavy hammer of FHE.

1.3.2 The Historical Perspective: The "Raw Data"

We now provide a standalone description of our each of our main results, without the distractions of an overarching story.

Breaking the Circuit-Size Barrier Under Quasi-Polynomial LPN [CM21]. In this work we introduce a new (circuit-dependent) homomorphic secret sharing (HSS) scheme for any $\log / \log\log$-local circuit, with communication proportional only to the width of the circuit and polynomial computation, which is secure assuming the super-polynomial hardness of learning parity with noise (LPN). At the heart of our new construction is a pseudorandom correlation generator (PCG) which allows two parties to locally stretch short seeds into pseudorandom instances of an arbitrary $\log / \log\log$-local additive correlation.

Our main application, and the motivation behind this work, is a generic two-party secure computation protocol for every layered (boolean or arithmetic) circuit of size s with total communication $O(s/\log\log s)$ and polynomial computation, assuming the super-polynomial hardness of the standard learning parity with noise assumption (a circuit is layered if its nodes can be partitioned in layers, such that any wire connects adjacent layers). This expands the set of assumptions under which the "circuit-size barrier" can be broken, for a large class of circuits. The strength of the underlying assumption is tied to the sublinearity factor: we achieve communication $\mathcal{O}(s/k(s))$ under the $s^{2^{k(s)}}$-hardness of LPN, for any $k(s) \leq (\log\log s)/4$.

Previously, the set of assumptions known to imply a PCG for correlations of degree $\omega(1)$ or generic secure computation protocols with sublinear communication was restricted to LWE, DDH, and a circularly secure variant of DCR.

Constrained Pseudorandom Functions from Homomorphic Secret-Sharing [CMPR23]. We propose and analyse a simple strategy for constructing 1-key constrained pseudorandom functions (CPRFs) from homomorphic secret sharing. In the process, we obtain the following contributions. First, we identify desirable properties for the underlying HSS scheme for our strategy to work. Second, we show that (most of) recent existing HSS schemes satisfy these properties, leading to instantiations of CPRFs for various constraints and from various assumptions. Notably, we obtain the first (1-key selectively secure, private) CPRFs for inner-product and (1-key selectively secure) CPRFs for NC^1 from the DCR assumption, and more. Lastly, we revisit two applications of HSS, equipped with these additional properties, to secure computation: we obtain secure computation in the silent preprocessing model with one party being able to precompute its whole preprocessing material before even knowing the other party, and we construct one-sided statistically secure computation with sublinear communication for restricted forms of computation.

Sublinear Secure Computation from New Assumptions [BCM22]. Secure computation enables mutually distrusting parties to jointly compute a function on their secret inputs, while revealing nothing beyond the function output. A long-running challenge is understanding the required communication complexity of such protocols—in particular, when communication can be *sublinear* in the circuit representation size of the desired function. For certain functions, such as Private Information Retrieval (PIR), this question extends to even sublinearity in the input size.

We develop new techniques expanding the set of computational assumptions for sublinear communication in both settings:

- **Circuit size.** We present sublinear-communication protocols for secure evaluation of general layered circuits, given any 2-round rate-1 batch oblivious transfer (OT) protocol with a particular "decomposability" property. In particular, this condition can be shown to hold for the recent batch OT protocols of (Brakerski et al. Eurocrypt 2022), in turn yielding a new sublinear secure computation feasibility result: from Quadratic Residuosity (QR) together with polynomial-noise-rate Learning Parity with Noise (LPN).

  Our approach constitutes a departure from existing paths toward sublinear secure computation, all based on fully homomorphic encryption or homomorphic secret sharing.

- **Input size.** We construct single-server PIR based on the Computational Diffie-Hellman (CDH) assumption, with *polylogarithmic* communication in the database input size n. Previous constructions from CDH required communication $\Omega(n)$. In hindsight, our construction comprises of a relatively simple combination of existing tools from the literature.

Sublinear-Communication Secure Multiparty Computation does not require FHE [BCM23]. Secure computation enables mutually distrusting parties to jointly compute a function on their secret inputs, while revealing nothing beyond the function output. A long-running challenge is understanding the required communication complexity of such protocols—in particular, when communication can be *sublinear* in the circuit representation size of the desired function.

Significant advances have been made affirmatively answering this question within the *two-party* setting, based on a variety of structures and hardness assumptions. In contrast, in the *multi-party* setting, only one general approach is known: using Fully Homomorphic Encryption (FHE).

This remains the state of affairs even for just three parties, with two corruptions.

We present a framework for achieving secure sublinear-communication $(N+1)$-party computation, building from a particular form of Function Secret Sharing for only N parties. In turn, we demonstrate implications to sublinear secure computation for various function classes in the 3-party and 5-party settings based on an assortment of assumptions not known to imply FHE.

1.4 Other Selected Contributions

This section describes joint work with Marshall Ball, Alexander Bienstock, Elette Boyle, Chris Brzuska, Ran Cohen, Christoph Egger, Yuval Ishai, Pihla Karanko, Lisa Kohl, Tal Malkin, Tal Moran, Robert Robere, and Gal Yehuda.

For completeness, we now mention our other results obtained during the course of our PhD studies. While they have *a priori* little link to the question of sublinear-communication secure computation, we acknowledge these works and their co-authors for bringing us broader insights into cryptography. As such, they inexorably contributed to our understanding of the main topic of this thesis, and we therefore include them.

Topology-Hiding Communication from Minimal Assumptions [BBC+20,BBC+23]. *Topology-hiding broadcast* (THB) enables parties communicating over an incomplete network to broadcast messages while hiding the topology from within a given class of graphs. THB is a central tool underlying general *topology-hiding secure computation* (THC) (Moran et al. TCC'15). Although broadcast is a privacy-free task, it was recently shown that THB for certain graph classes necessitates computational assumptions, even in the semi-honest setting, and even given a single corrupted party.

In this work we investigate the minimal assumptions required for topology-hiding communication: both *Broadcast* or *Anonymous Broadcast* (where the broadcaster's identity is hidden). We develop new techniques that yield a variety of necessary and sufficient conditions for the feasibility of THB/THAB in different cryptographic settings: information theoretic, given existence of key agreement, and given existence of oblivious transfer. Our results show that feasibility can depend on various properties of the graph class, such as *connectivity*, and highlight the role of different properties of topology when kept hidden, including *direction*, *distance*, and/or *distance-of-neighbors* to the broadcaster.

An interesting corollary of our results is a dichotomy for THC with a public number of at least three parties, secure against one corruption: information-theoretic feasibility if all graphs are 2-connected; necessity and sufficiency of key agreement otherwise.

Towards Founding Topology-Hiding Computation on Oblivious Transfer [BBKM23]. *Topology-Hiding Computation (THC)* enables parties to securely compute a function on an incomplete network without revealing the network topology. It is known that secure computation on a *complete* network can be based on oblivious transfer (OT), even if a majority of the participating parties are corrupt. In contrast, THC in the dishonest majority setting is only known from assumptions that imply (additively) homomorphic encryption, such as Quadratic Residuosity, Decisional Diffie-Hellman, or Learning With Errors.

In this work we move towards closing the gap between MPC and THC by presenting a protocol for THC on general graphs secure against all-but-one semi-honest corruptions from *constant-round constant-overhead secure two-party computation*. Our protocol is therefore the first to achieve THC on arbitrary networks without relying on assumptions with rich algebraic structure.

As a technical tool, we introduce the notion of *locally simulatable MPC*, which we believe to be of independent interest.

On Low-End Obfuscation and Learning [BIM+23]. Most recent works on cryptographic obfuscation focus on the high-end regime of obfuscating *general circuits* while guaranteeing *computational indistinguishability* between functionally equivalent circuits. Motivated by the goals of simplicity and efficiency, we initiate a systematic study of "low-end" obfuscation, focusing on simpler representation models and information-theoretic notions of security. We obtain the following results.

- **Positive results via "white-box" learning.** We present a general technique for obtaining perfect indistinguishability obfuscation from exact learning algorithms that are given restricted access to the representation of the input function. We demonstrate the usefulness of this approach by obtaining simple obfuscation for decision trees and multilinear read-k arithmetic formulas.
- **Negative results via PAC learning.** A *proper obfuscation* scheme obfuscates programs from a class $\mathcal{C}$ by programs from the same class. Assuming the exis-

tence of one-way functions, we show that there is no proper indistinguishability obfuscation scheme for k-CNF formulas for any constant $k \geq 3$; in fact, even obfuscating 3-CNF by k-CNF is impossible. This result applies even to computationally secure obfuscation, and makes an unexpected use of PAC learning in the context of *negative* results for obfuscation.

- **Separations.** We study the relations between different information-theoretic notions of indistinguishability obfuscation, giving cryptographic evidence for separations between them.

New Random Oracle Instantiations from Extremely Lossy Functions (ELFs) [BEC+23]. We instantiate two random oracle (RO) transformations using Zhandry's (Crypto'16) extremely lossy function (ELF) technique. Firstly, using ELFs and indistinguishabililty obfuscation (iO), we instantiate a modified version of the Fujisaki-Okamoto (FO) transform which upgrades a public-key encryption scheme (PKE) from indistinguishability under chosen plaintext attacks (IND-CPA) to indistinguishability under chosen ciphertext attacks (IND-CCA).

We side-step a prior uninstantiability result for FO by Brzuska, Farshim, and Mittelbach (TCC'15) by (1) hiding the randomness from the (potentially ill-designed) IND-CPA encryption scheme and (2) embedding an additional secret related to the hash-function into the secret-key of the IND-CCA-secure PKE, an idea brought forward by Murphy, O'Neill, Zaheri (Asiacrypt 2022) who also instantiate a modified FO variant also under ELFs and iO for the class of *lossy PKE*. Our transformation applies to all PKE which can be inverted given their randomness.

Secondly, we instantiate the hash-then-evaluate paradigm for pseudorandom functions (PRFs), $\mathsf{PRF}(k, x) := \mathsf{wPRF}(k, \mathsf{RO}(x))$, using ELFs together with minicrypt primitives, by observing that many weak PRFs are plausibly also secure under *pseudorandom* inputs, since analogous cryptanalysis applies. Our simple transformation applies to the entire family of PRF-style functions. Specifically, we obtain results for oblivious PRFs, which are a core building block for password-based authenticated key exchange (PAKE) and private set intersection (PSI) protocols, and we also obtain results for pseudorandom correlation functions (PCF), which are a key tool for silent oblivious transfer (OT) extension.

Chapter 2

Preliminaries

2.1 Notations

Notations for Chapter 4 We say that a function $\mathsf{negl}\colon \mathbb{N} \to \mathbb{R}^+$ is *negligible* if it vanishes faster than every inverse polynomial. For two families of distributions $X = \{X_\lambda\}$ and $Y = \{Y_\lambda\}$ indexed by a security parameter $\lambda \in \mathbb{N}$, we write $X \overset{\text{c}}{\approx} Y$ if X and Y are *computationally indistinguishable* (i.e. any family of circuits of size $\mathsf{poly}(\lambda)$ has a negligible distinguishing advantage), $X \overset{\text{s}}{\approx} Y$ if they are *statistically indistinguishable* (i.e. the above holds for arbitrary, unbounded, distinguishers), and $X \equiv Y$ if the two families are identically distributed.

We usually denote matrices with capital letters (A, B, C) and vectors with bold lowercase $(\vec{x}, \vec{y})$. By default, vectors are assumed to be column vectors. If $\vec{x}$ and $\vec{y}$ are two (column) vectors, we use $\vec{x}||\vec{y}$ to denote the (column) vector obtained by their concatenation. We write $\vec{x} \otimes \vec{y}$ to denote the tensor product between $\vec{x}$ and $\vec{y}$, i.e., the vector of length $n_x n_y$ with coordinates $x_i y_j$ (where n_x is the length of $\vec{x}$ and n_y is the length of $\vec{y}$). We write $\vec{x}^{\otimes 2}$ for $\vec{x} \otimes \vec{x}$, and more generally, $\vec{x}^{\otimes n}$ for the *n-th tensor power* of $\vec{x}$, $\vec{x} \otimes \vec{x} \otimes \cdots \otimes \vec{x}$. Given a vector $\vec{x}$ of length $|\vec{x}| = n$, the notation $\mathsf{HW}(x)$ denotes the Hamming weight $\vec{x}$, i.e. , the number of its nonzero entries. Let k be an integer. We let $\{0,1\}^k$ denote the set of bitstrings of length k. For two strings (x, y) in $\{0,1\}^k$, we denote by $x \oplus y$ their bitwise xor.

Notations for Chapter 5 We use λ to denote the security parameter. For a natural integer $n \in \mathbb{N}$, the set $\{0, 1, \ldots, n-1\}$ is denoted by $[n]$. We mostly use bold lowercase letters (e.g., $\mathbf{r}$) to denote vectors. For a vector $\mathbf{r} = (r_1, \ldots, r_n)$, the vector $(g^{r_1} \ldots, g^{r_n})$ is sometimes denoted by $g^{\mathbf{r}}$. We write $\mathsf{poly}(\lambda)$ to denote an arbitrary polynomial function. We denote by $\mathsf{negl}(\lambda)$ a negligible function in λ, and PPT stands for probabilistic polynomial-time. For a finite set S, we write $x \xleftarrow{\$} S$ to denote that x is sampled uniformly at random from S. For an algorithm $\mathcal{A}$, we denote by $y \leftarrow \mathcal{A}(x)$ the output y after running $\mathcal{A}$ on input x.

Notations for Chapter 6 Throughout the chapter, $[\vec{v}]_I$ denotes the subvector of $\vec{v}$ induced by set of indices I.

Notations for Chapter 7 N denotes a number of parties (but the total number of parties is sometimes $N + 1$). The number of inputs and outputs of an arithmetic circuit are denoted n and m respectively. If f is a function, $\tilde{f}$ is used to describe a polynomial-size description of f.

By default, vectors are assumed to be column vectors. If $\vec{x}$ and $\vec{y}$ are two (column) vectors, we use $\vec{x}||\vec{y}$ to denote the (column) vector obtained by their concatenation. We write $\vec{x}\otimes\vec{y}$ to denote the tensor product between $\vec{x}$ and $\vec{y}$, i.e., the vector of length $n_x n_y$ with coordinates $x_i y_j$ (where n_x is the length of $\vec{x}$ and n_y is the length of $\vec{y}$). We write $\vec{x}^{\otimes 2}$ for $\vec{x}\otimes\vec{x}$, and more generally, $\vec{x}^{\otimes k}$ for the *k-th tensor power* of $\vec{x}$, $\vec{x}\otimes\vec{x}\otimes\cdots\otimes\vec{x}$ (k times). Observe that evaluating a degree-d n-variate polynomial on some size-n vector $\vec{x}$ can be done by computing the inner product of the vector of coefficients of the polynomial with $\vec{x}^{\otimes d}\|\ldots\|\vec{x}^{\otimes 1}\|1)$. Indeed, each $\vec{x}^{\otimes k}$ corresponds the vector of every degree-k monomial in the coordinates of $\vec{x}$.

For $k\in\mathbb{N}$, $[k]$ denotes the set $\{1,\ldots,k\}$, $[0,k]$ denotes the set $\{0,1,\ldots,k\}$, and $\mathfrak{S}_k$ denotes the symmetric group of order k, i.e. the set of all permutations on $[k]$. If A and B are sets, A^B denotes the set of all functions from A to B.

We say that a function $\mathsf{negl}\colon\mathbb{N}\to\mathbb{R}^+$ is *negligible* if it vanishes faster than every inverse polynomial. For two families of distributions $X=\{X_\lambda\}$ and $Y=\{Y_\lambda\}$ indexed by a security parameter $\lambda\in\mathbb{N}$, we write $X\overset{c}{\approx}Y$ if X and Y are *computationally indistinguishable* (i.e. any family of circuits of size $\mathsf{poly}(\lambda)$ has a negligible distinguishing advantage), $X\overset{s}{\approx}Y$ if they are *statistically indistinguishable* (i.e. the above holds for arbitrary, unbounded, distinguishers), and $X\equiv Y$ if the two families are identically distributed.

2.2 Universal Composability

We refer the reader to [Can01] for details on the universal composability framework. The framework is based on the real/ideal paradigm for arguing about the security of a protocol. We say that a protocol π UC-realises (with computational security) an ideal functionality $\mathcal{F}$ in the presence of static semi-honest adversary corrupting at most t parties, if for any *p.p.t.* static semi-honest t-adversary $\mathcal{A}$ and any *p.p.t.* environment $\mathcal{Z}$, there exists a p.p.t. ideal-model t-adversary Sim such that the output distribution of $\mathcal{Z}$ in the ideal-model computation of $\mathcal{F}$ with Sim is *computationally indistinguishable* from its output distribution in the real-model execution of π with $\mathcal{A}$. The composition theorem of [Can01] states the following.

Theorem 1 (Composition Theorem [Can01], informal)**.** *Let ρ be a protocol that UC-realizes $\mathcal{F}$ in the presence of adaptive semi-honest t-adversaries, and let π be a protocol that UC-realizes $\mathcal{G}$ in the $\mathcal{F}$-hybrid model in the presence of adaptive semi-honest t-adversaries. Then, for any* p.p.t. *adaptive semi-honest t-adversary $\mathcal{A}$ and any* p.p.t. *environment $\mathcal{Z}$, there exists a* p.p.t. *adaptive semi-honest t-adversary* Sim *in the $\mathcal{F}$-hybrid model such that the output distribution of $\mathcal{Z}$ when interacting with the protocol π and* Sim *is computationally indistinguishable from its output distribution when interacting with the protocol π^ρ (where every call to $\mathcal{F}$ is replaced by an execution of ρ) and $\mathcal{A}$ in the real model.*

2.3 Cryptographic Primitives

2.3.1 Homomorphic Secret Sharing (HSS)

We consider here a definition of Homomorphic Secret Sharing with simulation-based security guarantee (as it is the most natural to use for chapter 7). This notion is equivalent to the natural indistinguishability-based definition (where simulation takes place by simply sharing a fixed input of appropriate length, *e.g.* $0^{n(\lambda)}$).

Definition 1 (Homomorphic Secret Sharing, [BGI16a])**.** *An N-party* homomorphic secret-sharing (HSS) *scheme (with additive reconstruction) for a class $\mathcal{F}$ of functions over a finite field $\mathbb{F}$ is a pair of algorithms* $\mathsf{HSS} = (\mathsf{HSS.Share}, \mathsf{HSS.Eval})$ *with the following syntax and properties:*

- $\mathsf{Share}(1^\lambda, x)$*: On input 1^λ (the security parameter) and $x \in \mathbb{F}^{n(\lambda)}$ (the input), the* sharing algorithm Share *outputs N input shares $(x^{(1)}, \ldots, x^{(N)})$.*
- $\mathsf{Eval}(i, f, x^{(i)})$*: On input $i \in [N]$ (the party index), $f \in \mathcal{F}$ (the function to be homomorphically evaluated, implicitly assumed to specify input and output lengths n, m), and $x^{(i)}$ (the i^{th} input share), the* evaluation algorithm Eval *outputs the i^{th} output share $y^{(i)} \in \mathbb{F}^m$.*
- ***Correctness:*** *For any 1^λ, input $x \in \mathbb{F}^{n(\lambda)}$, and any function $f \in \mathcal{F}$,*

$$\Pr\left[y^{(1)} + \cdots + y^{(N)} = f(x) \ : \ \begin{array}{l} (x^{(1)}, \ldots, x^{(N)}) \xleftarrow{\$} \mathsf{HSS.Share}(1^\lambda, x) \\ y^{(i)} \xleftarrow{\$} \mathsf{HSS.Eval}(i, f, x^{(i)}), \ i = 1 \ldots N \end{array} \right] = 1 \ .$$

- ***Security:*** *For every set of corrupted parties $\mathcal{D} \subsetneq [N]$, there exists a probabilistic polynomial-time algorithm* $\mathsf{Sim}^{\mathsf{HSS}}$ *(a* simulator*), such that for every sequence of inputs $x_1, x_2, \cdots \in \mathbb{F}^{n(\lambda)}$ the outputs of the following experiments* $\mathsf{Real}^{\mathsf{HSS}}$ *and* $\mathsf{Ideal}^{\mathsf{HSS}}$ *are computationally indistinguishable:*
  - $\mathsf{Real}^{\mathsf{HSS}}(1^\lambda)$: $(x^{(1)}, \ldots, x^{(N)}) \xleftarrow{\$} \mathsf{HSS.Share}(1^\lambda, x_\lambda)$*; Output* $(x^{(i)})_{i \in \mathcal{D}}$.
  - $\mathsf{Ideal}^{\mathsf{HSS}}(1^\lambda)$: *Output* $\mathsf{Sim}^{\mathsf{HSS}}(1^\lambda, 1^N, 1^n)$.

Remark 1 (Compact Single-Function HSS)**.** *A* single-function HSS *is an HSS scheme for a singleton function class. Let $\mathcal{F}$ be a (not necessarily singleton) function class. We say there exists* compact single-function HSS for any function in $\mathcal{C}$*, if for every $f \colon \mathbb{F}^n \to \mathbb{F}^m \in \mathcal{F}$ there exists an HSS scheme HSS_f for $\{f\}$ such that the circuit-size of $\mathsf{HSS}_f.\mathsf{Share}$ is a fixed polynomial in n (and otherwise independent of f).*

This notion can be seen as a weakening of compact HSS for $\mathcal{C}$ where the function to be homomorphically evalluated is known when running the sharing algorithm.

Definition 2 (Las Vegas HSS)**.** *A* Las Vegas *N-party homomorphic secret-sharing scheme with additive reconstruction is defined as above, with the following modification:*

1. *The algorithm* Eval *takes as input a* failure bound *δ, and additonally outputs a confidence flag $\mathsf{flag}_b \in \{\bot, \top\}$ to indicate full confidence ($\top$) or a possibility of failure ($\bot$).* Eval *can run in time polynomial in its input length and in $1/\delta$.*
2. *The correctness notion is relaxed to the following notion of* Las Vegas correctness*: for every input $x \in \mathbb{F}^n$, function $f \in \mathcal{F}$ with input length n, and failure bound $\delta > 0$, we have:*

$$\Pr\left[\exists i \leq N, (\mathsf{flag}_i = \bot)\right] \leq \delta,$$
$$\textit{and } \Pr[(\exists i \leq N, (\mathsf{flag}_i = \top) \wedge (\oplus_{i \leq n} y^{(i)} \neq f(x))] = 0,$$

where the probability is taken over the coins of Gen *and* $\mathsf{Eval}(\cdot, \cdot, \cdot, \delta)$*. We implicitly assume each execution of* Eval *to take an additional nonce input, which enables different invocations to have (pseudo)-independent failure probabilities. (See [BGI16a] for discussion.)*

2.3.2 Function Secret Sharing (FSS)

Informally, an FSS scheme for a class of functions $\mathcal{C}$ is a pair of algorithms $\mathsf{FSS} = (\mathsf{FSS.Gen}, \mathsf{FSS.Eval})$ such that:

- $\mathsf{FSS.Gen}$ given a function $f \in \mathcal{C}$ outputs a pair of keys (K_0, K_1);
- $\mathsf{FSS.Eval}$, given K_b and input x, outputs y_b such that y_0 and y_1 form additive shares of $f(x)$.

The security requirement is that each key K_b computationally hide f, except for revealing the input and output domains of f. Formally, we follow the function secret sharing definition of [BGI16b], for the specific leakage function which reveals the input and output domain sizes $(1^n, 1^m)$ of the secret function.

Definition 3 (Function Secret Sharing (FSS))**.** *An N-party* Function Secret-Sharing (FSS) *scheme (with additive reconstruction) for a function family $\mathcal{F}$ is a pair of algorithms $\mathsf{FSS} = (\mathsf{FSS.Gen}, \mathsf{FSS.Eval})$ with the following syntax and properties:*

- $\mathsf{Gen}(1^\lambda, \tilde{f})$ *is a probabilistic polynomial-time* key generation algorithm*, which on input 1^λ (a security parameter) and $\tilde{f} \in \{0,1\}^\star$ (the description of some function $f\colon \{0,1\}^n \to \{0,1\}^m \in \mathcal{F}$), outputs an N-tuple of keys $(k_1, \ldots, k_N)$. Each key is assumed to contain 1^n and 1^m.*
- $\mathsf{Eval}(i, k_i, x)$ *is a deterministic polynomial-time* evaluation algorithm*, which on input $i \in [N]$ (the party index), k_i (a key defining $f_i\colon \{0,1\}^n \to \{0,1\}^m$), and $x \in \{0,1\}^n$ (an input for f_i), outputs a value $y_i \in \{0,1\}^m$ (the value of $f_i(x)$, the i^{th} share of $f(x)$).*
- ***Correctness:*** *For all $\lambda \in \mathbb{N}$, all $f \in \mathcal{F}$ (described by $\tilde{f}$), and all $x \in \{0,1\}^n$,*

$$\Pr\left[y_1 + \cdots + y_N = f(x) \ : \ \begin{array}{l} (k_1, \ldots, k_N) \xleftarrow{\$} \mathsf{FSS.Gen}(1^\lambda, \tilde{f}) \\ y_i \leftarrow \mathsf{FSS.Eval}(i, k_i, x),\ i = 1 \ldots N \end{array} \right] = 1 \ .$$

- ***Security:*** *For every set of corrupted parties $\mathcal{D} \subsetneq [N]$, there exists a probabilistic polynomial-time algorithm $\mathsf{Sim}^{\mathsf{FSS}}$ (a* simulator*), such that for every sequence of functions $f_1, f_2, \cdots \in \mathcal{F}$ (described by $\tilde{f}_1, \tilde{f}_2, \ldots$), the outputs of the following experiments $\mathsf{Real}^{\mathsf{FSS}}$ and $\mathsf{Ideal}^{\mathsf{FSS}}$ are computationally indistinguishable:*
  - $\mathsf{Real}^{\mathsf{FSS}}(1^\lambda)$: $(k_1, \ldots, k_N) \xleftarrow{\$} \mathsf{Gen}(1^\lambda, \tilde{f}_\lambda)$*; Output* $(k_i)_{i \in \mathcal{D}}$.
  - $\mathsf{Ideal}^{\mathsf{FSS}}(1^\lambda)$: *Output* $\mathsf{Sim}^{\mathsf{FSS}}(1^\lambda, 1^N, 1^n, 1^m)$.

Our applications of FSS sometimes require applying the evaluation algorithm on *all inputs*. Following [BGI16b, BCGI18, BCG+19b, BCG+19a], given an FSS scheme $(\mathsf{FSS.Gen}, \mathsf{FSS.Eval})$, we denote by $\mathsf{FSS.FullEval}$ an algorithm which, on input a bit b, and an evaluation key K_b (which defines the input domain I), outputs a list of $|I|$ elements of $\mathsf{BilinearGen}$ corresponding to the evaluation of $\mathsf{FSS.Eval}(b, K_b, \cdot)$ on every input $x \in I$ (in some predetermined order). Below, we recall some results from [BGI16b] on FSS schemes for useful classes of functions.

2.3.2.1 Distributed Point Functions

A distributed point function (DPF) [GI14] is an FSS scheme for the class of point functions $f_{\alpha,\beta} : \{0,1\}^\ell \to \mathsf{BilinearGen}$ which satisfies $f_{\alpha,\beta}(\alpha) = \beta$, and $f_{\alpha,\beta}(x) = 0$ for any $x \neq \alpha$. A sequence of works [GI14, BGI15, BGI16b] has led to highly efficient constructions of DPF schemes from any pseudorandom generator (PRG).

Theorem 2 (PRG-based DPF [BGI16b]). *Given a PRG $G : \{0,1\}^\lambda \to \{0,1\}^{2\lambda+2}$, there exists a DPF for point functions $f_{\alpha,\beta} : \{0,1\}^\ell \to \mathsf{BilinearGen}$ with key size $\ell \cdot (\lambda + 2) + \lambda + \lceil \log_2 |\mathsf{BilinearGen}| \rceil$ bits. For $m = \lceil \frac{\log |\mathsf{BilinearGen}|}{\lambda+2} \rceil$, the key generation algorithm* Gen *invokes G at most $2(\ell+m)$ times, the evaluation algorithm* Eval *invokes G at most $\ell + m$ times, and the full evaluation algorithm* FullEval *invokes G at most $2^\ell(1+m)$ times.*

2.3.2.2 FSS for Multi-Point Functions

A k-point function evaluates to 0 everywhere, except on k specified points. When specifying multi-point functions we often view the domain of the function as $[n]$ for $n = 2^\ell$ instead of $\{0,1\}^\ell$.

Definition 4 (Multi-Point Function [BCGI18]). *An (n,t)-multi-point function over an abelian group $(\mathsf{BilinearGen}, +)$ is a function $f_{S,\vec{y}} : [n] \to \mathsf{BilinearGen}$, where $S = (s_1, \cdots, s_t)$ is an ordered subset of $[n]$ of size t and $\vec{y} = (y_1, \cdots, y_t) \in \mathsf{BilinearGen}^t$, defined by $f_{S,\vec{y}}(s_i) = y_i$ for any $i \in [t]$, and $f_{S,y}(x) = 0$ for any $x \in [n] \setminus S$.*

We assume that the description of S includes the input domain $[n]$ so that $f_{S,\vec{y}}$ is fully specified. A *Multi-Point Function Secret Sharing* (MPFSS) is an FSS scheme for the class of multi-point functions, where a point function $f_{S,\vec{y}}$ is represented in a natural way. We assume that an MPFSS scheme leaks not only the input and output domains but also the number of points t that the multi-point function specifies. An MPFSS can be easily obtained by adding t instances of a DPF.

2.3.3 Pseudorandom Correlation Generator (PCG)

Pseudorandom correlation generators (PCG) have been introduced in [BCG+19b]. Informally, a pseudorandom correlation generator allows to generate pairs of short keys (or seeds) $(\mathsf{k}_0, \mathsf{k}_1)$ such that each key k_σ can be expanded to a long string $R_\sigma = \mathsf{Expand}(\sigma, \mathsf{k}_\sigma)$, with the following guarantees: given the key $\mathsf{k}_{1-\sigma}$, the string R_σ is indistinguishable from a random string sampled conditioned on satisfying the target correlation with the string $R_{1-\sigma} = \mathsf{Expand}(1-\sigma, \mathsf{k}_{1-\sigma})$. We provide below the formal definition of pseudorandom correlation generators, after defining the notion of *reverse-sampleable correlation generator*.

Definition 5 (Correlation Generator). *A PPT algorithm $\mathcal{C}$ is called a* correlation generator*, if $\mathcal{C}$ on input 1^λ outputs a pair of elements in $\{0,1\}^n \times \{0,1\}^n$ for $n \in \mathsf{poly}(\lambda)$.*

In order to define security, we require the notion of a reverse-sampleable correlation generator introduced in the following.

Definition 6 (Reverse-sampleable Correlation Generator). *Let $\mathcal{C}$ be a correlation generator. We say $\mathcal{C}$ is* reverse sampleable *if there exists a PPT algorithm* RSample *such that for $\sigma \in \{0,1\}$ the correlation obtained via:*

$$\{(R'_0, R'_1) \,|(R_0, R_1) \xleftarrow{\$} \mathcal{C}(1^\lambda), R'_\sigma := R_\sigma, R'_{1-\sigma} \xleftarrow{\$} \mathsf{RSample}(\sigma, R_\sigma)\}$$

is computationally indistinguishable from $\mathcal{C}(1^\lambda)$.

Definition 7 (Pseudorandom Correlation Generator (PCG))**.** *Let $\mathcal{C}$ be a reverse-sampleable correlation generator. A* pseudorandom correlation generator (PCG) for $\mathcal{C}$ *is a pair of algorithms* $(\mathsf{PCG.Gen}, \mathsf{PCG.Expand})$ *with the following syntax:*

- $\mathsf{PCG.Gen}(1^\lambda)$ *is a PPT algorithm that given a security parameter λ, outputs a pair of seeds* $(\mathsf{k}_0, \mathsf{k}_1)$*;*
- $\mathsf{PCG.Expand}(\sigma, \mathsf{k}_\sigma)$ *is a polynomial-time algorithm that given party index $\sigma \in \{0,1\}$ and a seed k_σ, outputs a bit string $R_\sigma \in \{0,1\}^n$.*

The algorithms $(\mathsf{PCG.Gen}, \mathsf{PCG.Expand})$ *should satisfy the following:*

- **Correctness.** *The correlation obtained via:*

  $$\{(R_0, R_1) \,|(\mathsf{k}_0, \mathsf{k}_1) \xleftarrow{\$} \mathsf{PCG.Gen}(1^\lambda), R_\sigma \leftarrow \mathsf{PCG.Expand}(\sigma, \mathsf{k}_\sigma) \text{ for } \sigma \in \{0,1\}\}$$

  is computationally indistinguishable from $\mathcal{C}(1^\lambda)$.

- **Security.** *For any $\sigma \in \{0,1\}$, the following two distributions are computationally indistinguishable:*

  $$\{(\mathsf{k}_{1-\sigma}, R_\sigma) \mid (\mathsf{k}_0, \mathsf{k}_1) \xleftarrow{\$} \mathsf{PCG.Gen}(1^\lambda), R_\sigma \leftarrow \mathsf{PCG.Expand}(\sigma, \mathsf{k}_\sigma)\} \text{ and}$$
  $$\{(\mathsf{k}_{1-\sigma}, R_\sigma) \mid (\mathsf{k}_0, \mathsf{k}_1) \xleftarrow{\$} \mathsf{PCG.Gen}(1^\lambda), R_{1-\sigma} \leftarrow \mathsf{PCG.Expand}(\sigma, \mathsf{k}_{1-\sigma}), R_\sigma \xleftarrow{\$} \mathsf{RSample}(\sigma, R_{1-\sigma})\}$$

  where RSample *is the reverse sampling algorithm for correlation $\mathcal{C}$.*

Note that the above definition is trivial to achieve in general: We can let PCG.Gen on input 1^λ return $(R_0, R_1) \leftarrow \mathcal{C}(1^\lambda)$, and simply define Expand to be the identity. Typically, we will be interested in non-trivial constructions of PCGs, in which the seed size is significantly shorter than the output size. A pseudorandom generator with image in $\{0,1\}^n$ is a simple example for an expanding PCG for the equality correlation $\{(R, R) \mid R \in \{0,1\}^n\}$.

2.4 Computational Assumptions

2.4.1 Learning Parity with Noise (LPN)

Our constructions rely on the *Learning Parity with Noise* assumption [BFKL94] (LPN) over a field $\mathbb{F}$ (the most standard variant of LPN typically assumes $\mathbb{F} = \mathbb{F}_2$, but other fields can be considered). Unlike the LWE assumption, in LPN over $\mathbb{F}$ the noise is assumed to have a small Hamming weight. Concretely, the noise is a random field element in a small fraction of the coordinates and 0 elsewhere. Given a field $\mathbb{F}$, $\mathsf{Ber}_r(\mathbb{F})$ denote the distribution which outputs a uniformly random element of $\mathbb{F} \setminus \{0\}$ with probability r, and 0 with probability $1 - r$.

Definition 8 (LPN)**.** *For dimension $k = k(\lambda)$, number of samples (or block length) $q = q(\lambda)$, noise rate $r = r(\lambda)$, and field $\mathbb{F} = \mathbb{F}(\lambda)$, the $\mathbb{F}$-$\mathsf{LPN}(k, q, r)$ assumption states that*

$$\{(A, \vec{b}) \mid A \xleftarrow{\$} \mathbb{F}^{q \times k}, \vec{e} \xleftarrow{\$} \mathsf{Ber}_r(\mathbb{F})^q, \vec{s} \xleftarrow{\$} \mathbb{F}^k, \vec{b} \leftarrow A \cdot \vec{s} + \vec{e}\}$$
$$\overset{\text{c}}{\approx} \{(A, \vec{b}) \mid A \xleftarrow{\$} \mathbb{F}^{q \times k}, \vec{b} \xleftarrow{\$} \mathbb{F}^q\}$$

Here and in the following, all parameters are functions of the security parameter λ and computational indistinguishability is defined with respect to λ. Note that the search LPN problem, of finding the vector can be reduced to the decisional LPN assumption [BFKL94, AIK09]. In this paper, our protocols will mostly rely on a variant of LPN, called *exact LPN* (xLPN) [JKPT12]. In this variant, the noise vector $\vec{e}$ is not sampled from $\mathsf{Ber}_r(\mathbb{F})^q$, but it is sampled uniformly from the set $\mathsf{HW}_{rq}(\mathbb{F}^q)$ of length-q vectors over $\mathbb{F}$ with *exactly* rq nonzero coordinates (in contrast, a sample from $\mathsf{Ber}_r(\mathbb{F})^q$ has an *expected* number $r \cdot q$ of nonzero coordinates). While standard LPN is usually preferred since the Bernouilli distribution is convenient to analyze, xLPN is often preferred in concrete implementations, since it offers a potentially higher level of security for similar parameters (by avoiding weak instances with a low amount of noise). Furthermore, as outlined in [JKPT12], xLPN and LPN are equivalent: xLPN reduces to its search version using the sample-preserving reduction of [AIK07], and search-xLPN is easily seen to be polynomially equivalent to search-LPN.

Dual LPN. In our protocols, it will also prove convenient to work with the (equivalent) alternative *dual* formulation of LPN.

Definition 9 (Dual LPN)**.** *For dimension $k = k(\lambda)$, number of samples (or block length) $q = q(\lambda)$, noise rate $r = r(\lambda)$, and field $\mathbb{F} = \mathbb{F}(\lambda)$, the dual-$\mathbb{F}$-$\mathsf{LPN}(k, q, r)$ assumption states that*

$$\{(H, \vec{b}) \mid H \xleftarrow{\$} \mathbb{F}^{q-k\times q}, \vec{e} \xleftarrow{\$} \mathsf{Ber}_r(\mathbb{F})^q, \vec{b} \leftarrow H \cdot \vec{e}\}$$
$$\overset{c}{\approx}\{(H, \vec{b}) \mid H \xleftarrow{\$} \mathbb{F}^{q-k\times q}, \vec{b} \xleftarrow{\$} \mathbb{F}^q\}$$

Solving the dual LPN assumption is easily seen to be at least as hard as solving LPN: given a sample $(A, \vec{b})$, define $H \in \mathbb{F}^{q-k\times q}$ to be the parity-check matrix of A (hence $H \cdot A = 0$), and feed $(H, H \cdot \vec{b})$ to the dual LPN solver. Note that the parity check matrix of a random matrix is distributed as a random matrix. Furthermore, when $\vec{b} = A \cdot \vec{s} + \vec{e}$, we have $H \cdot \vec{b} = H \cdot (A \cdot \vec{s} + \vec{e}) = H \cdot \vec{e}$. For discussions regarding existing attacks on LPN and their efficiency, we refer the reader to [BCGI18, BCG+19b].

2.4.2 Quadratic Residuosity Assumption (QR)

We say that N is a Blum integer if $N = p \cdot q$ for some primes p and q such that $p \pmod 4 \equiv q \pmod 4 \equiv 3$. We denote by $\mathbb{J}_N$ the multiplicative group of the elements in $\mathbb{Z}_N^\star$ with Jacobi symbol $+1$ and by $\mathbb{QR}_N$ the multiplicative group of quadratic residues modulo N with generator g. Note that $\mathbb{QR}_N$ is a subgroup of $\mathbb{J}_N$, and that $\mathbb{QR}_N$ and $\mathbb{J}_N$ have order $\frac{\phi(N)}{4}$ and $\frac{\phi(N)}{2}$ respectively, where $\phi(\cdot)$ is Euler's totient function. It is useful to write $\mathbb{J}_N \colon \mathbb{H} \times \mathbb{QR}_N$, where $\mathbb{H}$ is the multiplicative group $(\pm 1, \cdot)$ of order 2. Note that is N is a Blum integer then $\gcd(2, \frac{\phi(N)}{4}) = 1$ and $-1 \in \mathbb{J}_N \setminus \mathbb{QR}_N$.

Definition 10 (Quadratic Residuosity Assumption, [GM82])**.** *Let N be a uniformly sampled Blum integer and let $\mathbb{QR}_N$ be the multiplicative group of quadratic residues modulo N with generator g. We say the* QR *assumption holds with respect to $\mathbb{QR}_N$ if for any p.p.t. adversary $\mathcal{A}$*

$$\left| \Pr_{a \xleftarrow{\$} \mathbb{QR}_N} [\mathcal{A}(N, g, a) = 1] - \Pr_{a \xleftarrow{\$} \mathbb{QR}_N} [\mathcal{A}(N, g, (-1) \cdot a) = 1] \right| \leq \mathsf{negl}(\lambda).$$

2.4.3 Decisional Diffie-Hellman (DDH)

Definition 11 (Decisional Diffie-Hellman, [Bon98])**.** *We say that the Decisional Diffie-Hellman assumption (*DDH*) holds if there exists a PPT group generator $\mathcal{IG}$ with the following properties. The output of $\mathcal{IG}(1^\lambda)$ is a pair $(\mathbb{G}, g)$ where $\mathbb{G}$ describes a cyclic group of a prime order q (where we use multiplicative notations for the group operation) and g describes a group generator. We assume that q is included in the group description $\mathbb{G}$. We also assume the existence of an efficient algorithm that given $\mathbb{G}$ and descriptions of group elements h_1, h_2 outputs a description of $h_1 h_2$. Finally, we require that for every nonuniform polynomial-time algorithm $\mathcal{A}$ there is a negligible function ϵ such that:*

$$|\Pr[\mathcal{A}(\mathbb{G}, g, g^a, g^b, g^{ab}) = 1 \colon (\mathbb{G}, g) \xleftarrow{\$} \mathcal{IG}; (a, b) \xleftarrow{\$} \mathbb{Z}_q^2] - \Pr[\mathcal{A}(\mathbb{G}, g, g^a, g^b, g^c) = 1 \colon (\mathbb{G}, g) \xleftarrow{\$} \mathcal{IG}; (a, b, c) \xleftarrow{\$} \mathbb{Z}_q^3]| \leq \epsilon(\lambda).$$

2.4.4 Decision Composite Residuosity (DCR)

Let SampleModulus be a polynomial-time algorithm which, on input the security parameter λ, outputs (N, p, q) where p and q are λ-bit primes and $N = p \cdot q$.

Definition 12 (Decision Composite Residuosity assumption, [Pai99])**.** *Let λ be a security parameter. We say that the Decision Composite Residuosity (*DCR*) problem is hard relative to* SampleModulus *if $(N, x) \approx_c (N, x^N)$ where $(N, p, q) \xleftarrow{\$} \mathsf{SampleModulus}(1^\lambda)$, $x \xleftarrow{\$} \mathbb{Z}_{N^2}^*$, and x^N is computed modulo N^2.*

Note that $\mathbb{Z}_{N^2}^*$ can be written as a product of subgroups $\mathbb{H} \times \mathbb{NR}_N$, where $\mathbb{H} = \{(1+N)^i : i \in [N]\}$ is of order N, and $\mathbb{NR}_N = \{x^N : x \in \mathbb{Z}_{N^2}^*\}$ is the subgroup of N-th residues that has order $\phi(N)$.

Circular-Secure Paillier Cryptosystem. We also recall in fig. 2.1 the circular-secure cryptosystem presented by Brakerski and Goldwasser in [BG10] which can be seen as a circular-secure version of Paillier's cryptosystem [Pai99]. The security of the scheme follows from the DCR assumption. The scheme is parameterised by $\ell \in \mathbb{N}$ that is polynomial in the security parameter λ.

PKE Circular-Secure Paillier Cryptosystem, [BG10]

$\mathsf{BG.KeyGen}(1^\lambda)$:

1. Sample $(N, p, q) \leftarrow \mathsf{SampleModulus}(1^\lambda)$.
2. Sample $\mathbf{g} = (g_0, \ldots, g_{\ell-1}) \xleftarrow{\$} \mathbb{NR}_N^\ell$.
3. Sample $\mathbf{d} = (d^{(0)}, \ldots, d^{(\ell-1)}) \xleftarrow{\$} \{0,1\}^\ell$.
4. Compute $\hat{g} = \prod_{i=0}^{\ell-1} g_i^{d^{(i)}} (\mod N^2)$.
5. Output $\mathsf{pk} = (N, \mathbf{g}, \hat{g})$ and $\mathsf{sk} = \mathbf{d}$.

$\mathsf{BG.Enc}(\mathsf{pk}, x)$:

1. Sample $r \xleftarrow{\$} \mathbb{Z}_N$.
2. Compute and output $\mathsf{ct} = (g_0^r, \ldots, g_{\ell-1}^r, \hat{g}^r \cdot (1+N)^x)$.

$\mathsf{BG.Dec}(\mathsf{sk}, \mathsf{ct})$

1. Parse $\mathsf{ct} = (c_0, \ldots, c_{\ell-1}, \hat{c})$.
2. Compute $\bar{c} = (\prod_{i=0}^{\ell-1} c_i^{-d^{(i)}}) \cdot \hat{c}(\mod N^2)$.
3. Compute and output $x = (\bar{c} - 1)/N$.

Figure 2.1: Brakerski and Goldwasser's Circular-Secure variant of the Paillier Cryptosystem.

Paillier-ElGamal Cryptosystem. The Paillier-ElGamal cryptosystem [CS02, DGS03, BCP03] is defined as in fig. 2.2, and boils down to using the ElGamal cryptosystem over the group $(\mathbb{Z}^\star_{N^2}, \times)$ where N is a Blum integer of the form $N = pq$, where p and q are primes.

PKE Paillier-ElGamal Cryptosystem, [CS02, DGS03, BCP03]

$\mathsf{PaillierEG.Gen}(1^\lambda)$:

1. Sample $g' \xleftarrow{\$} [N^2]$
2. Set $g \leftarrow (g')^{2N} \bmod N^2$
3. Sample $d \xleftarrow{\$} [N^2]$
4. Output $(\mathsf{pk} = g^d \bmod N^2, \mathsf{sk} = d)$

$\mathsf{PaillierEG.Enc}(\mathsf{pk}, x)$:

1. Sample $r \xleftarrow{\$} N$
2. Output $\mathsf{ct} = (g^r, \mathsf{pk}^r \cdot (1+N)^x)$

$\mathsf{PaillierEG.Dec}(\mathsf{sk}, \mathsf{ct} = (\mathsf{ct}_0, \mathsf{ct}_1))$:

1. Set $\mathsf{ct}' \leftarrow \mathsf{ct}_1 \cdot (\mathsf{ct}_0)^{-d} \bmod N^2$
2. Output $x = \frac{\mathsf{ct}'-1}{N}$

Figure 2.2: The Paillier-ElGamal Cryptosystem.

Assuming the DCR assumption (Definition 12), the Paillier-ElGamal cryptosystem is semantically secure. Observe that Paillier-ElGamal is a special case of the circular-secure Paillier cryptosystem of [BG10], where $\ell = 1$ (where ℓ is defined as in the previous paragraph). If we wish to encrypt (digits of) the secret key under Paillier-ElGamal however, we will need to assume the *circular security of the Paillier-ElGamal encryption scheme.*

2.4.4.1 Learning With Errors (LWE)

Definition 13 (Decisional Learning with Errors, [Reg05])**.** *Let $n \geq 1$ and $q \geq 2$ be integers. Let χ be an error distribution over $\mathbb{Z}$ and χ_{sk} be a secret key distribution over $\mathbb{Z}^n$. For $\vec{s} \xleftarrow{\$} \chi_{\mathsf{sk}}$, define $\mathsf{LWE}_{\chi,\vec{s}}$ to be the distribution obtained by sampling $\vec{a} \xleftarrow{\$} \mathbb{Z}_q^n$ uniformly at random, $\vec{e} \xleftarrow{\$} \chi$, and outputting $(\vec{a}, b =< \vec{a}, s > +e) \in \mathbb{Z}_q^{n+1}$. The decisional-$\mathsf{LWE}_{n,q,\chi,\chi_{\mathsf{sk}}}$ problem asks to distinguish polynomially many samples $(\vec{a}_i, b_i) \xleftarrow{\$} \mathsf{LWE}_{\chi,\vec{s}}$ from the same number of samples taken from the uniform distribution on $(\mathbb{Z}_q^n, \mathbb{Z}_p)$, where the secret $\vec{s}$ is sampled accordng to $\chi_s k$.*

Chapter 3

Prior (and Concurrent) Works on Sublinear-Communication Secure Computation

We say an N-party protocol for securely computing an $\mathbb{F}$-arithmetic circuit with s gates, n inputs, and m outputs is:

- *optimal-communication* if it uses an amount of communication of the form
$$\mathcal{O}(n+m) \cdot N \cdot \log|\mathbb{F}|$$
We note that we are using the word "optimal" a bit loosely, and that other works may use a different definition.
- *circuit-independent* if it uses an amount of communication of the form
$$(N + \lambda + \log|\mathbb{F}| + n + m)^{\mathcal{O}(1)}$$
- *linear-communication* if it uses an amount of communication of the form
$$(N + \lambda + \log|\mathbb{F}| + n + m)^{\mathcal{O}(1)} \cdot s$$
Note that the literature sometimes reserves this term for protocols with communication $\mathcal{O}(N \cdot \log|\mathbb{F}| \cdot s) + (\lambda + n + m)^{\mathcal{O}(1)}$
- *sublinear-communication* if it uses an amount of communication of the form
$$(N + \lambda + \log|\mathbb{F}| + n + m)^{\mathcal{O}(1)} + o(s) \cdot \log|\mathbb{F}|$$
Again the term is sometimes reserved for the more stringent $\mathcal{O}(N+n+m) \cdot o(s) \cdot \log|\mathbb{F}| + \lambda^{\mathcal{O}(1)}$. In either case, the term $o(s)$ cannot hide any $\lambda^{\mathcal{O}(1)}$ term.

Finally, the term "low-communication" is meant as an informal catch-all for all of the above.

In this chapter we survey prior results on sublinear-communication secure multiparty computation. We include some linear-communication protocols, insofar they can inform the state-of-the-art on sublinear-communication MPC. We restrict this survey to low-communication general-purpose protocols, loosely defined a protocols for computing a "large and expressive class of circuits", *e.g.* the class of all depth-d polynomial-size circuits for some $d = \omega(1)$. Special-purpose low-communication protocols, such as private information retrieval, are out of scope for this thesis.
We consider the three settings of information-theoretic security in the plain model, information-theoretic security in the correlated randomness model, and computational security (in the plain model).

3.1 Information-Theoretic MPC (in the plain model)

3.1.1 Linear Communication

In this paragraph, we will assume that N parties, who are fully connected by secure and authenticated point-to-point channels, wish to perform secure computation in the presence of a computationally unbounded adversary. It is known [BGW88, CCD88] that any functionality can be securely computed with perfect correctness and security, against a passive adversary if there is an honest majority of parties (threshold $t < N/2$), and against an active adversary if there is an honest two-thirds super-majority of parties ($t < N/3$). If the parties additionally have access to a secure broadcast channel, then statistical correctness can be achieved against active adversaries even if we only assume a simple honest majority [RB89]. A boolean function can be securely computed in the presence of a passive adversary, but without an honest majority ($t \geq N/2$), if and only if it can be cast as a big exclusive-OR of arbitrary functions of each party's input, and what's more such functions can even be computed with any number of corruptions ($t \leq N$) [CK89]. Therefore information-theoretic "general-purpose" computation requires an honest majority.

3.1.2 Sublinear Communication

Early works sought to understand how much communication was required for general information-theoretic MPC as well as for specific functionalities [Kus89, FY92, CK93, FKN94]. Beaver et al. [BFKR91] showed that any function $f\colon \{0,1\}^n \to \{0,1\}$ can be securely computed by $N = O(t \cdot n/\log n)$ computationally unbounded parties (which in fact will be using an exponential amount of computation) using $\mathsf{poly}(n+N)$ communication, while tolerating t corruptions. This can be seen as breaking the circuit-size barrier information-theoretically, but in a restricted setting and incurring an exponential cost in computation.

Lower-bounding the communication complexity is tied to long-standing open problems in complexity theory. Indeed, [IK04] showed that establishing non-trivial bounds on the communication that a constant number of parties needs to perform honest-majority MPC with information-theoretic security against semi-honest adversaries would have strong implications to information-theoretic constant-server *Private Information Retrieval* (PIR) and short *Locally Decodable Error-Correcting Codes* (LDC). The setting of MPC with a constant number of parties has also been studied in [DPP14] (specifically, they consider three-party computation).

Recently, the question of determining how much communication is required for information-theoretic MPC has regained interest, both with in the semi-honest setting with an honest majority and in the malicious setting with a two-thirds honest supermajority [DNOR16, DNPR16, DLN19, DLS21]. In particular, these works show that there is a circuit of size s with n inputs which requires communication $\Omega(n \cdot s)$, that any gate-by-gate protocol must communicate $\Omega(N)$ bits per multiplication gate, where N is the number of parties, and that general secure computation requires communication linear in the size of the inputs and in the outputs.

3.2 MPC in the Correlated Randomness Model

The question of achieving MPC with linear online-communication in the correlated randomness model was settled early on by Beaver [Bea92], who introduced the circuit-

randomisation technique (and "Beaver Multiplication Triples", both with passive and with active security. We mention there is a rich literature on achieving offline-online MPC with low (albeit constant) overhead (in the circuit-size), in particular by using somewhat homomorphic encryption, [IPS08, BDOZ11, DPSZ12, DZ13].

3.2.1 Sublinear Online Communication

All the protocols mentioned in this paragraph make no honest majority assumption and tolerate up to all-but-one corruptions. [IKM+13] showed that if parties have access to an exponential amount of correlated randomness then they only require communication linear in the inputs, which is optimal, in order to securely realise any multiparty functionality; this is achieved with perfect security against semi-honest adversaries, or with statistical security against malicious ones. In the special case of a two-party functionality where only one of the parties receives an output, [IKM+13] achieves perfect security even in the malicious setting. [BIKK14] later improved the correlated randomness complexity of the two-party setting with active security to the subexponential $2^{\tilde{O}(\sqrt{\log n})}$ (where $\log n$ is the bit-length of the inputs). Couteau [Cou19] constructed an unconditionally secure TwoPC protocol for the computation of layered circuits, which uses only a polynomial amount of correlated randomness and with communication sublinear in the circuit-size; security in the semi-honest setting is guaranteed for both arithmetic and boolean circuits, but only for boolean circuits in the malicious setting.

3.2.2 Lower Bounds?

[IKM+13] shows that achieving low communication with a small amount of correlated randomness would lead to a major breakthrough in private information retrieval. More specifically [IKM+13, Theorem 14] establishes that the existence of protocol for computing *every* (sender-receiver) functionality $\{0,1\}^n \times \{0,1\}^n \to \{\perp\} \times \{0,1\}$ with communication $c(n)$ and an amount of correlated randomness $r(n)$ would imply the existence of 3-server statistical PIR with communication complexity $\mathcal{O}(r(\log N) + c(\log N) + \log(N))$, where N is the size of the database. This can be seen as an implausibility result (or at least a "meta hardness" result) for achieving secure computation with circuit-independent communication in the correlated randomness model using only a polynomial amount of correlated randomness.

3.2.3 (Sub)linear-Communication Secure Computation in the Correlated Randomness Model, via Circuit Randomisation

In previous sections, we surveyed protocols in the correlated randomness model with at most linear communication complexity (in the circuit size). We now focus on a single paradigm, namely *circuit randomisation*, and provide a more in depth description of protocols following this approach. While other protocols in the literature may be cast to follow this paradigm, we restrict this presentation to the works of [Bea92, BCG+19b, IKM+13, DNNR17, Cou19] as these are central to understanding the protocol of [Cou19], and in turn our own works.

The "gate-by-gate" design is a popular paradigm for securely computing an $\mathbb{F}$-

arithmetic[1] circuit amongst N parties in the presence of a semi-honest adversary[2] corrupting at most t parties is to have the parties maintain (t, N)-threshold linear secret shares of the values at each wire. Initially, parties ensure they hold shares of the input wires (if an input is held only by a single party for instance, they can sample everyone's secret shares locally then deal them out using $(N-1) \cdot \log |\mathbb{F}|$ bit of communication), then proceed through the circuit gate by gate in topological order (linear operations can be done locally, but multiplications of two secret-shared values require interaction), and finally reconstruct the outputs by broadcast their shares. The communication's dependency on the circuit-size is due to the multiplications, which can be performed information-theoretically if there is an honest majority, and using oblivious transfer otherwise. Beaver [Bea92] showed that these expensive multiplications can be pre-processed in an offline phase using *circuit randomisation*, in such a way that each one only requires two reconstructions (and therefore at most $(t + (N-1)) \cdot \log |\mathbb{F}|$ bits of communication). The way Beaver describes this trick is as follows: "set every input to every gate in the circuit completely at random, and then make corrections". Boyle, Gilboa, and Ishai [BGI19] proposed a framework for achieving secure computation with input-independent preprocessing, based on *function secret-sharing*. This framework can be seen as a generalisation of Beaver's paradigm:

Paradigm (Circuit Randomisation, Informal, adapted from [BGI19])**.** *Two or more semi-honest parties wishing to securely evaluate some public function f on some linearly secret-shared input $x \in \mathbb{F}^n$ can follow this protocol template:*

- *Initially, each party is assumed to hold a linear share of some random mask $r \in \mathbb{F}^n$, as well as a "(one-time) share of the function $g_r(\cdot) \coloneqq f(\cdot - r)$". A (one-time) share of a function h is syntactically defined as a (deterministic) function whose value, on input X, is a share of X. The meaning of this will become clearer given the examples in the rest of this section.*
- *Each party locally computes a linear share of $c_x \coloneqq x + r$, then broadcasts this share. Then, everyone can reconstruct c_x.*
- *Each party evaluates their function share of g_r on the public value c_x, thereby generating a share of $g_r(c_x) = f(x)$.*

Depending on the setting, it may be convenient to assume the parties hold shares of some joint input (as above), or to assume each party holds their own input. We now present the modified version of circuit-randomisation, in the latter case. Suppose N parties wish to compute an N-input function f.

- In an offline phase, each party is P_i is initially given a mask r_i (of the same length as their eventual input). Each party is additionally given a (one-time) share of the function $g_{r_1,\ldots,r_N} \colon (X_1, \ldots, X_N) \mapsto f(X_1 - r_1, \ldots, X_N - r_N)$.
- At the beginning of the online phase, each party P_i, holding input x_i broadcasts $c_i \coloneqq x_i + r_i$.
- Each party evaluates their function share of $g_{r_1,\ldots,r_N}$ on the public $(c_1, \ldots, c_N)$, thereby generating a share of $g_{r_1,\ldots,r_N}(c_1, \ldots, c_N) = f(x_1, \ldots, x_N)$.

We start by recalling Beaver's circuit-randomisation technique for multiplication [Bea92] (in other words, and with the above notations, f is simply a multiplication).

[1] This includes the boolean case, i.e. $\mathbb{F} = \mathbb{F}_2$.

[2] It is well known that Beaver's approach can be upgraded to active security if the multiplication triples are authenticated. All protocols discussed in this section can similarly be upgraded to active security, however for simplicity we focus on passive security.

Beaver's circuit-randomisation technique for multiplication [Bea92]. Suppose that each party P_i holds linear shares $[x]_i$ and $[y]_i$ of x and y respectively, and further that they hold pre-computed secret-shares $[a]_i$, $[b]_i$, and $[c]_i$ which are assumed to have been generated in an offline phase with $a, b \xleftarrow{\$} \mathbb{F}$ and $c = a \cdot b$.
If each party P_i computes a share of $\alpha \coloneqq x + a$ as $[\alpha]_i \leftarrow [x]_i + [a]_i$ and a share of $\beta \coloneqq y + b$ as $[\beta]_i \leftarrow [y]_i + [b]_i$, they can broadcast these shares then reconstruct α and β locally from the other parties' shares. Each party P_i ($i \in [N]$) can set their share of the multiplication as $[x \cdot y]_i \leftarrow [c]_i + \alpha \cdot \beta + \alpha \cdot [b]_i + [a]_i \cdot \beta$. Correctness of reconstruction follows from eq. (3.1).

$$\begin{aligned} x \cdot y &= ((x + a) - a) \cdot ((y + b) - b) \\ &= (\alpha + a) \cdot (\beta + b) \\ &= ab + \alpha\beta + \alpha b + a\beta \\ &= c + \alpha\beta + \alpha b + a\beta \end{aligned} \tag{3.1}$$

With the more modern formulation we proposed above the secret-shared input is (x, y), and the mask is $r = (a, b)$. Each party's share of the function $g_r \colon (X, Y) \mapsto (X - a) \cdot (Y - b)$ is the function $(A, B) \mapsto [c]_i + A \cdot B + A \cdot [b]_i + [a]_i \cdot B$. This last function is "the i^{th} party's one-time share of the function g_r".

Boyle et al.'s constant-depth circuit randomisation [BCG+19b]. Let's now suppose that the parties hold shares of $x_1, \ldots, x_n$ and wish to compute shares of some constant-depth (fan-in two) circuit in the x_i . For simplicity, let us start with the case of a single-output fan-in two circuit, i.e. a polynomial P, whose (constant) degree we denote d. Suppose the parties hold shares of $r_1, \ldots, r_n$ (which are assumed to have been sampled i.i.d. uniformly at random) as well as shares of $\prod_{i \in [n]} r_i^{d_i}$ for every $(d_1, \ldots, d_n) \in [0, n]^n$ s.t. $d_1 + \cdots + d_n \leq d$ (note that by a balls-and-bins argument there are $\binom{n+d-1}{d-1}$ such tuples, which is polynomial). Similarly to before, the parties can locally compute shares of $\alpha_i \coloneqq x_i + r_i$ for $i \in [n]$, then broadcast them so all parties can reconstruct $\alpha_1, \ldots, \alpha_n$.

$$\begin{aligned} P(x_1, \ldots, x_n) &= \sum_{\substack{0 \leq d_1, \ldots, d_n \leq n \\ d_1 + \cdots + d_n \leq d}} c_{d_1, \ldots, d_n} \prod_{i \in [n]} x_i^{d_i} \\ &= \sum_{\substack{0 \leq d_1, \ldots, d_n \leq n \\ d_1 + \cdots + d_n \leq d}} c_{d_1, \ldots, d_n} \prod_{i \in [n]} ((x_i + r_i) - r_i)^{d_i} \\ &= \sum_{\substack{0 \leq d_1, \ldots, d_n \leq n \\ d_1 + \cdots + d_n \leq d}} c_{d_1, \ldots, d_n} \prod_{i \in [n]} \sum_{k=0}^{d_i} ((-1)^k \tbinom{d_i}{k} \alpha_i^{d_i - k} r_i^k) \\ &= \sum_{\substack{0 \leq d_1, \ldots, d_n \leq n \\ d_1 + \cdots + d_n \leq d}} c'_{d_1, \ldots, d_n} \prod_{i \in [n]} \alpha_i^{d_i} \\ \text{where } \quad c'_{d_1, \ldots, d_n} &= \sum_{\substack{0 \leq d'_1, \ldots, d'_n \leq n \\ (d_1 + d'_1) + \cdots + (d_n + d'_n) \leq d}} \left(c_{(d_1 + d'_1), \ldots, (d_n + d'_n)} \prod_{i \in [n]} \left((-1)^{d'_i} \tbinom{d_i}{d'_i} r_i^{d'_i} \right) \right) \\ &= \sum_{\substack{0 \leq d'_1, \ldots, d'_n \leq n \\ (d_1 + d'_1) + \cdots + (d_n + d'_n) \leq d}} \left(\left(c_{(d_1 + d'_1), \ldots, (d_n + d'_n)} \prod_{i \in [n]} (-1)^{d'_i} \tbinom{d_i}{d'_i} \right) \left(\prod_{i \in [n]} r_i^{d'_i} \right) \right) \end{aligned} \tag{3.2}$$

There are two dual interpretations of eq. (3.2):

- *A secret-shared function evaluated on a public input:* the "FSS viewpoint" is that $P(x_1, \ldots, x_n)$ can be expressed as a (secret) degree-d polynomial function in the (public) $(\alpha_i)_{i \in [n]}$, whose coefficients are degree-d polynomials in the $(r_i)_{i \in [n]}$;

- *A public function evaluated on a secret-shared input:* the "HSS viewpoint" is that $P(x_1, \ldots, x_n)$ can be expressed as a (public, given the $(\alpha_i)_{i\in[n]}$) degree-d polynomial function in the (secret) $(r_i)_{i\in[n]}$.

In either case, given the α_i $(= x_i + r_i)$ for $i \in [n]$, as well as additive shares of the $\prod_{i\in[n]} r_i^{d_i}$ for every $(d_1, \ldots, d_n) \in [n]^n$ s.t. $d_1 + \cdots + d_n \leq d$, each party P_j can locally generate an additive share of $P(x_1, \ldots, x_n)$ as follows:

$$[P(x_1, \ldots, x_n)]_j \leftarrow \sum_{\substack{0\leq d_1, d'_1, \ldots, d_n, d'_n \leq n \\ d_1+\cdots+d_n\leq d \\ (d_1+d'_1)+\cdots+(d_n+d'_n)\leq d}} \left(\prod_{i\in[n]} \alpha_i^{d_i}\right) \left(c_{(d_1+d'_1),\ldots,(d_n+d'_n)} \prod_{i\in[n]} (-1)^{d'_i} \binom{d_i}{d'_i}\right) \left[\prod_{i\in[n]} r_i^{d'_i}\right]_j .$$

More compactly, this can be rewritten as:

$$[P(x_1, \ldots, x_n)]_j \leftarrow [(1_{\mathbb{F}} \| \vec{r})^{\otimes d}]_j \cdot \vec{c}_{\alpha_1,\ldots,\alpha_n,P}$$
$$\text{where } \vec{c}_{\alpha_1,\ldots,\alpha_n,P} \in \mathbb{F}^{(n+1)^d} \text{ is the vector of coefficients of}$$
$$P(x_1, \ldots, x_n), \text{ seen as a polynomial function in } (r_1, \ldots, r_n) .$$

Finally, observe that holding the α_i as well as shares of the $\prod_{i\in[n]} r_i^{d_i}$ is enough to compute shares of $[P(x_1, \ldots, x_n)]$ for *every* degree-d polynomial P. In particular, this allows the parties to compute shares of every output wire of a constant-depth fan-in two circuit.

Ishai et al.'s "One-time truth tables" [IKM+13]. Ishai, Kushilevitz, Meldgaard, Orlandi, and Paskin-Cherniavsky introduced the "one-time truth table" protocol, perfectly realising (with active security) any N-party functionality with optimal communication, provided the parties are given access to an exponential amount of (function-dependent) correlated randomness. One way to view their protocol is simply as circuit randomisation, where the "function shares" of g_r $(= f(\cdot - r))$ are simply linear secret shares of the truth table of g_r. A convenient way to view this is that the i^{th} party is receiving the truth table of the function $g_r^{(i)}$, where the $(g_r^{(j)})_{j\in[N]}$ are chosen uniformly at random conditioned on $\sum_{j\in[N]} g_r^{(j)} = g_r$. In other words, a share of the truth table of a function is the truth table of a share of the function.

Couteau's loglog-depth circuit randomisation [Cou19]. The key idea of behind Couteau's protocol, already implictly present in Damgård, Nielsen, Nielsen, Ranellucci's "TinyTables" protocol [DNNR17], is that a low-depth circuit can be decomposed into chunks, such that each one has a polynomial-size truth table. More precisely, if a function can be computed by a depth-d circuit, where each gate has fan-in at most 2, then the function must be 2^d-local, meaning each of the m output bit/field element of f can be computed by only considering a size-2^d subset of the inputs. If $d \leq \log\log n$, then each output of f has a polynomial-size truth table, which provides an avenue to secret-share the function f with a polynomial amount of correlated randomness:

- Each party is given a linear share of r, as well as a linear share of $g_r^{(i)} \coloneqq f_i(\cdot - r[S_i])$ (which could *e.g.* be locally derived from $(1_{\mathbb{F}} \| r[S_i])^{\otimes 2^d}$)for $i \in [m]$.
- Each party locally computes a linear share of $c_x \coloneqq x + r$, then broadcasts this share. Then, everyone can reconstruct c_x.
- Each party evaluates their function share of $g_r^{(i)}$ on the public value $c_x[S_i]$ (for $i \in [m]$), then outputs the concatenation of the resulting shares.

3.3 Computational MPC

3.3.1 Linear Communication

In this setting, linear-communication protocols would be both too long to survey and out of scope. We simply note that the communication complexity of the seminal protocols is already only linear in the circuit-size. In the cryptographic setting, still under the assumption that the parties have access to a full set of pairwise connected channels, it is known that any function can be securely computed while tolerating all-but-one corruptions ($t < N$) from *Oblivious Transfer* (OT) [Yao82, GMW87a, Kil91] (note that Yao's [Yao82] garbled circuits protocol additionally requires the existence of one-way functions, which is implied by the existence of an OT *protocol*, while the other protocols hold in the OT-hybrid model[3]). However, the communication complexity of all these protocols (even in the OT-hybrid model) is linear in the circuit-size of the function being computed.

3.3.2 Sublinear Communication

We emphasise that this section does not include the results presented in section 1.3 of this thesis.

With the construction of *Fully Homomorphic Encryption* (FHE) (a problem introduced by [RAD78]) from standard assumptions [Gen09], protocols emerged for secure computation with constant communication and polynomial computation under variants of the *Learning With Errors* (LWE) computational assumption [Gen09, DFH12, AJL+12]. Recent progress in breaking the circuit-size barrier for layered circuits in the computational setting from assumptions not known to imply FHE follows from the advances in HSS (and PCGs, extended upon in the next paragraph) for super-constant depth circuits from a variety of assumptions: [BGI16a] for DDH and [FGJS17] from (a circular-secure variant of) DCR.
Finally, *Stacked Garbled Circuits*—which were first built in the *Random Oracle Model* (ROM) [HK20], then in the standard model [HK21]—allow for two-party computation with communication proportional to the longest execution path of the circuit. This leads to sublinear 2PC for the large class of circuits with high conditional branching. These results on secure computation with "free conditional branching" were extended to the multiparty setting [GHAHJ22]. It seems unclear whether this line of work on circuits with high conditional branching should be considered "general computation" or "specialised computation". Finally, we note that similar results exist in the correlated randomness model, as [HKP20] showed that Beaver triples [Bea92] can be reused across conditional branches.

[3]This distinction is relevant if we want to *e.g.* achieve oblivious transfer using noisy channels.

Chapter 4

Offline-Online Sublinear-Communication Two-Party Computation

This chapter describes results which have been communicated previously in [CM21].

Based on joint work with Geoffroy Couteau.

In this chapter, we show that circuit-dependent homomorphic secret sharing—i.e. HSS where the share generation requires knowing in advance the circuit to be evaluated homomorphically—for the class of log-local circuits exists, conditioned on (the quasi-polynomial hardness of) a well-studied 20th century assumption: the learning parity with noise (LPN) assumption [BFKL94]. Informally, the LPN assumption captures the hardness of solving an overdetermined system of linear equations over $\mathbb{F}_2$, when a small subset of the equations is perturbed with a random noise. The LPN assumption has a long history in computational learning theory, where it emerged. Furthermore, our results only require a flavour of LPN where the adversary is given a very limited number of samples (typically, $O(n)$ equations in n indeterminates). In this regime, LPN is equivalent to the hardness of decoding random linear codes over $\mathbb{F}_2$, which is the well-known *syndrome decoding* problem in the coding theory community, where it has been studied since the 60's [Pra62].

Details on the underlying assumption. In a bit more detail, given a security parameter λ, the (T, n, N, r)-LPN assumption with dimension $n = n(\lambda)$, number of samples $N = N(\lambda)$ and noise rate $r = r(\lambda)$ states that for every adversary Adv running in time at most $T = T(\lambda)$,

$$\Pr\left[A \xleftarrow{\$} \mathbb{F}_2^{N\times n}, \vec{e} \xleftarrow{\$} \mathsf{Ber}_r^N, \vec{s} \xleftarrow{\$} \mathbb{F}_2^n \ : \ \mathsf{Adv}(A, A\cdot\vec{s}+\vec{e}) = \vec{s}\right] = \mathsf{negl}(\lambda),$$

where Ber_r denotes the Bernouilli distribution which outputs 1 with probability r, and negl denote some negligible function. When T can be any polynomial (resp. any super-polynomial function, some super-polynomial function), we say that we assume the polynomial (resp. quasi-polynomial, super-polynomial) hardness of LPN. For arithmetic circuits, we need to assume LPN over large fields, or equivalently syndrome decoding for random linear codes over large fields; this is also a well-founded and well-studied assumption, used in several previous works, e.g. [BCGI18, BCG+19b].

HSS for Any loglog-Depth Circuit. We introduce a new circuit-dependent HSS scheme for the class of any $\log\log$-depth circuits. We emphasise that unlike traditional forms of HSS, here the input-sharing phase depends on the homomorphic evaluation circuit: in other words it is an HSS scheme for any singleton class comprised of a single circuit of depth $\log\log$, not for the class of all such circuits simultaneously.

Main Theorem 1 (HSS for any loglog-Depth Circuit, Informal)**.** *Let C be a size-s, n-input, m-output, $(\epsilon \cdot \log\log)$-depth arithmetic circuit over $\mathbb{F}$ (for some $\epsilon \leq 1/4$). If the $\mathbb{F}$-LPN assumption with super-polynomial dimension ℓ, $O(\ell)$ samples, and inverse super-polynomial rate holds then there exists a secure HSS scheme for the class $\{C\}$ with share size $n + O(m \cdot s \cdot \log s / c^{\log^{1-\epsilon} s - \log^{1-2\epsilon} s})$ (for some constant c) and computational complexity $O(m \cdot \mathsf{poly}(s) \cdot (\log|\mathbb{F}|)^2)$.*

Restricting the circuit class to depth-k size-s circuits where $k(s) \leq \log\log s/4$ leads to quantitative improvements in the size of the shares, the computational complexity of expanding shares, and the strength of the LPN assumption.

Application to Sublinear Computation. Our HSS scheme has (non black-box) implications for sublinear computation. As in [BGI16a], our results holds for all layered (boolean or arithmetic) circuits, in the two-party setting.

Main Theorem 2 (Sublinear Computation of Layered Circuits, Informal)**.** *For any layered arithmetic circuit C of polynomial size $s = s(\lambda)$ with n inputs and m outputs, for any function $k(s) \leq \log\log s - \log\log\log s + O(1)$, there exists a two party protocol for securely computing C in the honest-but-curious model, with total communication $2(n + m + s/k) \cdot \log|\mathbb{F}| + o(s/k)$ and computation bounded by $s^3 \cdot \mathsf{polylog}(s) \cdot (\log|\mathbb{F}|)^2$ under a set of LPN assumptions, the exact nature of which depends on the sublinearity factor k.*
In particular, setting $k \leftarrow O(\log\log s)$ leads to a protocol with total communication $O(n + m + s/\log\log s)$, secure under the super-polynomial hardness of:

- *$\mathbb{F}$-LPN with super-polynomial dimension ℓ, $O(\ell)$ samples, and inverse super-polynomial rate,*
- *$\mathbb{F}_2$-LPN with super-polynomial dimension ℓ', $O(\ell')$ samples, and inverse polynomial rate $1/s^{O(1)}$ (which is implied by the above if $\mathbb{F} = \mathbb{F}_2$).*

Furthermore (but with a slighly different choice of parameters than the one described above), as k is reduced to an arbitrarily small $k = \omega(1)$, we need only assume the quasi-polynomial hardness of:

- *$\mathbb{F}$-LPN with quasi-polynomial dimension ℓ, $O(\ell)$ samples, and inverse quasi-polynomial rate,*
- *$\mathbb{F}_2$-LPN with quasi-polynomial dimension ℓ', $O(\ell')$ samples, and inverse polynomial rate $1/s^{O(1)}$ (which is implied by the above if $\mathbb{F} = \mathbb{F}_2$).*

and the computation is reduced to $O(s^{1+o(1)} \cdot (\log|\mathbb{F}|)^2)$.

Remark 2. *While we require security against super-polynomial-time adversaries, this remains a relatively weak flavour of LPN where the dimension is very high, i.e. super-polynomial as well (and the adversary is allowed to run in time $O(\ell^2)$ where ℓ is the dimension), and the number of samples which the adversary gets is very limited, $O(\ell)$.*

On the other hand, we require a very small noise rate λ/N. For example, instantiating the above with $k = (\log\log s)/5$, we obtain a secure computation protocol with total communication $O(\ell+m+s/\log\log s)$ (sublinear in s) and polynomial computation, assuming that LPN *is hard against adversaries running in super-polynomial time $\lambda^{O(\log\lambda)}$, with dimension $\ell = \lambda^{O(\log\lambda)}$, $N = 2\ell$ samples, and noise rate λ/N. More generally, for any super-constant function $\omega(1)$, there is a two-party protocol with communication $O(n+m+s/\log\omega(1))$ assuming the $\lambda^{\omega(1)}$-hardness of* LPN *(i.e., the quasi-polynomial hardness of* LPN*).*

We note that, in this regime of parameters, the best known attacks are the information set decoding attack [Pra62] and its variants (which only shave constant in the exponents, hence have the same asymptotic complexity), which require time $2^{O(\lambda)}$.[1] Therefore, assuming hardness against $\lambda^{O(\log\lambda)}$-time adversaries is a very plausible assumption.

Remark 3 (On the Generality of Layered Circuits)**.** *Our construction is restricted to the class of (boolean or arithmetic) layered circuits. This restriction stems from the blockwise structure of the construction, and was also present in the previous works of [BGI16a] and [Cou19]. As noted in [Cou19], layered circuits are a relatively large and general class of circuits, which furthermore capture many "real-world" circuits such as FFT-like circuits (used in signal processing, integer multiplication, or permutation networks [?]), Symmetric crypto primitives (e.g. AES and algorithms that proceed in sequences of low-complexity rounds are naturally "layered by blocks"), or dynamic-programming algorithm (e.g. the Smith-Waterman distance, or the Levenshtein distance and its variants).*

Generalisation to the malicious setting. Our result can directly be generalised to the malicious setting using a generic GMW-style compiler [GMW87a], which is communication preserving when instantiated with succinct zero-knowledge arguments [NN01]. Such arguments exist under collision-resistant hash functions; hence, Theorem 2 extends to the malicious setting as well, at the cost of further assuming collision-resistant hash functions (which is a mild assumption). We note that CRHFs have recently been built from (sub-exponentially strong) flavours of LPN [AHI+17, YZW+19, BLVW19].

4.1 An Overview of Our Protocol

From now on, we set the number of parties to $N = 2$. The work of [BCG+19b, Section 6] provides a pseudorandom correlation generator under the LPN assumption, which generates correlated (pseudo) random strings for the low-degree polynomial correlation, i.e. shares of $(\vec{r}, \vec{r}^{\otimes 2}, \ldots, \vec{r}^{\otimes d})$ for some constant d, where $\vec{r}$ is a (pseudo)random vector. With the duality between PCGs and HSS this yields an HSS for constant-depth circuits. Our goal is to design a PCG which would lead to an HSS for super-constant depth circuits. More specifically, and keeping our end application in mind, we would like for our PCG to have short enough seeds to lead to a *compact* HSS scheme (i.e., shares of an input x should be at most $O(x)$). This is fundamental when using the scheme to generate correlated randomness in the protocol of [Cou19], which achieves sublinear communication in the correlated randomness model, and which is the starting point of our application to sublinear secure computation.

[1] BKW and its variants [BKW00, Lyu05] do not improve over information set decoding attacks in this regime of parameters, due to the very low number of samples.

Our approach is therefore to directly plug in the construction of [BCG+19b] and see where it fails. Two issues emerge: the computation is super-polynomial, and the communication not sublinear. Below, we outline each of these issues, and explain how we overcome them.

First Issue: Too Many Polynomials. The first problem which appears when plugging the PCG of [BCG+19b] in the protocol of [Cou19] is that the latter requires distributing *many* shares of multivariate polynomials $\hat{Q}$ – more precisely, s/k such polynomials (one for each coordinate of each first layer of a bloc). While the PCG of [BCG+19b] allows to compress pseudorandom pairs $(\vec{r}, Q(\vec{X} - \vec{r}))$ into short seeds, these seeds will still be of length at least $\omega(\log \lambda)$, where λ is the security parameter, for the PCG to have any hope of being secure. That means that even if we could manage to securely distribute all these seeds with optimal communication protocols, the overall communication would still be at the very least $\omega((s \log \lambda)/\log\log s)$, which cannot be sublinear since $\log\log s = o(\log \lambda)$ (as s is polynomial in λ).
We solve this first issue as follows: we fix a parameter β, and partition each $\vec{y}_i$ into w/β subvectors, each containing β consecutive coordinates of $\vec{y}_i$. Then, the core observation is that a simple variant of the PCG of [BCG+19b] allows in fact to generate shares of $(\vec{r}, \vec{r}^{\otimes 2}, \cdots, \vec{r}^{\otimes 2^k})$ for some pseudorandom r, where $\vec{r}^{\otimes j}$ denotes the tensor product of $\vec{r}$ with itself j times (which we call from now on the *j-th tensor power of $\vec{r}$*): this correlation is enough to generate shares of all degree-2^k polynomial in $\vec{r}$ rather than a single one. We will build upon this observation to show how to generate a batch of β shares of multivariate polynomials from a single tensor-power correlation, thus reducing the number of PCG seeds required in the protocol by a factor of β, at the tolerable cost of slightly increasing the size of each seed.

Solution: Batching β Multivariate Polynomials. Consider the first length-β subvector of $\vec{y}_{i+1}$, which we denote $\vec{v}$. Observe that the entire subvector $\vec{v}$ can depend on at most $\beta \cdot 2^k$ coordinates of $\vec{y}_i$, since each coordinate of $\vec{v}$ depends on at most 2^k coordinates of $\vec{y}_i$. Therefore, we can now see the computation of $\vec{v}$ from $\vec{y}_i$ as evaluating β multivariate polynomials $(Q_1 \cdots, Q_\beta)$, where all multivariate polynomials take as input the same size-$(\beta 2^k)$ subset of coordinates of $\vec{y}_i$. To securely compute shares of $\vec{v}$ from shares of $\vec{y}_i$, the parties can use the following type of correlated randomness: they will have shares of $(\vec{r}, \vec{r}^{\otimes 2}, \cdots \vec{r}^{\otimes 2^k})$, where $\vec{r}$ is a random mask of length $\beta \cdot 2^k$. Consider the following polynomials:

$$(\hat{Q}_1(\vec{X}), \cdots, \hat{Q}_\beta(\vec{X})) \stackrel{\text{def}}{=} (Q_1(\vec{X} - \vec{r}), \cdots, Q_\beta(\vec{X} - \vec{r})).$$

Each coefficient of each $\hat{Q}$ can be computed as a degree-2^k multivariate poynomial in the coordinates of $\vec{r}$ – or, equivalently, as a linear combination of the coordinates of $(\vec{r}, \vec{r}^{\otimes 2}, \cdots \vec{r}^{\otimes 2^k})$. Hence, given additive shares of $(\vec{r}, \vec{r}^{\otimes 2}, \cdots \vec{r}^{\otimes 2^k})$, the parties can locally compute additive shares of the coefficients of *all* the polynomials $(\hat{Q}_1, \cdots \hat{Q}_\beta)$. Using the PCG of [BCG+19b], the seeds for generating pseudorandom correlations of the form $(\vec{r}, \vec{r}^{\otimes 2}, \cdots \vec{r}^{\otimes 2^k})$ have length:

$$O\left(\lambda^{2^k} \cdot \log\left(\left(\beta \cdot 2^k\right)^{2^k}\right)\right),$$

where λ is some security parameter related to the hardness of the underlying LPN assumption. Or more simply, using the fact the computational cost of generating the correlations contains the term $\left(\beta \cdot 2^k\right)^{2^k}$ which must remain polynomial in s. Therefore, the total number of bits which the parties have to distribute (for all $(d/k) \cdot (w/\beta) = s/(\beta k)$ such seeds) is $O((s/k) \cdot (\lambda^{2^k} \cdot \log s)/\beta)$.

Choosing the Parameter β. Suppose for simplicity that we already have at hand an MPC protocol allowing to securely distribute such seeds between the parties, with linear overhead over the total length of the seeds generated. This means that generating the full material will require a total communication of $c \cdot s \cdot \lambda^{2^k} \cdot \log s/(\beta k)$. By setting β to be larger than $c \cdot \lambda^{2^k} \cdot \log s$, the total communication will be upper bounded by $O(s/k) = O(s/\log\log s)$ when setting $k \leftarrow O(\log\log s)$, which is the highest our techniques will allow it to be pushed. The most important remaining question is whether we can execute this process in polynomial time given such a large β. Put more simply, the core issue is that the *computational complexity* of expanding short seeds to shares of $(\vec{r}, \vec{r}^{\otimes 2}, \cdots \vec{r}^{\otimes 2^k})$ with the PCG of [BCG+19b] contains a term of the form $(\beta \cdot 2^k)^{2^k}$. To make the computation polynomial, we must therefore ensure that β is at most $s^{O(2^{-k})}$, which is subpolynomial. Fortunately, this can be done by setting the security parameter λ of the underlying PCG to be $s^{O(2^{-2k})}$. For instance, for any constant $\epsilon \in]0,1[$, we can set $\lambda \leftarrow 2^{\log^\epsilon s}$, $k \leftarrow \log\log s/c_\epsilon$, and $\beta \leftarrow s^{O(2^{-k})}$ for some explicit constant $c_\epsilon > 2$, at the cost of now having to assume the *quasi-polynomial security* of the LPN assumption.

4.1.1 Block Decomposition of Layered Circuits

Given an arithmetic circuit C and an input vector $\vec{x}$, we call *value of the gate g on input $\vec{x}$* the value carried by the output wire of a given gate g of C during the evaluation of $C(\vec{x})$. The following decomposition of layered circuits is implicit in [Cou19]; for completeness, we give the proof here. The decomposition of a layered circuits into chunks computable by low-degree functions is illustrated on Figure 4.1.

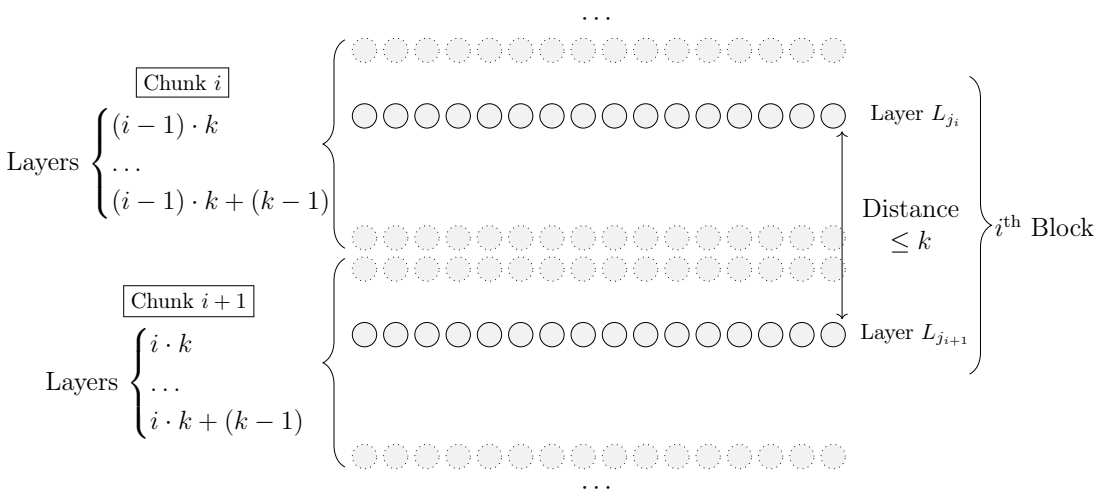


Figure 4.1: Block Decomposition of a Circuit.

Lemma 1 (Block-Decomposition of Layered Circuits, Implicit in [Cou19])**.** *Let C be a layered arithmetic circuit over a field $\mathbb{F}$ with n inputs and m outputs, of size s and depth $d = d(n)$. For any integer k, denoting $t = t(k) = \lceil d/k \rceil$, there exists $2t+1$ integers $(s_0 = 0, s_1, \cdots, s_{t-1}, s_t = 0)$, $(m_0, \cdots, m_{t-1})$, and functions $(f_0, \cdots, f_{t-1})$ with $f_i : \mathbb{F}^n \times \mathbb{F}^{s_i} \to \mathbb{F}^{s_{i+1}} \times \mathbb{F}^{m_i}$, such that:*

- *The algorithm A given below satisfies, for any input vector $\vec{x} \in \mathbb{F}^n$, $A(\vec{x}) = C(\vec{x})$ (that is, A computes C);*

  function *$A(\vec{x})$*

  $\vec{x}_0 \leftarrow \vec{x}$

for $i = 0$ *to* $t-1$ ***do*** $(\vec{x}_{i+1}, \vec{y}_i) \leftarrow f_i(\vec{x}_i)$
$\vec{y} \leftarrow \vec{y}_0 || \cdots || \vec{y}_{t-1}$
return $\vec{y}$

- *For any* $i \in [\![0, t-1]\!]$, $j \leq s_{i+1} + m_i$, *the* j*-th output*[2] *of* $f_i : \mathbb{F}^n \times \mathbb{F}^{s_i} \mapsto \mathbb{F}^{s_{i+1}} \times \mathbb{F}^{m_i}$ *can be computed by a multivariate polynomial* $P_{i,j}$ *over* $\mathbb{F}^{2^k}$ *of degree* $\deg P_{i,j} \leq 2^k$*;*
- $\sum_{i=0}^{t-1} s_i \leq s/k$ *and* $\sum_{i=0}^{t-1} m_i = m$.

Proof. Let C be a layered boolean circuit with n inputs and m outputs, of size s and depth d, with layers $(L_1, \cdots, L_d)$. For $i = 1$ to d, we let w_i denote the width of the layer L_i (that is, the number of computation gates it contains; note that $s = \sum_{i=1}^{d} w_i$). Fix an integer k and let $t = \lceil d/k \rceil$.
We start by considering t 'chunks' of layers, each containing k consecutive layers (the last may in fact contain fewer if $k \nmid d$). Observe there must exist a $j \in [0, k-1]$, such that the sum of the widths of the j^{th} layer of each chunk (with the convention that if the last chunk has fewer than j layer, its j^{th} one is empty; $w_i = 0$ if $i > d$) is at most s/k, i.e. $\sum_{i=1}^{t} w_{k\cdot(i-1)+j} \leq s/k$. Indeed otherwise $\forall j \in [0, k], \sum_{i=0}^{t-1} w_{r_0+1+k\cdot(i-1)+j} > s/k$, so $s = \sum_{i=1}^{d} w_i \geq \sum_{i=r_0+1}^{d-r_1} w_i = \sum_{j=0}^{k-1} \sum_{i=1}^{t} w_{r_0+1+k\cdot(i-1)+j} > k \cdot s/k = s$, which is a contradiction. With j being fixed we now define $j_i \leftarrow k \cdot (i-1) + j$ for $i \in [t-1]$ and $j_t \leftarrow \min(d,\ k \cdot (t-1) + j)$.
Now, for each $0 \leq i \leq t$, we let B_i the block containing the consecutive layers $(L_{j_i}, \cdots, L_{j_{i+1}})$. Note that the depth of each block is at most k. Let m_i denote the number of output nodes contained in B_i (note that $\sum_{i=0}^{t-1} m_i = m$). For each $1 \leq i \leq t$, set $s_i \leftarrow w_{j_i}$, and $s_0, s_d \leftarrow 0$. This decomposition into blocks is illustrated in fig. 4.1.
The intuition behind the decomposition of C is the following: each function f_i will take as input the n inputs $\vec{x}$ to C, together with the s_i values of the gates in the first layer of B_i. It evaluates the layers of B_i, starting from the s_i values of the first layer (using the input $\vec{x}$ when the layer contains an input node), and outputs the s_{i+1} values on the first layer of B_{i+1}, together with the m_i values of the output nodes contained in B_i.
Given this decomposition, the correctness of algorithm A is guaranteed by definition: A simply corresponds to a "block-by-block" evaluation of the circuit C, where each block evaluation outputs the current state $\vec{x}_{i+1}$ (which must be given as input to the next block in addition to the input vector $\vec{x}$) together with the outputs of C contained in this block. Since each block has depth at most k, each output of f_i can be computed by multivariate polynomials with at most 2^k inputs, and of degree at most 2^k. □

4.1.2 Securely Computing C in the Correlated Randomness Model

We represent in fig. 4.2 the ideal functionality for securely evaluating the layered arithmetic circuit C.

Functionality $\mathcal{F}_{\mathsf{SFE}}(C)$

The functionality is parametrised with a number of parties N and an arithmetic circuit C with $n = \ell_1 + \cdots + \ell_N$ inputs and m outputs over a finite field $\mathbb{F}$.

Input: Wait to receive $(\texttt{input}, i, x_i)$ from each party P_i $(i \in [N])$, where $x_i \in \mathbb{F}^{\ell_i}$, and set $\vec{x} \leftarrow x_1 \| \ldots \| x_N$.

[2] i.e. the j^{th} coordinate of the image by f_i, seen as $f_i \colon \mathbb{F}^n \times \mathbb{F}^{s_i} \to \mathbb{F}^{s_{i+1}+m_i}$.

Output: Compute $\vec{y} \leftarrow C(\vec{x})$; Output $\vec{y}$ to all parties $P_1, \ldots, P_N$.

Figure 4.2: Ideal functionality $\mathcal{F}_{\mathsf{SFE}}(C)$ for securely evaluating an arithmetic circuit C among N parties.

We represent on fig. 4.3 an ideal functionality for distributing (function-dependent) correlated randomness between the parties.

Theorem 3. *Let $k \leq \log\log s - \log\log\log s$. There exists a protocol Π_C which (perfectly) securely implements the N-party functionality $\mathcal{F}_C$ in the $\mathcal{F}_{\mathsf{corr}}$-hybrid model, against a static, passive, non-aborting adversary corrupting at most $N-1$ out of N parties, with communication complexity upper bounded by $O(N \cdot (n + \frac{s}{k} + m) \cdot \log|\mathbb{F}|)$ and polynomial computation.*

Functionality $\mathcal{F}_{\mathsf{corr}}$

The functionality is parametrised with a number of parties N and an arithmetic circuit C with $n = \ell_1 + \cdots + \ell_N$ inputs and m outputs over a finite field $\mathbb{F}$.

Parameters: For every $i = 0, \ldots, \lceil d/k \rceil - 1$, functionality is parameterised with subsets $(U^{\mathsf{in}}_{i,j}, U_{i,j})_{1 \leq j \leq \lceil s_{i+1}/\beta \rceil}$ and $(V^{\mathsf{in}}_{i,j}, V_{i,j})_{1 \leq j \leq \lceil m_i/\beta \rceil}$.

Parties: An adversary $\mathcal{A}$ and N parties $P_1, \cdots, P_N$.
The functionality aborts if it receives any incorrectly formatted message.

1. On input a message $(\mathsf{corrupt}, D)$ with $D \subsetneq [N]$ from $\mathcal{A}$, set $H \leftarrow [N] \setminus D$ and store (H, D).
2. On input a message input with from each party P_ℓ, send ready to $\mathcal{A}$.
3. *Setup input masks:* On input a message $(\mathsf{setinputshare}, (\vec{r}_{\mathsf{in},\ell})_{\ell \in D})$ from $\mathcal{A}$ with $\forall \ell \in D, \vec{r}_{\mathsf{in},\ell} \in \mathbb{F}^n$, sample $(\vec{r}_{\mathsf{in},\ell})_{\ell \in H} \xleftarrow{\$} (\mathbb{F}^n)^{|H|}$, and set $\vec{r}_{\mathsf{in}} \leftarrow \sum_{\ell \in [N]} \vec{r}_{\mathsf{in},\ell}$.
4. For $i = 1$ to $\lceil d/k \rceil - 1$:
   (a) *Setup masks for the computation gates of the first layer of the i^{th} chunk:* On input a message $(\mathsf{setblockshare}, i, (\vec{r}_{i,\ell})_{\ell \in D})$ from $\mathcal{A}$ with $\forall \ell \in D, \vec{r}_{i,\ell} \in \mathbb{F}^{s_i}$, sample $(\vec{r}_{i,\ell})_{\ell \in H} \xleftarrow{\$} (\mathbb{F}^{s_i})^{|H|}$, and set $\vec{r}_{\mathsf{in}} \leftarrow \sum_{\ell \in [N]} \vec{r}_{\mathsf{in},\ell}$.
   (b) *Setup evaluation of the computation gates on the final layer of the i^{th} chunk:*
   - For $j = 1$ to $\lceil s^{i+1}/\beta \rceil$, set:
   $$\vec{\pi}^{(i,j)} \leftarrow \left(1 \parallel \vec{r}_{\mathtt{in}}[U^{\mathtt{in}}_{i,j}] \parallel \vec{r}_i[U_{i,j}]\right)^{\otimes 2^k}.$$
   - Wait for a message $(\mathsf{setshare}, (i,j), (\vec{\pi}^{(i,j)}_\ell)_{\ell \in D})$ from $\mathcal{A}$ with $\vec{\pi}^{(i,j)}_\ell \in \mathbb{F}^\delta$;
   - Compute uniformly random shares $(\vec{\pi}^{(i,j)}_\ell)_{\ell \in |H|}$ of $\vec{\pi}^{(i,j)} - \sum_{\ell \in D} \vec{\pi}^{(i,j)}_\ell$.
   (c) *Setup evaluation of the output gates in the i^{th} chunk:*

- For $j = 1$ to $\lceil m_i/\beta \rceil$, set:

$$\vec{\pi}^{(i,j)} \leftarrow \left(1 \parallel \vec{r}_{\mathtt{in}}[V^{\mathtt{in}}_{i,j}] \parallel \vec{r}_i[V_{i,j}]\right)^{\otimes 2^k}.$$

- Wait for a message $(\mathsf{setoutputshare}, (i,j), (\vec{\pi}^{(i,j)}_\ell)_{\ell \in D})$ from $\mathcal{A}$ with $\vec{\pi}^{(i,j)}_\ell \in \mathbb{F}^\delta$;
- Compute uniformly random shares $(\vec{\pi}^{(i,j)}_\ell)_{\ell \in |H|}$ of $\vec{\pi}^{(i,j)} - \sum_{\ell \in D} \vec{\pi}^{(i,j)}_\ell$.

5. Output $(\vec{r}_{\mathsf{in},\ell}, (\vec{r}_{i,\ell}, (\vec{\pi}^{(i,j)}_\ell)_{1 \le j \le \lceil s_{i+1}/\beta \rceil}, (\vec{\pi}^{(i,j)}_{\mathsf{out},\ell})_{1 \le j \le \lceil m_i/\beta \rceil})_{0 \le i < \lceil d/k \rceil})$ to each party P_ℓ.

Figure 4.3: Ideal corruptible functionality $\mathcal{F}_{\mathsf{corr}}$ to deal out correlated randomness to the parties.

The protocol follows closely the construction of [Cou19], with some tedious technical adaptations which are necessary to rely on the specific type of correlated randomness which we will manage to securely generate with low communication overhead. The rest of this section is dedicated to making Π_C explicit and to analysing its security.
In the sequel, we fix a layered arithmetic C with block decomposition:

$$(s_0, s_1, \cdots, s_{t-1}), (m_0, \cdots, m_{t-1}), \text{ and } (f_0, \cdots, f_{t-1}).$$

We now proceed with the description of the protocol Π_C, which securely implements $\mathcal{F}_C$ in the $\mathcal{F}_{\mathsf{corr}}$-hybrid model, with security against a static adversary passively corrupting at most $N-1$ parties. Fix parameters $\beta, k \in \mathbb{N}^*$. We let $\vec{x}_\ell$ denote the input vector of party P_ℓ over $\mathbb{F}$. We slightly abuse this notation and view each vector $\vec{x}_\ell$ as a length-n vector with zeroes at the positions were P_ℓ does not hold an input, so that the vectors $\vec{x}_\ell$ form additive shares of the input vector $\vec{x} = \sum_{\ell=1}^N \vec{x}_\ell$.
For each block B_i, we denote by $\vec{u}^i$ the values on the nodes of the first layer of B_i, and by $\vec{v}^i$ the values on all output nodes in B_i. Observe that by definition of f_i, we have $f_i(\vec{x}, \vec{u}^i) = (\vec{u}^{i+1}, \vec{v}^i)$. We further partition the outputs of f_i in subvectors $(\vec{u}^{i+1}_j)_{1 \le j \le \lceil s_{i+1}/\beta \rceil}$ and $(\vec{v}^i_j)_{1 \le j \le \lceil m_i/\beta \rceil}$, such that each subvector has length at most β. Recall that each output of f_i depends on at most 2^k inputs; therefore, each subvector $\vec{u}^{i+1}_j$ and $\vec{v}^i_j$ depends on at most $\beta \cdot 2^k$ coordinates of $(\vec{x}, \vec{u}^i)$. For each subvector $\vec{u}^{i+1}_j$ (resp. $\vec{v}^i_j$), we denote by $U^{\mathsf{in}}_{i,j} \subset [n]$ (resp. $V^{\mathsf{in}}_{i,j} \subset [n]$) the subset of coordinates of $\vec{x}$ which influence $\vec{u}^{i+1}_j$ (resp. $\vec{v}^i_j$), and by $U_{i,j} \subset [s_i]$ (resp. $V_{i,j} \subset [s_i]$) the subset of coordinates of $\vec{u}^i$ which influence $\vec{u}^{i+1}_j$ (resp. $\vec{v}^i_j$). Note that $|U^{\mathsf{in}}_{i,j}| + |U_{i,j}| \le \beta \cdot 2^k$ and $|V^{\mathsf{in}}_{i,j}| + |V_{i,j}| \le \beta \cdot 2^k$. This decomposition is illustrated in fig. 4.4.

4.1.2.1 Initialisation.

- Each party P_ℓ sends input to $\mathcal{F}_{\mathsf{corr}}$ and waits until it receives

$$(\vec{r}_{\mathsf{in},\ell}, (\vec{r}_{i,\ell}, (\vec{\pi}^{(i,j)}_\ell)_{1 \le j \le \lceil s_{i+1}/\beta \rceil}, (\vec{\pi}^{(i,j)}_{\mathsf{out},\ell})_{1 \le j \le \lceil m_i/\beta \rceil})_{0 \le i < t}).$$

- Each party P_ℓ broadcasts $\vec{z}^{\mathsf{in}}_\ell \leftarrow \vec{x}_\ell + \vec{r}_{\mathsf{in},\ell}$. All parties compute $\vec{z}^{\mathsf{in}} \leftarrow \sum_\ell \vec{z}^{\mathsf{in}}_\ell = \vec{x} + \vec{r}_{\mathsf{in}}$.

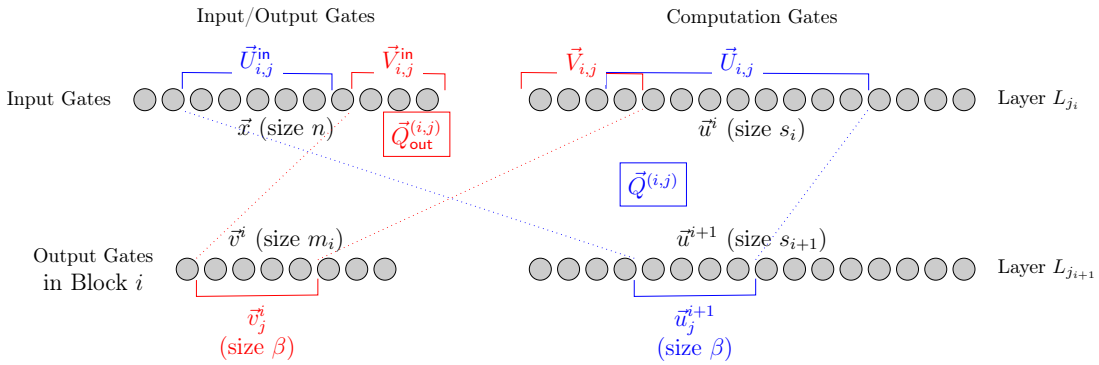


Figure 4.4: Decomposition of the i^{th} Block into low-degree Polynomials.

4.1.2.2 i-th Block Evaluation.

We now resume the description of the protocol, and assume that the protocol maintains the following invariant: at the beginning of the i-th block evaluation, each party P_ℓ holds an additive share $\vec{u}^i_\ell$ of the values $\vec{u}^i$ on the first layer of the block. The block decomposition of f_i guarantees that each output of $f_i : \mathbb{F}^n \times \mathbb{F}^{s_i} \mapsto \mathbb{F}^{s_{i+1}} \times \mathbb{F}^{m_i}$ can be computed by a multivariate polynomial of degree at most 2^k. Let $\vec{Q}^{(i,j)}$ denote the vector of multivariate polynomials of degree at most 2^k such that $\vec{Q}^{(i,j)}(\vec{x}[U^{\mathsf{in}}_{i,j}] \,||\, \vec{u}^i[U_{i,j}]) = \vec{u}^{i+1}_j \in \mathbb{F}^{s_{i+1}}$. Similarly, let $\vec{Q}^{(i,j)}_{\mathsf{out}}$ denote the vector of multivariate polynomials of degree at most 2^k such that $\vec{Q}^{(i,j)}_{\mathsf{out}}(\vec{x}[V^{\mathsf{in}}_{i,j}] \,||\, \vec{u}^i[V_{i,j}]) = \vec{v}^i_j \in \mathbb{F}^{m_i}$.

- Each party P_ℓ broadcasts $\vec{z}^i_\ell \leftarrow \vec{u}^i_\ell + \vec{r}_{i,\ell}$. All parties compute $\vec{z}^i \leftarrow \sum_\ell \vec{z}_\ell = \vec{u}^i + \vec{r}_{i,\ell}$.

- Define the following vectors of multivariate polynomials:

$$\vec{\hat{Q}}^{(i,j)}(\vec{X}) \leftarrow \vec{Q}^{(i,j)}(\vec{X} - (\vec{r}_{\mathsf{in}}[U^{\mathsf{in}}_{i,j}] \,||\, \vec{r}_i[U_{i,j}]))$$
$$\vec{\hat{Q}}^{(i,j)}_{\mathsf{out}}(\vec{X}) \leftarrow \vec{Q}^{(i,j)}_{\mathsf{out}}(\vec{X} - (\vec{r}_{\mathsf{in}}[V^{\mathsf{in}}_{i,j}] \,||\, \vec{r}_i[V_{i,j}])).$$

  Observe that each coefficient of $\vec{\hat{Q}}^{(i,j)}(\vec{X})$ is itself a multivariate polynomial of degree at most 2^k in the coordinates of $(\vec{r}_{\mathsf{in}}[U^{\mathsf{in}}_{i,j}] \,||\, \vec{r}_i[U_{i,j}])$. Therefore, all its coefficients can be computed as linear combinations of the coordinates of $\vec{\pi}^{(i,j)} = (1_{\mathbb{F}} \,||\, \vec{r}_{\mathsf{in}}[U^{\mathsf{in}}_{i,j}] \,||\, \vec{r}_i[U_{i,j}])^{\otimes 2^k}$. Similarly, all coefficients of $\vec{\hat{Q}}^{(i,j)}_{\mathsf{out}}(\vec{X})$ can be computed as linear combinations of the coordinates of $\pi^{(i,j)}_{\mathsf{out}} = (1_{\mathbb{F}} \,||\, \vec{r}_{\mathsf{in}}[V^{\mathsf{in}}_{i,j}] \,||\, \vec{r}_i[V_{i,j}])^{\otimes 2^k}$.

- Each party P_ℓ computes shares $(\vec{\hat{Q}}^{(i,j)}_\ell(\vec{X}), \vec{\hat{Q}}^{(i,j)}_{\mathsf{out},\ell}(\vec{X}))$ of $(\vec{\hat{Q}}^{(i,j)}(\vec{X}), \vec{\hat{Q}}^{(i,j)}_{\mathsf{out}}(\vec{X}))$ from his shares $(\vec{\pi}^{(i,j)}_\ell, \vec{\pi}^{(i,j)}_{\mathsf{out},\ell})$ of $(\vec{\pi}^{(i,j)}, \vec{\pi}^{(i,j)}_{\mathsf{out}})$, and sets:

$$\vec{u}^{i+1}_\ell \leftarrow \left(\vec{\hat{Q}}^{(i,j)}_\ell(\vec{z}^{\mathsf{in}}[U^{\mathsf{in}}_{i,j}], \vec{z}^i[U_{i,j}])\right)_{1 \le j \le \lceil s_{i+1}/\beta \rceil}$$
$$\vec{v}^i_\ell \leftarrow \left(\vec{\hat{Q}}^{(i,j)}_{\mathsf{out},\ell}(\vec{z}^{\mathsf{in}}[V^{\mathsf{in}}_{i,j}], \vec{z}^i[V_{i,j}])\right)_{1 \le j \le \lceil m_i/\beta \rceil}.$$

4.1.2.3 Output Reconstruction.

Each party P_ℓ sets and broadcasts $\vec{v}_\ell$, where:

$$\vec{v}_\ell \leftarrow (\vec{v}^{(0,j)}_\ell)_{j \le q_0} || (\vec{v}^{(1,j)}_\ell)_{1 \le j \le \lceil m_1/\beta \rceil} || \cdots || (\vec{v}^{(\lceil d/k \rceil - 1, j)}_\ell)_{1 \le j \le \lceil m_{\lceil d/k \rceil - 1}/\beta \rceil}.$$

All parties reconstruct and output $\vec{v} \leftarrow \sum_\ell \vec{v}_\ell$.
We now prove Theorem 3 by proving the above protocol satisfies the necessary requirements.

Proof. The efficiency of the protocol follows by inspection: the total length of all messages broadcast in Π_C is

$$N \cdot \left(n + \sum_{i=0}^{t-1} s_i + \sum_{i=0}^{t-1} m_i \right) \leq N \cdot \left(n + \frac{s}{k} + m \right).$$

We now analyse the security of Π_C. We describe in fig. 4.5 a simulator Sim which perfectly simulates an execution of Π_C.

Simulator Sim

Let $D \subsetneq [N]$ be the subset of statically corrupted parties, and $H = [N] \setminus D$ be the (nonempty) subset of honest parties.

- **Initialisation.**
  - Sim honestly simulates the functionality $\mathcal{F}_{\mathsf{corr}}$ and stores the following values:

$$(\vec{r}_{\mathsf{in},\ell}, (\vec{r}_{i,\ell}, (\vec{\pi}_\ell^{(i,j)})_{1 \leq j \leq \lceil s_{i+1}/\beta \rceil}, (\vec{\pi}_{\mathsf{out},\ell}^{(i,j)})_{1 \leq j \leq \lceil m_i/\beta \rceil})_{0 \leq i < \lceil d/k \rceil})_{\ell \in [N]}.$$

  - Sim broadcasts a random vector $\vec{z}_\ell^{\mathsf{in}}$ on behalf of all honest parties. Let $\vec{z}^{\mathsf{in}} \leftarrow \sum_{\ell \leq N} \vec{z}_\ell^{\mathsf{in}}$. For each $\ell \in D$, Sim extracts $\vec{x}_\ell \leftarrow \vec{z}_\ell^{\mathsf{in}} - \vec{r}_{\mathsf{in},\ell}$.
  - Sim sends $(\mathsf{input}, \vec{x}_\ell)$ to $\mathcal{F}_C$ on behalf of all corrupted parties, and receives an output $\vec{v}$.
- **i-th Block Evaluation.**

  Sim broadcasts uniformly random $\vec{z}_\ell^i \xleftarrow{\$} \mathbb{F}^{s_i}$ on behalf of all honest parties, waits to receive $(\vec{z}_\ell^i)_{\ell \in D}$, and sets $\vec{z}^i \leftarrow \sum_{\ell \in [N]} \vec{z}_\ell^i$.
- **Output Phase.**

  For each $\ell \in D$, Sim computes $\vec{v}_\ell^i \leftarrow \vec{\hat{Q}}_{\mathsf{out},\ell}^{(i,j)}(\vec{z}^{\mathsf{in}}[V_{i,j}^{\mathsf{in}}], \vec{z}^i[V_{i,j}])$ and sets $\vec{v}_\ell \leftarrow \vec{v}_\ell^0 || \cdots || \vec{v}_\ell^{t-1}$. Eventually, Sim broadcasts uniformly random shares of $\vec{v} - \sum_{\ell \in D} \vec{v}_\ell$ on behalf of the honest parties.

Figure 4.5: Simulator Sim for the sublinear protocol in the $\mathcal{F}_{\mathsf{corr}}$-hybrid model.

It remains to show that Sim perfectly simulates a run of the real protocol Π_C in the $\mathcal{F}_{\mathsf{corr}}$-hybrid model. But this follows almost immediately by inspection, as the view of all corrupted parties in Π_C contains exactly:

- the tuple $(\vec{r}_{\mathsf{in},\ell}, (\vec{r}_{i,\ell}, (\vec{\pi}_\ell^{(i,j)})_{1 \leq j \leq \lceil s_{i+1}/\beta \rceil}, (\vec{\pi}_{\mathsf{out},\ell}^{(i,j)})_{1 \leq j \leq \lceil m_i/\beta \rceil})_{0 \leq i < t})_{\ell \in D}$, which is chosen by the adversary;
- the vectors $(\vec{z}_\ell^{\mathsf{in}} = \vec{x}_\ell + \vec{r}_{\mathsf{in},\ell})_{\ell \in H}$, which are perfectly random since each $\vec{r}_{\mathsf{in},\ell}$ is perfectly random, by definition of $\mathcal{F}_{\mathsf{corr}}$;

- the vectors $(\vec{z}_\ell^i = \vec{u}_\ell^i + \vec{r}_\ell^i)_{\ell \in H}$, which are perfectly random since each $\vec{r}_\ell^i$ is perfectly random, by definition of $\mathcal{F}_{\mathsf{corr}}$;

- the vectors $(\vec{v}_\ell)_{\ell \in H} = (\vec{v}_\ell^0 || \cdots || \vec{v}_\ell^{t-1})_{\ell \in H}$. By construction, and by definition of the $\vec{\hat{Q}}_{v,\ell}^{(i,j)}(\vec{X})$, the $\vec{v}_\ell$ satisfy:

$$\begin{aligned}\vec{v}_\ell^{(i,j)} &= \vec{\hat{Q}}_{\mathsf{out},\ell}^{(i,j)}(\vec{w}[V_{i,j}^{\mathsf{in}}], \vec{z}^i[V_{i,j}]) \\ &= \vec{Q}_{\mathsf{out},\ell}^{(i,j)}((\vec{w}[V_{i,j}^{\mathsf{in}}], \vec{z}^i[V_{i,j}]) - (\vec{r}[V_{i,j}^{\mathsf{in}}], \vec{r}^i[V_{i,j}])) \\ &= \vec{Q}_{\mathsf{out},\ell}^{(i,j)}(\vec{x}[V_{i,j}^{\mathsf{in}}], \vec{u}^i[V_{i,j}])).\end{aligned}$$

  Therefore, the $\vec{v}_\ell$ are uniformly random conditioned on:

$$\sum_{\ell \in H} \vec{v}_\ell^{(i,j)} = \vec{Q}_{\mathsf{out}}^{(i,j)}(\vec{x}[V_{i,j}^{\mathsf{in}}], \vec{u}^i[V_{i,j}]) - \sum_{\ell \in D} \vec{v}_\ell^{(i,j)}.$$

  By definition of the $\vec{Q}_{\mathsf{out}}^{(i,j)}(\vec{X})$, we have $\vec{Q}_{\mathsf{out}}^{(i,j)}(\vec{x}[V_{i,j}^{\mathsf{in}}], \vec{u}^i[V_{i,j}]) = \vec{v}^{(i,j)}$, where the $\vec{v}^{(i,j)}$ are such that $f_i(\vec{x}, \vec{u}^i) = (\vec{u}^{i+1}, (\vec{v}^{(i,j)})_{1 \le j \le \lceil m_i/\beta \rceil})$. Therefore, by definition of the f_i, the vectors $\vec{v}_\ell$ are uniformly random conditioned on:

$$\sum_{\ell \in H} \vec{v}_\ell = \vec{v} - \sum_{\ell \in D} \vec{v}_\ell, \text{ where } \vec{v} = C(\vec{x}).$$

This concludes the proof that Sim perfectly simulates Π_C. □

4.2 Generating the Correlated Randomness from Quasi-Polynomial LPN

In this section, we construct a protocol Π_{corr}, which implements the ideal functionality $\mathcal{F}_{\mathsf{corr}}$ with small communication, under the quasi-polynomial LPN assumption. A very natural approach to realise a functionality that distributes correlated random coins using a small amount of communication is to rely on *pseudorandom correlation generators*, a primitive recently defined an constructed (for various types of correlations, and under a variety of assumptions) in [BCG+19b]. At a high level, [BCG+19b] suggests to distribute correlated randomness with the following approach:

- Use a generic secure computation protocol Π_{Gen} to distributively execute the $\mathsf{PCG.Gen}$ functionality of the pseudorandom correlation generator. Note that $\mathsf{PCG.Gen}$ outputs short seeds, much smaller than the correlated pseudo-random strings which can be stretched from these seeds. Therefore, Π_{Gen} can potentially have a relatively high communication overhead in its inputs and outputs, while maintaining the overall communication overhead of Π_{corr} small.

- Expand the distributively generated seeds locally using the Expand algorithm of the PCG. Each such string is guaranteed, by the security of the PCG, to be indistinguishable (from the viewpoint of the other parties) from a uniformly random string sampled conditioned on satisfying the target correlation with the expanded strings held by the other parties.

While this approach does not necessarily leads to a secure implementation of an ideal functionality generating correlated random coins, it was shown in [BCG+19b] (Theorem 19 in [BCG+19b]) that it provides a provably secure implementation for all *corruptible* ideal functionalities for distributing correlated random coins. Note that this property is satisfied by our functionality $\mathcal{F}_{\mathsf{corr}}$. Our protocol Π_{corr} will follow this approach. We start by constructing a pseudorandom correlation generator for the type of correlated randomness produced by $\mathcal{F}_{\mathsf{corr}}$, building upon an LPN-based construction of [BCG+19b].

4.2.1 Substrings Tensor Powers Correlations (stp)

We now describe our construction of a PCG for generating the type of correlated randomness produced by $\mathcal{F}_{\mathsf{corr}}$. As all constructions of [BCG+19b], our construction will be restricted to the two-party setting; hence, we focus on $N = 2$ parties from now on. Abstracting out the unnecessary details, the functionality $\mathcal{F}_{\mathsf{corr}}$ does the following. It is parametrised with a vector length w, subsets $(S_i)_{1\leq i\leq \mathsf{n_s}} \in \binom{[w]}{\leq K}^{\mathsf{n_s}}$, a tensor power parameter tpp, and generates shares of:

$$(\vec{r}, ((1_{\mathbb{F}} \,||\, \vec{r}[S_i])^{\otimes \mathsf{tpp}})_{1\leq i\leq \mathsf{n_s}}), \text{ where } \vec{r} \in \mathbb{F}^w \text{ is random.}$$

We call $\mathcal{C}$ the correlation generator associated with $\mathcal{F}_{\mathsf{corr}}$, i.e. the PPT algorithm that, on input the security parameter in unary 1^λ, samples correlated random string as above (where the parameters $(\mathsf{n_s}, K, \mathsf{tpp})$ are functions of λ). It is straightforward to see that $\mathcal{C}$ is a reverse-samplable correlation generator (see Definition 6), since it is an additive correlation: given any fixed share share_0, a matching share can be reverse-sampled by sampling $\vec{r}$ and setting $\mathsf{share}_1 \leftarrow (\vec{r}, ((1_{\mathbb{F}} \,||\, \vec{r}[S_i])^{\otimes \mathsf{tpp}})_{1\leq i\leq \mathsf{n_s}}) - \mathsf{share}_0$. We call this type of correlated randomness a *subsets tensor powers* (stp). Below, we describe a pseudorandom correlation generator for such correlations.

4.2.2 Good Cover

Before we proceed with the description of a PCG to generate such correlations, we need to introduce a concept, that of a *good cover*. The notations in this subsection are completely self-contained, and may conflict with the parameters defined for the main protocol. In the course of our construction we will want to solve the following problem: given a vector $\vec{v}$ of size n, a family $(S_i)_{i\in[t]} \in \mathcal{P}([n])^t$ of t (*short*) subsets of coordinates of $\vec{v}$, and a (*small*) bound $B > 0$, the problem is to find a family $(\vec{v}_j)_{j\in[M]}$ of some number m of size-K subvectors of $\vec{v}$ such that:

1. The subvectors collectively cover $\vec{v}$;
2. For each $i \in [t]$, there are at most B subvectors in $(\vec{v}_j)_{j\in[M]}$ whose coordinates intersect S_i.

We call such a family a *B-Good Cover* of $(\vec{v}, (S_i)_{i\in[t]})$. First of all we note that the values of the vectors and subvectors do not matter, so we will conflate them with sets and subsets (of coordinates) for simplicity, which leads to a more natural formulation.

Definition 14 (Good Cover – Set Formulation)**.** *Let $n, B, K, t, q, M \in \mathbb{N}$ and $(S_i)_{i\in[t]} \in \binom{[n]}{\leq q}^t$ a family of t subsets of $[n]$ of size at most q each. A family $A = (\vec{\alpha}^j)_{j\in[M]} \in \binom{[n]}{K}^M$ is a* B-Good Cover *of $(S_i)_{i\in[t]}$ if:*

1. *A* covers *$[n]$: $\bigcup_{j=1}^{M} \vec{\alpha}^j = [n]$*

2. Each S_i intersects at most B elements of A: $\forall i \in [t], |\{j \in [M] \colon \vec{\alpha}^j \cap S_i \neq \varnothing\}| \leq B$.

We abusively conflate the two views, where a good cover is just a family of subsets $A \in \binom{[n]}{K}^M$ and where the good cover is a family of sparse vectors—given by a set of coordinates and a short vector of values—$A \in (\binom{[n]}{K} \times \mathbb{F}^K)^M$.

Lemma 2 (Random Covers are Good Covers.)**.** *Let $n, \kappa, \kappa' \in \mathbb{N} \setminus \{0, 1\}$, and $(S_i)_{i\in[t]} \in \binom{[n]}{\leq q}^t$ a family of t subsets of $[n]$ of size at most q each. Let $A = (\vec{\alpha}^j)_{j\in[M]} \in \binom{[n]}{K}^M$ be a sequence of M* i.i.d. *uniform random size-K subsets of $[n]$, with $M = \kappa \cdot n \ln n/K$. Let $B \leftarrow \kappa'\kappa \cdot q \cdot \ln n$.*
It holds that $A = (\vec{\alpha}^j)_{j\in[M]}$ is a B-Good Cover of $(S_i)_{i\in[t]}$ with probability at least:

$$1 - \frac{1}{n^{\kappa-1}} - \frac{t}{n^{(\kappa'-2)\kappa\cdot q/2}}.$$

The proof, which is obtained in a straightforward fashion by combining the union and Chernoff bounds, is given below.

Proof.

1. The probability that $i \notin A$ (for any $i \in [n]$) is equal to $(1 - \binom{n-1}{K-1}/\binom{n}{K})^M = (1 - \frac{K}{n})^M = e^{M\cdot\ln(1-K/n)} \leq e^{-MK/n} = n^{-\kappa}$ (using $\forall x \geq -1, \ln(1+x) \leq x$), so by union bound, the probability that A does not cover $[n]$ is at most $n \cdot n^{-\kappa}$.

2. For each $i \in [q]$, we arbitrarily extend S_i to an $\tilde{S}_i$ of size exactly q; in particular $\forall i \in [q], S_i \subseteq \tilde{S}_i$ so $|\{j \colon \vec{\alpha}^j \cap S_i \neq \varnothing\}| \leq |\{j \colon \vec{\alpha}^j \cap \tilde{S}_i \neq \varnothing\}|$. For any $j \in [M]$, the indicator random variable of the event $\{\vec{\alpha}^j \cap \tilde{S}_i \neq \varnothing\}$ follows a Bernouilli law with parameters $(M, \frac{q}{p})$, where $p = 1 - \binom{n-q}{K}/\binom{n}{K}$. Its expectancy is $\mu \coloneqq \mathbb{E}(X) = M \cdot (1 - \binom{n-q}{K}/\binom{n}{K})$. Note that, using Bernouilli's inequality ($\forall r \in \mathbb{N}^*, \forall x \geq -1, (1+x)^r > (1+rx)$):

$$\begin{aligned}\mu = M \cdot \left(1 - \frac{\prod_{j=0}^{K-1}(n-q-j)}{\prod_{j=0}^{K-1}(n-j)}\right) &\leq m \cdot (1 - \frac{(n-2q)^K}{n^K}) \\ &= M \cdot (1 - (1-2q/n)^K) \leq M\frac{2qK}{n} = 2\kappa \cdot q \cdot \ln n.\end{aligned}$$

   Therefore the probability that $X \coloneqq |\{j \colon \alpha_j \in \tilde{S}_i\}| \leq B$ is, by (a looser version of the multiplicative form of) Chernoff's bound, at most:

$$\forall \delta \geq 0, \Pr[X > (1+\delta)\mu] \leq \exp\left(-\frac{\delta^2\mu}{2+\delta}\right).$$

   Setting $\delta \leftarrow B/\mu - 1$ yields:

$$\begin{aligned}\Pr[X > B] \leq \exp\left(-\frac{\delta^2\mu}{2+\delta}\right) \leq \exp(-\delta\mu/2) = \exp\left(-\frac{B-\mu}{2}\right) \\ \leq n^{-(\kappa'-2)\kappa\cdot q/2}.\end{aligned}$$

3. The desired bound is then obtained by union bound.

□

4.2.3 PCG for Subsets Tensor Powers ($\mathsf{PCG}_{\mathsf{stp}}$)

We now proceed with the description of a pseudorandom correlation generator for subsets tensor powers.

PCG for Low-Degree Polynomials from [BCG+19b]. We start by recalling a natural variant of pseudorandom correlation generator of [BCG+19b, Section 6], which generates shares of $\vec{r}^{\otimes \mathsf{tpp}}$, for a parameter tpp and a pseudorandom $\vec{r}$. It relies on the xLPN assumption with dimension n, number of samples $n' > n$, and a number λ of noisy coordinates. In our instantiation, we will typically consider $n' = O(n)$, e.g. $n' = 12n$; this corresponds to a particularly conservative variant of LPN with a very limited number of samples, and is equivalent to the hardness of decoding a random constant-rate linear code (which is known as the *syndrome decoding* problem). All known attacks on the syndrome decoding problem for constant-rate codes have complexity $2^{O(\lambda)}$. The PCG of [BCG+19b] is parametrised by integers $1^\lambda, n, n', \lambda, \mathsf{tpp} \in \mathbb{N}$ (where $n' > n$), a field $\mathbb{F}$, and a random parity-check matrix $H_{n',n} \xleftarrow{\$} \mathbb{F}^{(n'-n)\times n'}$.

PCG for degree-tpp Polynomial Correlations

PCG.Gen: On input 1^λ:

1. Pick a random λ-sparse vector $\vec{e} \xleftarrow{\$} \mathsf{HW}_\lambda(\mathbb{F}^{n'})$. Note that $\vec{e}^{\otimes \mathsf{tpp}} \in \mathsf{HW}_{\lambda^{\mathsf{tpp}}}(\mathbb{F}^{(n')^{\mathsf{tpp}}})$. Let $f : [(n')^{\mathsf{tpp}}] \mapsto \mathbb{F}$ be the multi-point function with λ^{tpp} points, such that $f(i)$ returns the i-th coordinate of $\vec{e}^{\otimes \mathsf{tpp}}$.
2. Compute $(K_0^{\mathsf{fss}}, K_1^{\mathsf{fss}}) \xleftarrow{\$} \mathsf{MPFSS.Gen}(1^\lambda, f)$. Output $\mathsf{k}_0 \leftarrow (n, K_0^{\mathsf{fss}})$ and $\mathsf{k}_1 \leftarrow (n, K_1^{\mathsf{fss}})$.

PCG.Expand: On input $(\sigma, \mathsf{k}_\sigma)$, compute $\vec{v}_\sigma \leftarrow \mathsf{MPFSS.FullEval}(\sigma, K_\sigma^{\mathsf{fss}})$ in $\mathbb{F}^{(n')^{\mathsf{tpp}}}$ and set $\vec{r}_\sigma \leftarrow H_{n',n}^{\otimes \mathsf{tpp}} \cdot \vec{v}_\sigma$. Output $\vec{r}_\sigma$.

Figure 4.6: Pseudorandom Correlation Generator for Low-Degree Polynomials from [BCG+19b].

Correctness follows from the fact that $\vec{v}_0 + \vec{v}_1 = \vec{e}^{\otimes \mathsf{tpp}}$ by the correctness of MPFSS, and $H_{n',n}^{\otimes \mathsf{tpp}} \cdot \vec{e}^{\otimes \mathsf{tpp}} = (H_{n',n} \cdot \vec{e})^{\otimes \mathsf{tpp}}$ by multilinearity of the tensor product. Hence, denoting $\vec{r} = H_{n',n} \cdot \vec{e}$, it holds that $\vec{r}_0 + \vec{r}_1 = \vec{r}^{\otimes \mathsf{tpp}}$. For security, we must show that the following distributions are indistinguishable for any $\sigma = 0, 1$:

$$\begin{aligned}
&\{(\mathsf{k}_\sigma, \vec{r}_{1-\sigma}) \ : \ (\mathsf{k}_0, \mathsf{k}_1) \xleftarrow{\$} \mathsf{Gen}(1^\lambda), \vec{r}_{1-\sigma} \leftarrow \mathsf{Expand}(1-\sigma, \mathsf{k}_{1-\sigma})\} \\
\overset{\mathrm{c}}{\approx} &\{(\mathsf{k}_\sigma, \vec{r}_{1-\sigma}) \ : \ (\mathsf{k}_0, \mathsf{k}_1) \xleftarrow{\$} \mathsf{Gen}(1^\lambda), \vec{r}_\sigma \leftarrow \mathsf{Expand}(\sigma, \mathsf{k}_\sigma), \vec{r} \xleftarrow{\$} \mathbb{F}^n, \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \vec{r}_{1-\sigma} \leftarrow \vec{r}^{\otimes \mathsf{tpp}} - \vec{r}_\sigma\}
\end{aligned}$$

Proof. We sketch the analysis for the sake of completeness; the full proof is given in [BCG+19b]. Security is shown with the following sequence of hybrids: first generate $(\mathsf{k}_\sigma, \vec{r}_{1-\sigma})$ as in the first distribution above. Then, generate $(\mathsf{k}_\sigma, \vec{r}_{1-\sigma})$ as before, and generate an alternative key k'_σ solely from the parameters $(1^\lambda, \mathbb{F}, n, n', t, \mathsf{tpp})$, using the simulator of the MPFSS. Output $(\mathsf{k}'_\sigma, \vec{r}_{1-\sigma})$; under the security of the MPFSS, this distribution is indistinguishable from the previous one. Note that k'_σ does not

depend anymore on the noise vector $\vec{e}$. In the next hybrid, generate $\vec{r} \xleftarrow{\$} H_{n',n} \cdot \vec{e}$ and set $\vec{r}_{1-\sigma} \leftarrow \vec{r}^{\otimes \mathsf{tpp}} - \mathsf{Expand}(\sigma, \mathsf{k}_\sigma)$; this game is perfectly indistinguishable from the previous one. Finally, replace $\vec{r} \xleftarrow{\$} H_{n',n} \cdot \vec{e}$ by $\vec{r} \xleftarrow{\$} \mathbb{F}^n$; under the LPN assumption, this last game (which correspond exactly to the second distribution) is computationally indistinguishable from the previous one, and security follows. □

Our New PCG. We now describe a variant of the above PCG, tailored to computing the tensor powers of many short subsets. The PCG is parametrised by $(S_i)_{i\in[K]} \in \binom{[w]}{\leq K}^{\mathsf{n_s}}$, $\mathsf{n_s}$ subsets of at most K indices taken from $[w]$. We assume for simplicity, but morally without loss of generality[3], that $\bigcup_{i=1}^{\mathsf{n_s}} S_i = [w]$. Our goal is for the parties to obtain shares of some pseudorandom vector $\vec{r} \in \mathbb{F}^w$ as well as shares of $(1 \mid\mid \vec{r}[S_i])^{\otimes \mathsf{tpp}} \in \mathbb{F}^{w \cdot \mathsf{tpp}}$ for each $i \in [\mathsf{n_s}]$.

We start by generating a B-good cover (for some integer B) of the $(S_i)_i$ of the form $(\alpha_j, \vec{r}_j)_{j\in[m]} \in (\binom{[w]}{\theta} \times \mathbb{F}^\theta)^m$ where each $\vec{r}_j$ is pseudorandom. We generate each of the m pseudorandom masks $\vec{r}_j$ using a different instance of xLPN, i.e. $\vec{r}_j \leftarrow H_j \cdot \vec{e}_j$, where $\vec{e}_j \in \mathbb{F}^{\theta'}$ is λ-sparse and $H_j \xleftarrow{\$} \mathbb{F}^{\theta \times \theta'}$ for some $\theta' = O(\theta)$. For each S_i, we denote $I_i \coloneqq \{j \in [m] \colon \alpha_j \cap S_i \neq \varnothing\} = \{j_1, \ldots, j_{|I_i|}\}$ the set of the indices of the masks which 'intersect' with S_i. Note that $\forall i \in [\mathsf{n_s}], |I_i| \leq B$ by definition of a B-good cover. We can now proceed with our main goal: generating shares of a subsets tensor powers correlation.

We define $\vec{r} \coloneqq \sum_{j=1}^m f_{\alpha_j, \vec{r}_j} \in \mathbb{F}^w$, where $f_{\alpha_j,\vec{r}_j} \in \mathbb{F}^w$ is the sparse vector defined by $(f_{\alpha_j,\vec{r}_j})_{|\alpha_j} = \vec{r}_j$ (and which is equal to $0_\mathbb{F}$ on $[w] \setminus \alpha_j$). Since $\bigcup_{i=1}^{\mathsf{n_s}} S_i = [w]$ and each of the $\vec{r}_j$ is pseudorandom, $\vec{r}$ is also pseudorandom.

Note that for any given $i \in [\mathsf{n_s}]$, $(1_\mathbb{F} \mid\mid \vec{r}[S_i])$ is a subvector of the vector $\tilde{\vec{r}}_i$ obtained by multiplying the block-diagonal matrix $H'_i = \mathsf{Diag}((1_\mathbb{F}), H_{j_1}, \ldots, H_{j_{|I_i|}})$ with the vector $\vec{e}'_i = (1_\mathbb{F} || e_{j_1} || \cdots || e_{j_{|I_i|}})$. Therefore for any tensor power tpp (i.e. the degree of the polynomial correlation), $\tilde{\vec{r}}_i^{\otimes \mathsf{tpp}} = (H'_i \cdot \vec{e}'_i)^{\otimes \mathsf{tpp}} = (H'_i)^{\otimes \mathsf{tpp}} \cdot (\vec{e}'_i)^{\otimes \mathsf{tpp}}$. If the parties use an MPFSS scheme to generate small seeds which expand to $(\vec{e}'_i)^{\otimes \mathsf{tpp}}$, they can then locally obtain shares of $\tilde{\vec{r}}_i^{\otimes \mathsf{tpp}}$ (since $(H'_i)^{\otimes \mathsf{tpp}}$ is public), and therefore of $(1_\mathbb{F} \mid\mid \vec{r}[S_i])^{\otimes \mathsf{tpp}}$. From all these shares of all the $(1_\mathbb{F} \mid\mid \vec{r}[S_i])^{\otimes \mathsf{tpp}}, i \in [\mathsf{n_s}]$ the parties can locally extract shares of all the $\vec{r}[S_i]$ and thence shares of $\vec{r}$ (since $\bigcup_{i=1}^{\mathsf{n_s}} S_i = [w]$). The protocol is given in Figure 4.7.

PCG for subset tensor powers $\mathsf{PCG_{stp}}$

Parameters: $w, \mathsf{tpp}, \lambda \in \mathbb{N}$ and $(S_i)_{1\leq i\leq \mathsf{n_s}} \subseteq [w]^{\mathsf{n_s}}$.

Gen: On input 1^λ:

1. Generate a family of subsets $(\alpha_j)_{1\leq j\leq m} \in \binom{[m]}{\theta'}^m$ which form a B-good cover of the $(S_i)_{i\in[\mathsf{n_s}]}$ (when the α_j are paired with length-θ' vectors in $\mathbb{F}^{\theta'}$), and contracting matrices[a] $(H_j)_{j\in[m]} \in (\mathbb{F}^{\theta\times\theta'})^m$.
2. Pick m random λ-sparse vectors $\vec{e}_j \xleftarrow{\$} \mathsf{HW}_\lambda(\mathbb{F}^{\theta'}), j \in [m]$ and define:

$$\vec{r}_j \leftarrow H_j \cdot \vec{e}'_j, \text{ for all } j \in [m].$$

[3] If $\bigcup_{i=1}^{\mathsf{n_s}} S_i \neq \varnothing$, and with the notations of the rest of the section, the vector $\vec{r}$ we generate is equal to $0_\mathbb{F}$ on $[w] \setminus \bigcup_{i=1}^{\mathsf{n_s}} S_i$, hence not pseudorandom. However, we can simply have the parties generate another mask $\vec{r}' = H' \cdot \vec{e}'$, pseudorandom under xLPN, to cover $[w] \setminus \bigcup_{i=1}^{\mathsf{n_s}} S_i$. Since the parties do not need shares of $(\vec{r}')^{\otimes \mathsf{tpp}}$, the communication complexity of generating the λ-sparse $\vec{e}'$ using an MPFSS is not an issue.

3. For each $i = 1 \dots \mathsf{n_s}$:

   (a) Denoting $I_i \coloneqq \{j \in [m] \colon \alpha_j \cap S_i \neq \varnothing\} = \{j_1, \cdots, j_{m_i}\}$ (with $m_i \leq B$), set:
   $$\tilde{\vec{r}}_i \leftarrow (1_{\mathbb{F}} \,||\, H_{j_1} \cdot \vec{e}^{\intercal}_{j_1} \,||\, \cdots \,||\, H_{j_{m_i}} \cdot \vec{e}^{\intercal}_{j_{m_i}})^{\intercal}.$$

   (b) Let $f_i : [(1 + m_i \cdot \theta')^{\mathsf{tpp}}] \to \mathbb{F}$ be the multi-point function with $(1 + m_i \cdot \lambda)^{\mathsf{tpp}}$ points, such that $f_i(x) = (1_{\mathbb{F}}||\vec{e}_{j_1}|| \cdots ||\vec{e}_{j_{m_i}})^{\otimes \mathsf{tpp}}[x]$. Compute $(K^{\mathsf{fss}}_{i,0}, K^{\mathsf{fss}}_{i,1}) \xleftarrow{\$} \mathsf{MPFSS.Gen}(1^\lambda, f_i)$.

4. Output $\mathsf{k}_0 \leftarrow (w, (K^{\mathsf{fss}}_{i,0})_{i \leq \mathsf{n_s}})$ and $\mathsf{k}_1 \leftarrow (w, (K^{\mathsf{fss}}_{i,1})_{i \leq \mathsf{n_s}})$.

Expand: On input $(\sigma, \mathsf{k}_\sigma)$, parse k_σ as $(w, (K^{\mathsf{fss}}_{i,\sigma})_{i \leq \mathsf{n_s}})$.

1. For each $i = 1 \dots \mathsf{n_s}$:

   Set $H'_i \leftarrow \mathsf{Diag}((1_{\mathbb{F}}), H_{j_1}, \dots, H_{j_{m_i}})$, compute

   $$\vec{v}_{i,\sigma} \leftarrow \mathsf{MPFSS.FullEval}(\sigma, K^{\mathsf{fss}}_{i,\sigma}) \in \mathbb{F}^{(1+m_i\lambda)^{\mathsf{tpp}}}$$

   and set $\vec{y}_\sigma \leftarrow ((H'_i)^{\otimes \mathsf{tpp}} \cdot \vec{v}_\sigma)_{1 \leq i \leq \mathsf{n_s}}$.

2. Extract from $\vec{y}_\sigma$ the appropriate linear combinations of its elements corresponding to a share of $(\vec{r}, ((1_{\mathbb{F}} \,||\, \vec{r}[S_i])^{\otimes tpp})_{i \in [\mathsf{n_s}]})$. // If there are several ways to do so, it must be consistent accross $\sigma \in \{0, 1\}$.

[a] Implicitly, the H_j are supposed to be 'suitably chosen' for xLPN to be presumed hard, e.g. that they were randomly and independently sampled.

Figure 4.7: Pseudorandom correlation generator $\mathsf{PCG_{stp}}$ for generating pseudorandom instances of the subsets tensor powers correlation over a field $\mathbb{F}$.

Theorem 4. *Let $w > 0$, and $(S_i)_{i \in [\mathsf{n_s}]}$ a list of $\mathsf{n_s}$ subsets of $[w]$. Let B, θ' such that there exists a B-good cover of $(S_i)_{i \in [\mathsf{n_s}]}$ comprised of size-θ' vectors, and let $\theta < \theta'$. Assume that the $\mathbb{F}$-xLPN$(\theta, \theta', \lambda)$ assumption holds, and that MPFSS is a secure multi-point function secret-sharing scheme for the family of $(1+\mu \cdot \lambda)^{\mathsf{tpp}}$-point functions from $[(1 + \mu \cdot \theta')^{\mathsf{tpp}}]$ to $\mathbb{F}$ for all $\mu \in [B]$. Then $\mathsf{PCG_{stp}}$ is a secure pseudorandom correlation generator, which generates pseudorandom shares of a subsets tensor powers correlation $(\vec{r}, ((1_{\mathbb{F}} \,||\, \vec{r}[S_i])^{\otimes \mathsf{tpp}})_{1 \leq i \leq \mathsf{n_s}})$ where $\vec{r} \in \mathbb{F}^w$.*

- Communication: *If the MPFSS seeds have size $O[\lambda \cdot (1+B\lambda)^{\mathsf{tpp}} \cdot \log((1+B\theta')^{\mathsf{tpp}})]$ and MPFSS.FullEval can be computed with $O((1+B\lambda)^{\mathsf{tpp}} \cdot (1+B\theta')^{\mathsf{tpp}} \cdot \frac{\log|\mathbb{F}|}{\lambda})$ invocations of a pseudorandom generator $\mathsf{PRG} : \{0,1\}^\lambda \mapsto \{0,1\}^{2\lambda+2}$, then $\mathsf{PCG_{stp}.Gen}$ outputs seeds of size:*
  $$|\mathsf{k}_\sigma| = O\left(\mathsf{n_s} \cdot \lambda \cdot (1 + B\lambda)^{\mathsf{tpp}} \cdot \log\left((1 + B\theta')^{\mathsf{tpp}}\right)\right).$$
- Computation: *The computational complexity of $\mathsf{PCG_{stp}.Expand}$ is predominantly that of $O(\mathsf{n_s} \cdot (1 + B\lambda)^{\mathsf{tpp}} \cdot (1 + B\theta') \cdot \frac{\log|\mathbb{F}|}{\lambda})$ invocations of a PRG, plus $\mathsf{n_s}$ matrix-vector products with a matrix of dimensions $(1 + B\theta)^{\mathsf{tpp}} \times (1 + B\theta')^{\mathsf{tpp}}$ which requires at most $O(\mathsf{n_s} \cdot (B\theta)^{\mathsf{tpp}} \cdot (B\theta')^{\mathsf{tpp}}) \subseteq O(\mathsf{n_s} \cdot (B\theta')^{2 \cdot \mathsf{tpp}})$ arithmetic operations over $\mathbb{F}$.*

Proof. Let us first show correctness of our candidate PCG. By correctness of the MPFSS, $\vec{v}_{i,0} + \vec{v}_{i,1} = (1_{\mathbb{F}}||(\vec{e}_\ell)_{\ell\in I_i})^{\intercal}$ for every $i \in [\mathsf{n_s}]$. Therefore,

$$\begin{aligned}\vec{y}_0 + \vec{y}_1 &= ((H'_i)^{\otimes \mathsf{tpp}} \cdot \vec{v}_0 + (H'_i)^{\otimes \mathsf{tpp}} \cdot \vec{v}_1)_{1\le i\le \mathsf{n_s}} \\ &= ((H'_i)^{\otimes \mathsf{tpp}} \cdot ((1_{\mathbb{F}} \,||\, (\vec{e}_\ell)_{\ell\in I_i})^{\intercal})_{1\le i\le \mathsf{n_s}} = (\tilde{\vec{r}}_i^{\otimes \mathsf{tpp}})_{1\le i\le \mathsf{n_s}}.\end{aligned}$$

$\vec{r}[S_i] = \left(\sum_{j=1}^{m} f_{\alpha_j,\vec{r}_j}\right)[S_i] = \left(\sum_{j\in I_i} f_{\alpha_j,\vec{r}_j}\right)[S_i]$, where $f_{\alpha_j,\vec{r}_j}$ is the sparse vector obtained by spreading the vector $\vec{r}_j$ over the coordinates in the ordered set α_j. It follows that $(1_{\mathbb{F}} \,||\, \vec{r}[S_i])^{\otimes \mathsf{tpp}}$ can be extracted from $\tilde{\vec{r}}_j$ (since any degree-$(\le \mathsf{tpp})$ polynomial in $\vec{a}+\vec{b}$ is also a degree-$(\le \mathsf{tpp})$ polynomial in $\vec{a}||\vec{b}$).
Security follows exactly by the same sequence of hybrids as in the previous analysis. We show the following indistinguishability:

$$\begin{aligned}&\{(\mathsf{k}_\sigma, \vec{r}_{1-\sigma}) \;:\; (\mathsf{k}_0,\mathsf{k}_1) \xleftarrow{\$} \mathsf{Gen}(1^\lambda), \vec{r}_{1-\sigma} \leftarrow \mathsf{Expand}(1-\sigma, \mathsf{k}_{1-\sigma})\} \\ \overset{\text{c}}{\approx}&\{(\mathsf{k}_\sigma, \vec{r}_{1-\sigma}) \;:\; (\mathsf{k}_0,\mathsf{k}_1) \xleftarrow{\$} \mathsf{Gen}(1^\lambda), \vec{r}_\sigma \leftarrow \mathsf{Expand}(\sigma, \mathsf{k}_\sigma), \\ &\qquad\qquad \vec{r}' \xleftarrow{\$} \mathbb{F}^m, \vec{r}_{1-\sigma} \leftarrow ((1_{\mathbb{F}}||\vec{r}')^{\otimes \mathsf{tpp}})_{i\le \mathsf{n_s}} - \vec{r}_\sigma\}.\end{aligned}$$

We first switch all the $K^{\mathsf{fss}}_{i,\sigma}$ to simulated keys using the security of the MPFSS, then replace the $H_i \cdot \vec{e}_i^{\intercal}$ by random vectors, applying m times the security of LPN, once for each replacement. The sum of sparse random vectors which form a good cover is itself a random vector (by item 1 in definition 14), therefore the resulting distribution is exactly the second distribution above, hence security follows.
Finally, the efficiency claims can be read directly from the construction, and follows from the fact that Gen consists of $\mathsf{n_s}$ MPFSS seeds, and the cost of Expand is dominated by $\mathsf{n_s}$ calls to $\mathsf{MPFSS.FullEval}$ and $\mathsf{n_s}$ matrix-vector products. □

4.2.4 Instantiating the MPFSS

Theorem 4 assumes the existence of an MPFSS scheme MPFSS for the family of all $(1+\mu\cdot\lambda)^{\mathsf{tpp}}$-point functions from $[(1+\mu\cdot\theta')^{\mathsf{tpp}}]$ to $\mathbb{F}$ for some $\mu \in [B]$ (or, equivalently, an MPFSS for each μ which can then all be combined into one scheme), with the following efficiency guarantees: $\mathsf{MPFSS.Gen}(1^\lambda)$ outputs seeds of size $O((1+B\lambda)^{\mathsf{tpp}} \cdot \lambda \cdot \log((1+B\theta')^{\mathsf{tpp}}))$, and $\mathsf{MPFSS.FullEval}$ can be computed with $O((1+B\lambda)^{\mathsf{tpp}} \cdot (1+B\theta')^{\mathsf{tpp}} \cdot \frac{\log|\mathbb{F}|}{\lambda})$ invocations of a pseudorandom generator $\mathsf{PRG} : \{0,1\}^\lambda \mapsto \{0,1\}^{2\lambda+2}$. The works of [BGI16b, BCGI18] provides exactly such a construction, which makes a black box use of any pseudorandom generator $\mathsf{PRG} : \{0,1\}^\lambda \mapsto \{0,1\}^{2\lambda+2}$.
We instantiate the PRG using the LPN-based construction of [BKW03], which we now recall. Fix some (constant) noise rate ε, a random matrix $A \xleftarrow{\$} \mathbb{F}_2^{n_1\times n_2}$. Given a random bitstring $r \in \{0,1\}^{n_2+n_1\cdot h(\varepsilon)}$, where h is the binary entropy function, define $\vec{s} = \vec{s}(r) \in \mathbb{F}_2^{n_2}$ to be the n_2 first bits of r, and use the remaining $n_1 \cdot h(\varepsilon)$ bits to sample a random vector $\vec{e}(r)$ from $\mathsf{Ber}_\varepsilon(\mathbb{F}_2)^{n_1}$ (it is well known that this distribution can be sampled using roughly $n_1\cdot h(\varepsilon)$ bits of randomness). Define the pseudorandom generator $\mathsf{PRG} : \{0,1\}^\lambda \mapsto \{0,1\}^{2\lambda+2}$ as $\mathsf{PRG}(r) = A\cdot\vec{s}(r) + \vec{e}(r) \in \mathbb{F}_2^{n_1}$. Security follows from the $\mathbb{F}_2$-$\mathsf{LPN}(n_2, n_1, \varepsilon)$ assumption, and this PRG stretches $\lambda = n_2 + n_1 \cdot h(\varepsilon)$ bits to $n_1 = 2\lambda+2$ bits when $n_2 = n_1\cdot(1/2 - h(\varepsilon)) - 1$. Hence, given the security parameter λ, security follows from the $\mathbb{F}_2$-$\mathsf{LPN}(\lambda\cdot(1-2h(\varepsilon)) - 2h(\varepsilon), 2\lambda+2, \varepsilon)$ assumption for any constant ε; for example, setting $\varepsilon = 1/8$, this assumption is implied by the $\mathbb{F}_2$-$\mathsf{LPN}(\lambda/4, 3\lambda, 1/8)$ assumption. The cost of evaluating PRG is dominated by the matrix-vector product $A\cdot\vec{s}$, which requires at most $n_1 n_2 = O(\lambda^2)$ arithmetic operations.

4.2.5 Securely Distributing MPFSS.Gen and Π_{stp}

The seeds of the MPFSS scheme of [BCGI18] can be securely generated by using parallel instances of a generic secure computation protocols to securely evaluate the above PRG. Using GMW to instantiate the generic protocol, we have:

Corollary 1. *There exists a semi-honest secure two-party protocol Π_{MPFSS} which distributes the seeds of a multi-point function secret-sharing scheme MPFSS for the family of t'-point functions from $[(1+B\theta')^{\mathsf{tpp}}]$ to $\mathbb{F}$, using $O(t' \cdot \nu \cdot \lambda^2)$ calls to an ideal oblivious transfer functionality, where $\nu = \log((1+B\theta')^{\mathsf{tpp}})$ and $t' = (1+B\lambda')^{\mathsf{tpp}}$, with an additional communication of $O(t' \cdot \nu \cdot \lambda^2)$ bits, and total computation polynomial in $t' \cdot \nu \cdot \lambda$.*

Proof. Let $t' \leftarrow (1+B\lambda)^{\mathsf{tpp}}$. The MPFSS scheme MPFSS for the family of t'-point functions from $[(1+B\theta')^{\mathsf{tpp}}]$ to $\mathbb{F}$ of [BCGI18] is constructed using t' independent instances of a single-point function secret sharing scheme (also called *distributed point function* [GI14]): any t'-point function $f : [(1+B\theta')^{\mathsf{tpp}}] \mapsto \mathbb{F}$ can be written as the sum $\sum_{i=1}^{t'} f_i$ of point functions $f_i : [(1+B\theta')^{\mathsf{tpp}}] \mapsto \mathbb{F}$ which evaluate to $0_{\mathbb{F}}$ everwhere, except on a single entry j_i where they take the value $f(j_i)$ (the j_i being the entries on which f does not evaluate to $0_{\mathbb{F}}$). Given a distributed point function $(\mathsf{Gen}, \mathsf{Eval})$, the construction and its analysis are straightforward:

- $\mathsf{MPFSS.Gen}$: On input $(1^\lambda, f)$, decompose f as $\sum_{i=1}^{t'} f_i$ as above, and output $(\mathsf{Gen}(1^\lambda, f_i))_{1 \le i \le t'}$.
- $\mathsf{MPFSS.Eval}$: On input $(\sigma, (K_{i,\sigma})_{i \in [t']}, x)$, output $y_\sigma \leftarrow \sum_{i=1}^{t'} \mathsf{Eval}(K_{i,\sigma}, f_i, x)$.

Therefore, securely distributing MPFSS seeds reduces to t' invocations of a secure protocol for distributing the seeds of a distributed point function (DPF). The $\mathsf{DPF.Gen}$ construction of [BGI16b] for point functions over the domain $[(1+B\theta')^{\mathsf{tpp}}]$ works as follows: the two output keys are $K_0 = (s_0^{(0)}, cw_1, \ldots, cw_{\nu+1})$ and $K_1 = (s_1^{(0)}, cw_1, \ldots, cw_{\nu+1})$ where $s_0^{(0)}, s_1^{(0)}$ are two random seeds for the PRG and $\nu = \log((1+B\theta')^{\mathsf{tpp}})$. Gen proceeds in $\nu+1$ steps. In the i-th step it expands $s_0^{(i-1)}$ and $s_1^{(i-1)}$ by using one PRG invocation for each seed and obtains $s_0^{(i)}, s_1^{(i)}$, and cw_i. In the final step the algorithm computes $cw_{\nu+1}$ as a function of the expanded seeds and the target value.

Securely Generating DPF Seeds. We recall that the well known GMW protocol [GMW87b] allows two parties to securely evaluate any circuit of size s (in the semi-honest model) using $O(s)$ calls to an oblivious transfer functionality, and $O(s)$ additional bits of communication. Using GMW for distributing the Gen procedure of the DPF over a field $\mathbb{F}$, the communication and computation of the protocol are dominated by two factors: $O(\lambda)$ oblivious transfers for a seed and location of the designated point and by $O(\nu)$ secure evaluations of the PRG. Since evaluating the PRG can be done using $O(\lambda^2)$ arithmetic operations over $\mathbb{F}_2$, it can be generically computed using $O(\lambda^2)$ calls to an oblivious transfer functionality, and $O(\lambda^2)$ additional bits of communication. Hence the following lemma:

Lemma 3. *There exists a semi-honest secure two-party protocol Π_{DPF} which distributes the seeds of a distributed point function DPF for the family of point functions from $[(1+B\theta')^{\mathsf{tpp}}]$ to $\mathbb{F}$, using $O(\nu \cdot \lambda^2)$ calls to an ideal oblivious transfer functionality, where $\nu = \log((1+B\theta')^{\mathsf{tpp}})$, with an additional communication of $O(\nu \cdot \lambda^2)$ bits, and total computation polynomial in $\nu \cdot \lambda$.*

□

As a direct corollary of Corollary 1, since the seeds of $\mathsf{PCG}_{\mathsf{stp}}$ contain exactly n_{s} independent MPFSS seeds, we have:

Corollary 2. *There exists a semi-honest secure two-party protocol Π_{stp} which distributes the seeds of the pseudorandom correlation generator $\mathsf{PCG}_{\mathsf{stp}}$ represented on Figure 4.7, using $O(\mathsf{n}_{\mathsf{s}} \cdot t' \cdot \nu \cdot \lambda^2)$ calls to an ideal oblivious transfer functionality, where $\nu = \log((B\theta'+1)^{\mathsf{tpp}})$ and $t' = (1+B\lambda)^{\mathsf{tpp}}$, with an additional communication of $O(\mathsf{n}_{\mathsf{s}} \cdot t' \cdot \nu \cdot \lambda^2)$ bits, and total computation $O(\mathsf{n}_{\mathsf{s}} \cdot \mathsf{poly}(t' \cdot \nu \cdot \lambda))$.*

Instantiating the oblivious transfer. To execute the GMW protocol, we need an oblivious transfer. Under the $\mathbb{F}_2$-$\mathsf{LPN}(\lambda, O(\lambda), 1/\lambda^\delta)$ assumption (δ is any small constant), there exists oblivious transfers (with simulation security) with $\mathsf{poly}(\lambda)$ communication and computation; see for example [DGH+20].

Constructing Π_{corr}. The work of [BCG+19b] shows that any corruptible functionality distributing the output of a correlation generator $\mathcal{C}$ can be secure instantiated using any semi-honest secure two-party protocol Π for distributing the Gen procedure of a PCG for $\mathcal{C}$, with the same communication as Π, and with computational complexity dominated by the computational complexity of Π plus the computational complexity for computing the $\mathsf{PCG.Expand}$ procedure. Therefore, using their result together with our protocol Π_{stp} for generating the seeds of a PCG for subsets tensor powers correlation allows to securely instantiate $\mathcal{F}_{\mathsf{corr}}$ (with $N = 2$).
Recall that the computation of $\mathsf{PCG}_{\mathsf{stp}}.\mathsf{Expand}$ is dominated by $O(\mathsf{n}_{\mathsf{s}} \cdot (1+B\lambda)^{\mathsf{tpp}} \cdot (1+B\theta')^{\mathsf{tpp}} \cdot \frac{\log|\mathbb{F}|}{\lambda})$ invocations of a PRG – which requires at most $O(\lambda^2 \cdot \mathsf{n}_{\mathsf{s}} \cdot (1+B\lambda)^{\mathsf{tpp}} \cdot (1+B\theta')^{\mathsf{tpp}} \cdot \frac{\log|\mathbb{F}|}{\lambda})$ operations over $\mathbb{F}_2$ using the simple LPN-based PRG from [BKW03] –, plus an additional $O(\mathsf{n}_{\mathsf{s}} \cdot (1+B\theta)^{\mathsf{tpp}} \cdot (1+B\theta')^{\mathsf{tpp}})$ arithmetic operations over $\mathbb{F}$. Since each operation over $\mathbb{F}$ can be computed with $O(\log|\mathbb{F}|)^2)$ boolean operations, combining the two, we get computation $O(\lambda \cdot \mathsf{n}_{\mathsf{s}} \cdot (1+B\theta)^{\mathsf{tpp}} \cdot (1+B\theta')^{\mathsf{tpp}} \cdot (\log|\mathbb{F}|)^2)$.
All that remains is for the parties to generate the necessary material for $\mathsf{PCG}_{\mathsf{stp}}$: m random $\mathbb{F}^{\theta\times\theta'}$ matrices and m size-θ' subsets of $[w]$. At its core, this is just a matter for the parties to generate and hold the same $m \cdot (\theta \cdot \theta' \cdot \log|\mathbb{F}| + \log\binom{w}{\theta'})$ (pseudo)-random bits. This can be achieved by having one party sample a seed of size λ, send it to the other, and both parties can expand it locally by calling the length-doubling PRG from [BKW03] (and used above) $m \cdot \theta' \cdot (\theta \cdot \log|\mathbb{F}| + \log w)/\lambda$ times (in a GGM tree-like approach). This requires λ bits of communication and $O(m \cdot \theta' \cdot (\theta \cdot \log|\mathbb{F}| + \log w) \cdot \lambda)$ bits of local computation. This is summarised in an intermediate theorem, Theorem 5 below.

Theorem 5. *Assume the $\mathbb{F}$-$\mathsf{xLPN}(\theta, \theta'-\theta, \lambda/\theta')$ and $\mathbb{F}_2$-$\mathsf{LPN}(\lambda, O(\lambda), 1/\lambda^\delta)$ – where δ is any small enough constant – assumptions hold. Then there exists a semi-honest secure two-party protocol Π_{stp} which securely generates a subsets tensor powers correlation for subsets $(S_i)_{i\in[\mathsf{n}_{\mathsf{s}}]}$ of $[w]$ and for which there exists a B-good cover comprised of m size-θ' masks, using the following resources:*

- Communication:

$$O\Big(\mathsf{n}_{\mathsf{s}} \cdot \mathsf{poly}(\lambda) \cdot (1+B\lambda)^{\mathsf{tpp}} \cdot \log(1+B\theta')^{\mathsf{tpp}}\Big).$$

- Computation:

$$O\Big(\lambda \cdot \mathsf{n_s} \cdot (1+B\theta)^{\mathsf{tpp}} \cdot (1+B\theta')^{\mathsf{tpp}} \cdot (\log|\mathbb{F}|)^2 + \mathsf{n_s} \cdot \mathsf{poly}\big((1+B\lambda)^{\mathsf{tpp}} \cdot \log(1+B\theta')^{\mathsf{tpp}} \cdot \lambda\big) + m \cdot \theta' \cdot (\theta \cdot \log|\mathbb{F}| + \log w) \cdot \lambda\Big).$$

Wrapping up, using Π_{stp} with an appropriate good cover suffices to construct a protocol Π_{corr} for securely implementing the functionality $\mathcal{F}_{\mathsf{corr}}$. For each $i = 1 \dots \lceil d/k \rceil - 1$, the parties need to generate a B-good cover of the $((U^{\mathsf{in}}_{i,j})_{j\in[\lceil s^{i+1}/\beta\rceil]}, (U_{i,j})_{j\in[\lceil s^{i+1}/\beta\rceil]}, (V^{\mathsf{in}}_{i,j})_{j\in[\lceil m^i/\beta\rceil]}, (V_{i,j})_{j\in[\lceil m^i/\beta\rceil]})$ seen as subsets of $[n+s_i]$. A way to do so is to generate:

- a $B/2$-good cover A_{in} of $((U^{\mathsf{in}}_{i,j})_{j\in[\lceil s^{i+1}/\beta\rceil]}, (V^{\mathsf{in}}_{i,j})_{j\in[\lceil m^i/\beta\rceil]})_{1\le i<\lceil d/k\rceil}$ seen as subsets of $[n]$ comprised of M_{in} size-θ' masks;
- for each $i = 1 \dots \lceil d/k\rceil - 1$, a $B/2$-good cover A_i of $((U_{i,j})_{j\in[\lceil s^{i+1}/\beta\rceil]}, (V_{i,j})_{j\in[\lceil m^i/\beta\rceil]})$ seen as subsets of $[s_i]$ comprised of M_i size-θ' masks.

Let $\kappa, \kappa', \kappa_{\mathsf{in}}$ be correctness parameters. We set $M_{\mathsf{in}} \leftarrow \kappa_{\mathsf{in}} \cdot n \cdot \log n$, $M_i \leftarrow \kappa \cdot s_i \cdot \log s_i$ (which is upper-bounded by $M := \kappa \cdot s \cdot \log s$), and $B \leftarrow 2\kappa' \cdot \kappa \cdot \ln s$. The probability $p = p(\kappa_{\mathsf{in}}, \kappa, \kappa')$ all the above conditions are satisfied is then, by union-bound (and with $s_i/\beta \le s/\beta$), at least:

$$1 - \left(\frac{1}{n^{\kappa_{\mathsf{in}}-1}} + \frac{(s/k)/\beta}{n^{(\kappa'-2)\cdot\kappa_{\mathsf{in}}\cdot\theta/2}}\right) - \lceil d/k\rceil \cdot \left(\frac{1}{(s/k)^{\kappa-1}} + \frac{s/\beta}{(s/k)^{(\kappa'-2)\cdot\kappa\cdot\theta/2}}\right).$$

Wrapping-up, we get the following parametrised theorem.

Theorem 6. *Assume the* $\mathbb{F}$-$\mathsf{xLPN}(\theta, \theta'-\theta, \lambda/\theta')$ *and* $\mathbb{F}_2$-$\mathsf{LPN}(\lambda, O(\lambda), 1/\lambda^\delta)$ *– where* δ *is any small enough constant – assumptions hold. Then there exists a probabilistic semi-honest secure two-party protocol* Π_{corr} *which securely implements the functionality* $\mathcal{F}_{\mathsf{corr}}$ *given on Figure 4.3 with success probability:*

$$p^* = 1 - \left(\frac{1}{n^{\kappa_{in}-1}} + \frac{(s/k)/\beta}{n^{(\kappa'-2)\cdot\kappa_{in}\cdot\theta/2}}\right) - \lceil d/k\rceil \cdot \left(\frac{1}{(s/k)^{\kappa-1}} + \frac{s/\beta}{(s/k)^{(\kappa'-2)\cdot\kappa\cdot\theta/2}}\right).$$

Furthermore, it uses the following resources, where $B = 2\kappa\kappa' \cdot \ln s$*:*

- Communication:

$$O\Big(\sum_{i=1}^{\lceil d/k\rceil-1} \frac{s_{i+1}+m_i}{\beta} \cdot \mathsf{poly}(\lambda) \cdot (1+B\lambda)^{2^k} \cdot \log(1+B\theta')^{2^k}\Big).$$

- Computation:

$$O\Bigg(\sum_{i=1}^{\lceil d/k\rceil-1} \frac{s_{i+1}+m_i}{\beta}\Big[\lambda \cdot (1+B\theta)^{\mathsf{tpp}} \cdot (1+B\theta')^{\mathsf{tpp}} \cdot (\log|\mathbb{F}|)^2 + \cdot\mathsf{poly}\big((1+B\lambda)^{\mathsf{tpp}} \cdot \log(1+B\theta')^{\mathsf{tpp}} \cdot \lambda\big)\Big] + (\kappa_{in} \cdot n\log n \cdot \theta' \cdot (\theta \cdot \log|\mathbb{F}| + \log n) \cdot \lambda) + (2\kappa \cdot s\log s \cdot \theta' \cdot (\theta \cdot \log|\mathbb{F}| + \log s) \cdot \lambda)\Bigg).$$

Chapter 5

Bridging the Gap between HSS and FHE

This chapter describes results which have been communicated previously in [CMPR23].

Based on joint work with Geoffroy Couteau, Alain Passelègue, and Mahshid Riahinia.

In this chapter, we show how to adapt essentially all existing constructions of homomorphic secret sharing schemes for branching programmes to enjoy some limited "programmability" properties, thereby leading to sublinear-communication secure for layered circuits with one-sided statistically security from DDH or DCR. Because this chapter is mostly independent of the others (whereas in contrast chapter 7 builds heavily on chapters 4 and 6), we will take the liberty of keeping the presentation of our results at a relatively high level.

Extending HSS Properties. We identify two natural extensions of homomorphic secret sharing, which we term respectively *homomorphic secret sharing with simulatable memory shares* and *staged homomorphic secret sharing*. At a high level, both notions capture the ability to perform some limited form of *programming* of HSS shares, i.e. , to construct one of the two HSS shares of an input x before knowing x. It turns out that most of known HSS constructions already achieve these extensions, leading to constructions based on a wide variety of assumptions.

One-sided statistically secure computation with sublinear communication. A core feature of FHE-based sublinear secure computation is that it achieves *one-sided statistical security* when using an FHE scheme with statistical circuit-privacy, since homomorphic evaluation of $f(\cdot, y)$ leaks *statistically* no information about y beyond $f(x, y)$. In other words, Bob's security in the aforementioned protocol holds unconditionally. One-sided statistical security is a desirable security notion and can be achieved quite easily if we do not require sublinear communication, e.g., by using the seminal GMW protocol [GMW87a] with a one-sided statistically secure oblivious transfer [NP01] (to our knowledge, this was first observed in [Cha90]). Yet, as of today, one-sided statistically secure computation with sublinear communication is *only known from FHE*: all HSS-based constructions inherently achieve only computational security for both parties.
Using staged HSS, we obtain the first non-FHE-based constructions of one-sided statistically secure protocols with sublinear communication. Concretely, we obtain secure

computation for any log log-depth circuits with optimal communication, where x remains statistically hidden, provided that $|x| < |y|/\mathsf{poly}(\lambda)$ (where $\mathsf{poly}(\lambda)$ denotes some fixed polynomial), via a black-box use of staged HSS. We also get secure computation of any layered arithmetic circuit C of size s over a sufficiently large ring $\mathbb{Z}_n$, with sublinear communication $O(s/\log\log s)$ and one-sided statistical security (without any restriction on the statistically protected input size), assuming the Paillier encryption scheme is circular-secure. The latter construction is non-black box and exploits the specific structure of a concrete Paillier-based staged HSS scheme from [OSY21].

5.1 An Overview of this Chapter's Results

5.1.1 An overview of staged HSS

We start by providing a high-level description of HSS schemes, which applies to essentially all known HSS constructions (beside FHE-based constructions).
HSS schemes rely on an additively homomorphic encryption scheme with some form of linear decryption. The public key of the HSS scheme is the public key pk of the underlying encryption scheme, and evaluation keys $\mathsf{ek}_0, \mathsf{ek}_1$ are additive shares of the underlying secret key s. A scheme uses two types of data: (1) **Input shares** $(\mathsf{I}_0, \mathsf{I}_1)$ which are generated by running $\mathsf{Input}(\mathsf{pk}, x)$ on some input x and consist in an encryption of $(x, x \cdot s)$, and (2) **Memory shares** (M_0, M_1) which are typically additive shares of $(x, x \cdot s)$ over $\mathbb{Z}$. Two types of operations are handled: **Additions of memory shares** (simply add the shares as $(x, x \cdot s) + (y, y \cdot s) = (x + y, (x + y) \cdot s)$), and a restricted form of **Multiplication**. Specifically, multiplication can only be performed between an *input share* of some value x and a *memory share* of some value y, and returns a *memory share* of their product $x \cdot y$. Typically, multiplication uses the memory share $(y, y \cdot s)$ to "linearly multiply-and-decrypt" the encryption of $(x, x \cdot s)$, getting some encoding of $(xy, xy \cdot s)$. Then, the encoding is converted into a valid memory share using a specific procedure, which depends on the concrete scheme and is often a form of *distributed discrete logarithm*. We provide more details about multiplication later. Note that one can transform any input share into a memory share of the same value by multiplying it with a memory share of 1. At the end of a computation, each party recovers a memory value consisting in an additive share of $(z, z \cdot s)$, and therefore a share of the result z by dropping the second part. One can evaluate any polynomial-size program following the above restrictions, which precisely corresponds to *restricted multiplication straight-line* (RMS) programs, and encompasses branching programs, NC^1, and more.

Our starting point is the result of two observations. First, we observe that any HSS following the above structure does in fact allow for a limited form of programming regarding memory values. Indeed, while input shares include a homomorphic encryption of the input (which cannot be generated without knowing the input), *memory shares* are simply additive shares. Thus, we can always *simulate* a memory share of one party before knowing the value to share, by generating a first random share u. The other share is later set to $x - u$ when the actual value x to share is known.
Second, we remark that two parties sharing input shares of some values $(x_1, \ldots, x_n)$ as well as memory shares of a value z can compute memory shares of $z \cdot P(x_1, \ldots, x_n)$ for any RMS program P. The trick is to evaluate all the operations of P "with z in front", i.e. by maintaining as an invariant that any memory share for any value y that should be used in the computation is replaced by a memory share for the value $z \cdot y$. This

invariant being preserved by the two RMS operations (addition and multiplication), it is sufficient to guarantee that every memory value satisfies it when created. This is simply done by transforming an input x into a memory value by multiplying it with the memory share of z in order to get a memory share for $z \cdot x$ rather than for x.

While the above already offers enough flexibility to evaluate linear functions (and extensions thereof, such as low-degree polynomials), we still cannot handle general computations like NC^1 circuits. To overcome this limitation, we show by a deeper analysis of known HSS schemes that most of them also achieve some specific, limited form of programmability, which turns out to be sufficient to construct CPRFs for all RMS programs (hence in particular for NC^1).
Concretely, for a vector $\vec{u} = (u_1, \ldots, u_\ell)$, our core observation is that it is possible to share $\vec{u}$ between parties P_0 and P_1 with two alternate sharing algorithms $(\overline{\mathsf{Input}}_0, \overline{\mathsf{Input}}_1)$ such that: (1) P_0's share of $\vec{u}$, obtained from $\overline{\mathsf{Input}}_0$, is *independent* of $\vec{u}$ (and can be generated without $\vec{u}$), (2) P_0 and P_1 can use specific $\overline{\mathsf{Eval}}_0, \overline{\mathsf{Eval}}_1$ evaluation algorithms to produce memory shares of $P(\vec{u})$ for any RMS program P, *provided that P_1 knows $\vec{u}$ in the clear*. We call staged-HSS an HSS scheme satisfying the latter properties, as it intuitively allows to split share generation and evaluation in 2 stages: a first *input-independent* stage, corresponding to P_0's view, and a second *input-dependent* stage corresponding to P_1's view.
At first sight, staged-HSS might not seem particularly useful: if P_1 knows $\vec{u}$ in the clear, then P_1 can already compute $P(\vec{u})$ for any RMS program P. The key observation is that P_0 and P_1 get *memory shares* of $P(\vec{u})$, and not just $P(\vec{u})$. This memory share can then be combined with the prior observations to let P_0, P_1 compute additive shares of $P(\vec{u}) \cdot Q(\vec{v})$, for any other RMS program P, Q, given *input shares* of $\vec{v}$. Setting $\vec{u}$ to be the description of the constraint C, P to be a universal circuit (with input x hardwired) which on input C returns $C(x)$, $\vec{v}$ to be a PRF key k, and Q to be the RMS program (with x hardwired) which on input k returns $F_k(x)$, parties P_0 and P_1 can then compute shares of $C(x) \cdot F_k(x)$, with shares of P_0 being independent of C. We can then instantiate our simple aforementioned strategy for constructing CPRFs while circumventing the need for C during KeyGen. As a result, we obtain (1-key selective) CPRFs for RMS programs (and therefore for NC^1) from any staged-HSS, i.e. from a wide variety of assumptions (including DCR [OSY21, RS21], class groups assumptions, or variants of QR [ADOS22, CLT22], and more.). The security analysis is similar to our construction for inner-product, though this new construction is no longer constraint-hiding, since the CEval algorithm now relies on knowing C (i.e. $\vec{u}$ above) in clear.

It remains to explain why known HSS schemes are also staged-HSS schemes. To illustrate this, we use the simple ElGamal-based HSS scheme from [BGI16a][1]. We assume basic knowledge of ElGamal encryption in what follows. This scheme follows the general structure detailed above by instantiating the additively homomorphic encryption scheme with ElGamal encryption. That is, an *input share* for x is an ElGamal encryption of the pair $(x, x \cdot s)$[2], i.e. a tuple $(c_0, c_0', c_1, c_1') = (g^{r_0}, h^{r_0} \cdot g^x, g^{r_1}, h^{r_1} \cdot g^{x \cdot s})$ with $s \in \mathbb{Z}_p$ being the secret key, $h = g^s$ being the public key, and $r_0, r_1 \xleftarrow{\$} \mathbb{Z}_p$ encryption randomness[3].

[1] This scheme does not yield CPRFs as it does not achieve statistical correctness, but staged-HSS is easily illustrated with it.

[2] Actually of x and $x \cdot s_i$'s for each bit s_i of s.

[3] s is encrypted bit-by-bit in the actual construction.

Multiplication between an input share (c_0, c'_0, c_1, c'_1) of x and a memory share $(\alpha_\sigma, \beta_\sigma)$ of y (which is just an additive share of $(y, y \cdot s)$ over $\mathbb{Z}_p$ owned by party P_σ) is done as follows. First, party P_σ computes $g_\sigma \leftarrow (c'_0)^{\alpha_\sigma}/c_0^{\beta_\sigma}$. Observe that $g_0 \cdot g_1 = (c'_0)^{\alpha_0+\alpha_1}/c_0^{\beta_0+\beta_1} = (g^{sr} \cdot g^x)^y/(g^r)^{sy} = g^{xy}$. Hence, parties get *multiplicative shares* g_0, g_1 of g^{xy}. Doing the same with c_1, c'_1 allows to get multiplicative shares of $g^{xy \cdot s}$. Then, an operation termed *distributed discrete logarithm* allows to transform these multiplicative shares of $(g^{xy}, g^{xy \cdot s})$ into additive shares of $(xy, xy \cdot s)$, i.e. memory shares for the value xy, as desired. Despite being at the core of HSS constructions, the details of the distributed discrete logarithm procedure do not matter here. The only important observation is that the $c_i = g^{r_i}$ components of input shares are independent of the input x; only the c'_i components actually depend on x. Furthermore, in the multiplication above, the only place where c'_i is involved is in the computation of $g_\sigma \leftarrow (c'_i)^{\alpha_\sigma}/c_i^{\beta_\sigma}$. Now, assume that one of the parties, say, P_1, already knows y in the clear: in this case, one can simply define $\alpha_1 \leftarrow y$ and $\alpha_0 \leftarrow 0$, which form valid additive shares of y. But now, P_0 does no longer need to know c'_i components either, since we now have $g_0 = 1/(c_i)^{\beta_0}$.

5.1.2 An overview of sublinear-communication from staged HSS

Staged HSS allows Alice and Bob, respectively owning private inputs x and y, to securely retrieve, given shares of their joint input (x, y), additive shares of $f(x) \cdot g(y)$ for any RMS programs f, g—and even of any $P(x, y) = \sum_{i=1}^{m} f_i(x) \cdot g_i(y)$, where the (f_i, g_i) are RMS programs since additive shares can be added locally—while statistically protecting one of the two inputs (e.g., x). Moreover, the class of such functions $F(x, y)$ contains in particular all arithmetic circuits (with fan-in 2) of size s and depth $\log \log s$, as in such circuits, every output bit depends on at most $\log s$ inputs, and can therefore be written as a multivariate polynomial in the inputs, with at most s monomials. As a consequence, if there is a secure computation protocol for generating staged HSS shares of inputs x and y with communication $c(|x|, |y|)$, then there exists a protocol for securely computing all circuits of size s and depth $\log \log s$ with $|x| + |y|$ inputs and m outputs with communication $c(|x|, |y|) + 2m$, which is asymptotically optimal. It only remains to find a protocol to securely distribute staged HSS shares with linear communication.

This is not easily done in general, as the standard technique to generate HSS shares with low communication uses *hybrid encryption*: to share an input x, one generate HSS shares of some seed seed (using a generic secure computation protocol), and publishes $x \oplus \mathsf{PRG}(\mathsf{seed})$. Then, the homomorphic evaluation first computes $\mathsf{PRG}(\mathsf{seed})$, unmaskes x, and then applies the function. The issue is that this is inherently incompatible with having (one-sided) statistical security. We describe two cases where we can get around this issue:

1. The first way is to use hybrid encryption only on y, for which we just aim to computational security, and to share x using the standard staged HSS sharing algorithm. This yields a one-sided statistically secure protocol for all $\log \log$-depth circuits with communication $|y| + |x| \cdot \mathsf{poly}(\lambda) + O(m)$, which is optimal as soon as $|x| < |y|/\mathsf{poly}(\lambda)$. In other terms, if the input to be statistically protected is polynomially smaller than the other input, we achieve optimal communication.

2. Our second solution relies on a specific construction of staged HSS scheme that relies on the circular security of the Paillier-ElGamal encryption scheme. Here,

we manage to leverage the inherent compactness of this specific scheme to get a protocol with optimal communication $|y| + |x| + O(m)$ for arithmetic circuits over a sufficiently large ring (since Paillier encryption is compact only when the values are from a large ring), by designing a tailored low-communication HSS share distribution protocol. By breaking the circuit into log log-depth blocks, this generalizes naturally to a one-sided statistically secure protocol with *sub-linear* communication $O(s/\log\log s)$ for any layered arithmetic circuits[4] over a sufficiently large field.

5.2 Staged HSS

The core notion underlying our constructions is homomorphic secret sharing (HSS), introduced by Boyle et al. in [BGI16a]. In this section, we remind the standard definition of HSS as well as propose several extensions, in particular defining some special properties that play an important role in our constructions. We further remark that these extensions are easily instantiated using the DCR-based HSS construction from [OSY21].

5.2.1 Homomorphic Secret Sharing

We start by recalling the standard definition of homomorphic secret sharing, as well as of Restricted Multiplication Straight-line (RMS) programs which is the common model of computation in the context of HSS.

Definition 15 (Homomorphic Secret Sharing)**.** *Denote by λ a security parameter. A* Homomorphic Secret Sharing *(HSS) scheme for a class of programs $\mathcal{P}$ which is defined over a ring $\mathcal{R}$ and has input space $\mathcal{I} \subseteq \mathcal{R}$ consists of three PPT algorithms* $(\mathsf{Setup}, \mathsf{Input}, \mathsf{Eval})$ *such that:*

- $\mathsf{Setup}(1^\lambda) \to (\mathsf{pk}, (\mathsf{ek}_0, \mathsf{ek}_1))$*: On input the security parameter λ, the setup algorithm outputs a public key* pk *and a pair of evaluation keys* $(\mathsf{ek}_0, \mathsf{ek}_1)$*.*
- $\mathsf{Input}(\mathsf{pk}, x) \to (\mathsf{I}_0, \mathsf{I}_1)$*: On input the public key* pk *and an input $x \in \mathcal{I}$, the input algorithm outputs a pair of input information* $(\mathsf{I}_0, \mathsf{I}_1)$*.*
- $\mathsf{Eval}(\sigma, \mathsf{ek}_\sigma, \mathsf{I}_\sigma = (\mathsf{I}_\sigma^{(1)}, \ldots, \mathsf{I}_\sigma^{(\rho)}), P) \to y_\sigma$*: On input a party index $\sigma \in \{0,1\}$, an evaluation key* ek_σ*, a vector of ρ input values* $(\mathsf{I}_\sigma^{(1)}, \ldots, \mathsf{I}_\sigma^{(\rho)})$*, and a program $P \in \mathcal{P}$, the evaluation algorithm outputs the party σ's corresponding share of the output y_σ.*

We require an HSS scheme to satisfy the following two properties:

- ***Correctness.*** *For any security parameter $\lambda \in \mathbb{N}$, and any program $P \in \mathcal{P}$ with input space $\mathcal{I} \subseteq \mathcal{R}$, we have:*

$$\Pr\left[y_0 - y_1 = P(x^{(1)}, \ldots, x^{(\rho)})\right] \geq 1 - \mathsf{negl}(\lambda) \ ,$$

where the probability is taken over $(\mathsf{pk}, (\mathsf{ek}_0, \mathsf{ek}_1)) \leftarrow \mathsf{Setup}(1^\lambda)$*,* $(\mathsf{I}_0^{(i)}, \mathsf{I}_1^{(i)}) \leftarrow \mathsf{Input}(\mathsf{pk}, x^{(i)})$ *for $i \in [\rho]$, and* $y_\sigma \leftarrow \mathsf{Eval}(\sigma, \mathsf{ek}_\sigma, (\mathsf{I}_\sigma^{(1)}, \ldots, \mathsf{I}_\sigma^{(\rho)}), P)$*, for $\sigma \in \{0,1\}$.*

[4] An arithmetic circuit is layered if its nodes can be partitioned into layers, such that any wire connects adjacent layers.

- ***Security.*** *For any PPT adversaries $\mathcal{A}, \mathcal{A}'$, and any bit $\sigma \in \{0,1\}$ the following value should be negligible in λ:*

$$\left| \Pr \left[b' = b : \begin{array}{l} (x_0, x_1, \mathsf{st}) \leftarrow \mathcal{A}(1^\lambda) \\ (\mathsf{pk}, (\mathsf{ek}_0, \mathsf{ek}_1)) \leftarrow \mathsf{Setup}(1^\lambda) \\ b \xleftarrow{\$} \{0,1\} \\ (\mathsf{I}_0, \mathsf{I}_1) \leftarrow \mathsf{Input}(x_b) \\ b' \leftarrow \mathcal{A}'(\mathsf{st}, \mathsf{pk}, \mathsf{ek}_\sigma, \mathsf{I}_\sigma) \end{array} \right] - \frac{1}{2} \right|$$

We now remind the definition of Restricted Multiplication Straight-line (RMS) programs. RMS programs form a class of programs which encompasses branching programs of polynomial-size and therefore NC^1 circuits. In an RMS program, the multiplication is restricted to happen between an input value and an intermediate value of the computation (so-called "memory" value).

Definition 16 (RMS Programs)**.** *An RMS program with magnitude bound B is defined as a sequence of the instructions as follows:*

- $\mathsf{ConvertInput}(\mathsf{I}^x) \to \mathsf{M}^x$*: Loads an input x into memory.*
- $\mathsf{Add}(\mathsf{M}^x, \mathsf{M}^y) \to \mathsf{M}^{x+y}$*: Adds two memory values.*
- $\mathsf{Mul}(\mathsf{I}^x, \mathsf{M}^y) \to \mathsf{M}^{x \cdot y}$*: Multiplies an input value and a memory value to produce a memory value of their product.*
- $\mathsf{Output}(\mathsf{M}^x, n) \to x \bmod n$*: Outputs a memory value* w.r.t. *a modulus $n < B$.*

5.2.2 HSS following the RMS Template

Similarly to [BCG+17], we first propose a more specific definition for HSS with additional algorithms that are relevant in the context of RMS programs.

Definition 17 (HSS Following the RMS Template)**.** *A homomorphic secret sharing scheme $\mathsf{HSS} = (\mathsf{Setup}, \mathsf{Input}, \mathsf{MemGen}, \mathsf{Eval})$ following the RMS template is an HSS scheme as defined in definition 15 with an additional algorithm* MemGen *which serves to produce memory values as follows:*

- $\mathsf{MemGen}(\sigma, \mathsf{ek}_\sigma, x) \to \mathsf{M}_\sigma$*: On input a party index $\sigma \in \{0,1\}$, an evaluation key ek_σ, and an input $x \in \mathcal{I}$, the memory generator algorithm outputs a memory value M_σ.*

Moreover, the Eval *algorithm proceeds with sub-routines following the RMS operations* ConvertInput, Add, Mul, Output *as follows:*

- $\mathsf{Eval}(\sigma, \mathsf{ek}_\sigma, (\mathsf{I}_\sigma^{(1)}, \ldots, \mathsf{I}_\sigma^{(\rho)}), P) \to y_\sigma$*: On input a party index $\sigma \in \{0,1\}$, an evaluation key ek_σ, a vector of ρ input values $(\mathsf{I}_\sigma^{(1)}, \ldots, \mathsf{I}_\sigma^{(\rho)})$, and an RMS program P, this algorithm follows the instructions of P and processes them as follows:*
  - $\mathsf{ConvertInput}(\sigma, \mathsf{ek}_\sigma, \mathsf{I}_\sigma^x) \to \mathsf{M}_\sigma^x$*: This algorithm simply uses the* MemGen *and* Mult *algorithms as follows:*
    - *Run* $\mathsf{MemGen}(\sigma, \mathsf{ek}_\sigma, 1) \to \mathsf{M}_\sigma^1$.
    - *Run* $\mathsf{Mult}(\sigma, \mathsf{ek}_\sigma, \mathsf{I}_\sigma^x, \mathsf{M}_\sigma^1) \to \mathsf{M}_\sigma^x$.
  - $\mathsf{Add}(\sigma, \mathsf{ek}_\sigma, \mathsf{M}^x, \mathsf{M}^y) \to \mathsf{M}^{x+y}$*: This algorithm directly adds the given memory values of x and y. Namely, $\mathsf{M}_\sigma^{x+y} = \mathsf{M}_\sigma^x + \mathsf{M}_\sigma^y$.*

– $\mathsf{Mul}(\sigma, \mathsf{ek}_\sigma, \mathsf{I}^x, \mathsf{M}^y) \to \mathsf{M}^{x \cdot y}$*: It multiplies an input value* I_x *and a memory value* M_y *and outputs a memory value of* $x \cdot y$*. The template does not impose any non-black box requirement on this algorithm.*

– $\mathsf{Output}(\sigma, \mathsf{M}^x, n) \to x \bmod n$*: It uses* M^x *to output* $x_\sigma \bmod n$*.*

Correctness and security properties are defined as in definition 15, and we further require the following property:

Additively Homomorphic Memory. *The memory values generated in* HSS *should be additively homomorphic. Meaning that for any two* $x, y \in \mathcal{I}$ *and any party index* $\sigma \in \{0, 1\}$*, it holds that*

$$\mathsf{M}_\sigma^x + \mathsf{M}_\sigma^y = \mathsf{M}_\sigma^{x+y} \ ,$$

where $\mathsf{M}_\sigma^z \leftarrow \mathsf{MemGen}(\sigma, \mathsf{ek}_\sigma, z)$*, for* $z \in \{x, y\}$*, and* $(\mathsf{pk}, (\mathsf{ek}_0, \mathsf{ek}_1)) \leftarrow \mathsf{Setup}(1^\lambda)$*. Throughout this work, we may refer to memory values satisfying this property as "valid" memory values.*

5.2.3 Extended Evaluation and Simulatable Memory Values

Any HSS following the RMS template as defined above satisfies the following lemma, which states that one can evaluate share of $z \cdot P(x^{(1)}, \ldots, x^{(\rho)})$ using only a memory value of z (instead of an input value) together with the input values of the rest of variables $(x^{(1)}, \ldots, x^{(\rho)})$. This lemma plays a central role in our CPRF constructions.

Lemma 4. *Let* $\mathsf{HSS} = (\mathsf{Setup}, \mathsf{Input}, \mathsf{MemGen}, \mathsf{Eval})$ *be an HSS scheme following the RMS template. There exists an extended evaluation algorithm* ExtEval*:*

- $\mathsf{ExtEval}(\sigma, \mathsf{ek}_\sigma, \mathsf{M}_\sigma, (\mathsf{I}_\sigma^{(1)}, \ldots, \mathsf{I}_\sigma^{(\rho)}), P) \to y_\sigma$*: On input a party index* $\sigma \in \{0, 1\}$*, an evaluation key* ek_σ*, a single memory value* M_σ*, a vector of* ρ *input values* $(\mathsf{I}_\sigma^{(1)}, \ldots, \mathsf{I}_\sigma^{(\rho)})$*, and an RMS program* P*, return a value* y_σ *such that the following holds.*

For any security parameter $\lambda \in \mathbb{N}$ *and any RMS program* P*, we have:*

$$\Pr\left[y_0 - y_1 = z \cdot P(x^{(1)}, \ldots, x^{(\rho)})\right] \geq 1 - \mathsf{negl}(\lambda) \ , \tag{5.1}$$

where the probability is taken of the choice of $(\mathsf{pk}, (\mathsf{ek}_0, \mathsf{ek}_1)) \leftarrow \mathsf{Setup}(1^\lambda)$*,* $(\mathsf{I}_0^{(i)}, \mathsf{I}_1^{(i)}) \leftarrow \mathsf{Input}(\mathsf{pk}, x^{(i)})$*,* $\mathsf{M}_\sigma \leftarrow \mathsf{MemGen}(\sigma, \mathsf{ek}_\sigma, z)$*, and* $y_\sigma \leftarrow \mathsf{ExtEval}(\sigma, \mathsf{ek}_\sigma, \mathsf{M}_\sigma, (\mathsf{I}_\sigma^{(1)}, \ldots, \mathsf{I}_\sigma^{(\rho)}), P)$*, for* $\sigma \in \{0, 1\}, i \in [\rho]$*.*

The proof of the above lemma is detailed in [CMPR23]. It essentially consists in recursively incorporating the memory value M_σ using the standard Eval algorithm by first multiplying inputs with it.

We now introduce an additional property termed *simulatable memory values*. Here, we require that for an input $x \in \mathcal{I}$, the memory value of one of the two parties can be generated ahead of time and without the knowledge of x using a simulation algorithm, while the other memory value can be generated given the pre-computed first memory value and the exact value of x. This simulation should not affect the correctness of ExtEval.

Definition 18 (HSS with Simulatable Memory Values)**.** *Let* $\mathsf{HSS} = (\mathsf{Setup}, \mathsf{Input}, \mathsf{MemGen}, \mathsf{Eval})$ *be an HSS following the RMS template as per definition 17, with input space* $\mathcal{I}$ *over the ring* $\mathcal{R}$*. We say that* HSS *is simulatable with respect to its memory values if there exist algorithms* Sim_0 *and* Sim_1 *such that*

- $\mathsf{Sim}_0(1^\lambda) \to \mathsf{M}_0$*: on input the security parameter* λ *outputs a memory value* M_0*.*
- $\mathsf{Sim}_1(\mathsf{M}_0, z, (\mathsf{ek}_0, \mathsf{ek}_1)) \to \mathsf{M}_1$*: on input a memory value* M_0*, an element* $z \in \mathcal{I}$*, and two encoding keys* $(\mathsf{ek}_0, \mathsf{ek}_1)$ *outputs a memory value* M_1*.*

We also require the two following properties:
Simulation Correctness. *For any* $\lambda \in \mathbb{N}$ *and any* $z \in \mathcal{I}$*, the above correctness condition (eq.* (5.1)*) still holds when the memory value is simulated,* i.e. *when we first sample* $\mathsf{M}_0 \leftarrow \mathsf{Sim}_0(1^\lambda)$ *and then* $\mathsf{M}_1 \leftarrow \mathsf{Sim}_1(\mathsf{M}_0, z, (\mathsf{ek}_0, \mathsf{ek}_1))$*.*
Simulation Security. *It should be computationally hard to distinguish the two memory values obtained via the simulation algorithms. That is, for any* $\lambda \in \mathbb{N}$ *and any* $z \in \mathcal{I}$*, we have* $(z, \mathsf{M}_0) \approx_c (z, \mathsf{M}_1)$ *for any* $(\mathsf{pk}, (\mathsf{ek}_0, \mathsf{ek}_1)) \leftarrow \mathsf{Setup}(1^\lambda)$*,* $\mathsf{M}_0 \leftarrow \mathsf{Sim}_0(1^\lambda)$*, and* $\mathsf{M}_1 \leftarrow \mathsf{Sim}_1(\mathsf{M}, z, (\mathsf{ek}_0, \mathsf{ek}_1))$*.*

5.2.4 Staged Homomorphic Secret Sharing

Finally, we define a new notion termed staged-HSS which is merely extending the idea of HSS with simulatable memory values to the case where we require the possibility of input values to be simulatable as well.

Definition 19 (staged-HSS)**.** *Let* $\mathsf{HSS} = (\mathsf{Setup}, \mathsf{MemGen}, \mathsf{Input}, \mathsf{Eval})$ *be an HSS scheme following the RMS template, with input space* $\mathcal{I}$ *over the ring* $\mathcal{R}$*. We say it is a staged-HSS if there exist additional algorithms* $(\overline{\mathsf{Input}}_0, \overline{\mathsf{Input}}_1)$*, and* $(\overline{\mathsf{Eval}}_0, \overline{\mathsf{Eval}}_1)$ *such that:*

- $\overline{\mathsf{Input}}_0(\mathsf{pk}) \to (\bar{\mathsf{I}}_0, \mathsf{aux})$*: On input a public key* pk*, return a value* $\bar{\mathsf{I}}_0$ *and an auxiliary output* aux*.*
- $\overline{\mathsf{Input}}_1(\mathsf{pk}, x, \mathsf{aux}, (\mathsf{ek}_0, \mathsf{ek}_1)) \to \bar{\mathsf{I}}_1$*: On input a public key* pk*, an input* $x \in \mathcal{I}$*, an auxiliary input* aux*, and two encoding keys* $(\mathsf{ek}_0, \mathsf{ek}_1)$*, return a value* $\bar{\mathsf{I}}_1$*.*
- $\overline{\mathsf{Eval}}_0(\mathsf{ek}_0, (\bar{\mathsf{I}}_0^{(1)}, \ldots, \bar{\mathsf{I}}_0^{(\rho)}), P) \to \mathsf{M}_0$*: On input an evaluation key* ek_0*, a vector of* ρ *input values* $(\bar{\mathsf{I}}_0^{(1)}, \ldots, \bar{\mathsf{I}}_0^{(\rho)})$*, and a program* P*, return a memory value* M_0*.*
- $\overline{\mathsf{Eval}}_1(\mathsf{ek}_1, (\bar{\mathsf{I}}_1^{(1)}, \ldots, \bar{\mathsf{I}}_1^{(\rho)}), (x^{(1)}, \ldots, x^{(\rho)}), P) \to \mathsf{M}_1$*: On input an evaluation key* ek_1*, a vector of* ρ *input values* $(x^{(1)}, \ldots, x^{(\rho)})$ *as well as* $(\bar{\mathsf{I}}_1^{(1)}, \ldots, \bar{\mathsf{I}}_1^{(\rho)})$*, and a program* P*, return a memory value* M_1*.*

We further require the two following properties:
Correctness. *We would like the outputs of* $\overline{\mathsf{Eval}}_0$ *and* $\overline{\mathsf{Eval}}_1$ *to be usable within the extended evaluation algorithm* $\mathsf{ExtEval}$ *(lemma 4). Formally, for any* $\lambda \in \mathbb{N}$ *and any two RMS programs* $P, Q \in \mathcal{P}$*, it should hold that*

$$\Pr[y_0 - y_1 = P(z^{(1)}, \ldots, z^{(\ell)}) \cdot Q(x^{(1)}, \ldots, x^{(\rho)})] \geq 1 - \mathsf{negl}(\lambda) \ ,$$

where

- $(\mathsf{pk}, (\mathsf{ek}_0, \mathsf{ek}_1)) \leftarrow \mathsf{Setup}(1^\lambda)$*,*
- $(\mathsf{I}_0^{x^{(i)}}, \mathsf{I}_1^{x^{(i)}}) \leftarrow \mathsf{Input}(\mathsf{pk}, x^{(i)})$ *for all* $i \in [\rho]$*,*
- $(\bar{\mathsf{I}}_0^{z^{(i)}}, \mathsf{aux}^{(i)}) \leftarrow \overline{\mathsf{Input}}_0(\mathsf{pk})$ *and* $\bar{\mathsf{I}}_1^{z^{(i)}} \leftarrow \overline{\mathsf{Input}}_1(\mathsf{pk}, z^{(i)}, \mathsf{aux}^{(i)}, (\mathsf{ek}_0, \mathsf{ek}_1))$ *for all* $i \in [\ell]$*,*

- $\mathsf{M}_0 \leftarrow \overline{\mathsf{Eval}}_0(\mathsf{ek}_0, (\bar{\mathsf{I}}_0^{z^{(1)}}, \ldots, \bar{\mathsf{I}}_0^{z^{(\ell)}}), P)$,
- $\mathsf{M}_1 \leftarrow \overline{\mathsf{Eval}}_1(\mathsf{ek}_1, (\bar{\mathsf{I}}_1^{z^{(1)}}, \ldots, \bar{\mathsf{I}}_1^{z^{(\ell)}}), (z^{(1)}, \ldots, z^{(\ell)}), P)$,
- $y_\sigma \leftarrow \mathsf{ExtEval}(\sigma, \mathsf{ek}_\sigma, (\mathsf{M}_\sigma, \mathsf{I}_\sigma^{x^{(1)}}, \ldots, \mathsf{I}_\sigma^{x^{(\rho)}}), Q)$, *for* $\sigma \in \{0,1\}$.

Security. *The output of* $\overline{\mathsf{Input}}_1$ *and* Input *should be computationally indistinguishable. Formally, for any* $\lambda \in \mathbb{N}$*, and any* $x \in \mathcal{I}$*, the two following distributions should be computationally indistinguishable:*

$$\left\{ \bar{\mathsf{I}}_1 : \begin{array}{l} (\mathsf{pk}, (\mathsf{ek}_0, \mathsf{ek}_1)) \leftarrow \mathsf{Setup}(1^\lambda) \\ (\bar{\mathsf{I}}_0, \mathsf{aux}) \leftarrow \overline{\mathsf{Input}}_0(\mathsf{pk}) \\ \bar{\mathsf{I}}_1 \leftarrow \overline{\mathsf{Input}}_1(\mathsf{pk}, x, \mathsf{aux}, (\mathsf{ek}_0, \mathsf{ek}_1)) \end{array} \right\} \overset{c}{\approx} \left\{ \mathsf{I}_1 : \begin{array}{l} (\mathsf{pk}, (\mathsf{ek}_0, \mathsf{ek}_1)) \leftarrow \mathsf{Setup}(1^\lambda), \\ (\mathsf{I}_0, \mathsf{I}_1) \leftarrow \mathsf{Input}(\mathsf{pk}, x) \end{array} \right\}.$$

Theorem 7. *Assuming the hardness of* DCR*, there exists HSS scheme following the RMS template which generates simulatable memory values, as well as staged-HSS scheme for the class of RMS programs.*

The above theorem follows from the HSS scheme introduced by Orlandi, Scholl, and Yakoubov in [OSY21] that supports the class of RMS programs and works under the DCR assumption.

5.3 Staged HSS from DCR

In this section we provide instantiations for the three new variants of HSS introduced in section 5.2 under the DCR assumption, therefore proving theorem 7 and theorem 7. In fact, our goal is to show that the HSS scheme introduced by Orlandi, Scholl, and Yakoubov in [OSY21] that supports the class of RMS programs and works under the DCR assumption satisfies the properties of all our three definitions. First, we recall the following lemma due to [OSY21], where they introduce a distributed discrete logarithm algorithm for a subset of $\mathbb{Z}_{N^2}^*$, where $N = pq$ for λ-bit primes p and q.

Lemma 5. *There exists an algorithm* $\mathsf{DDLog}_N(g)$ *for which the following holds: Let* $g_0, g_1 \in \mathbb{Z}_{N^2}^*$*, such that* $g_0 = g_1(1+N)^x (\text{mod } N^2)$*. If* $z_0 = \mathsf{DDLog}_N(g_0)$ *and* $z_1 = \mathsf{DDLog}_N(g_1)$*, then* $z_0 - z_1 = x (\text{mod } N)$*.*

More precisely, $\mathsf{DDLog}_N(g)$ works as follows:

- $\mathsf{DDLog}_N(g)$
  - Write $g = h + h'N$, where $h, h' < N$, using the division algorithm.
  - Output $z = h'h^{-1} \bmod N$.

We now recall the HSS construction of [OSY21] based on circular-secure Paillier encryption (section 2.4.4). The input space of the scheme is $\mathbb{Z}_N$ for a Blum integer $N = pq$.

HSS from Paillier Encryption, [OSY21]

Let $2^{-\kappa}$ be the correctness error. Let $N = pq$ be a Blum integer. Let $\mathcal{P}$ be the set of programs supported by the scheme, and B_{msg} be the magnitude bound of programs in $\mathcal{P}$. We require that $B_{\mathsf{msg}} = N/2^\kappa$. Let $\mathsf{BG} = (\mathsf{BG.KeyGen}, \mathsf{BG.Enc}, \mathsf{BG.Dec})$ be the circular-secure Paillier encryption as in section 2.4.4.

- $\mathsf{Setup}(1^\lambda)$:
  - Run $(\mathsf{BG.pk}, \mathsf{BG.sk}) \leftarrow \mathsf{BG.KeyGen}(1^\lambda)$, and parse them as $\mathsf{BG.pk} = (N, \mathbf{g}, \hat{g})$, and $\mathsf{BG.sk} = \mathbf{d} = (d^{(0)}, \ldots, d^{(\ell-1)})$.
  - Sample $\mathsf{share1}_0$ as a random element of $[2^\kappa]$, and set $\mathsf{share1}_1 := \mathsf{share1}_0 - 1 \bmod N$.
  - For each $i \in [\ell]$, set $\mathsf{share}d^{(i)}{}_0$ to be a random element of $[2^\kappa]$, and set $\mathsf{share}d^{(i)}{}_1 := \mathsf{share}d^{(i)}{}_0 - d^{(i)} \bmod N$.
  - For $i \in [\ell]$, compute $D^{(i)} \leftarrow \mathsf{BG.Enc}(\mathsf{BG.pk}, d^{(i)})$.
  - Sample a PRF key k_{PRF} for a PRF F that outputs values in $\mathbb{Z}_N$.
  - Set and output $\mathsf{pk} = (\mathsf{BG.pk}, D^{(0)}, \ldots, D^{(\ell-1)})$, and $\mathsf{ek}_\sigma = (k_{\mathsf{PRF}}, \mathsf{share1}_\sigma, \mathsf{share}d^{(0)}{}_\sigma, \ldots, \mathsf{share}d^{(\ell-1)}{}_\sigma)$ for each $\sigma \in \{0, 1\}$.
- $\mathsf{Input}(\mathsf{pk}, x)$
  - Parse $\mathsf{pk} = (\mathsf{BG.pk}, D^{(0)}, \ldots, D^{(\ell-1)})$, and $\mathsf{BG.pk} = (\mathbf{g}, \hat{g})$, and $D^{(i)} = (\mathbf{c}^{(i)}, \hat{c}^{(i)})$ for $i \in [\ell]$.
  - Compute $X \leftarrow \mathsf{BG.Enc}(\mathsf{BG.pk}, x)$.
  - For $i \in [\ell]$, compute $X^{(i)} \leftarrow (\mathbf{g}^{r'_i} \cdot (\mathbf{c}^{(i)})^x, \hat{g}^{r'_i} \cdot (\hat{c}^{(i)})^x)$, where $r'_i \xleftarrow{\$} \mathbb{Z}_N$.
  - Set $\mathsf{I} = (X, X^{(0)}, \ldots, X^{(\ell-1)})$, and output $(\mathsf{I}_0 = \mathsf{I}, \mathsf{I}_1 = \mathsf{I})$.
- $\mathsf{Eval}(\sigma, \mathsf{ek}_\sigma, (\mathsf{I}^{(0)}, \ldots, \mathsf{I}^{(n)}), P)$
  This function is divided into the following sub-modules:
  - $\mathsf{ConvertInput}(\sigma, \mathsf{ek}_\sigma, \mathsf{I}_x = (X, X^{(0)}, \ldots, X^{(\ell-1)}))$
    - Set $\mathsf{M}^1_\sigma = (\mathsf{share1}_\sigma, \mathsf{share}d^{(0)}{}_\sigma, \ldots, \mathsf{share}d^{(\ell-1)}{}_\sigma)$ for $\sigma \in \{0, 1\}$.
    - Compute $\mathsf{M}^x_\sigma \leftarrow \mathsf{Mult}(\sigma, \mathsf{ek}_\sigma, \mathsf{I}^x, \mathsf{M}^1_\sigma)$.
  - $\mathsf{Add}(\sigma, \mathsf{ek}_\sigma, \mathsf{M}^x_\sigma, \mathsf{M}^y_\sigma)$
    - Parse $\mathsf{M}^x_\sigma = (\mathsf{share}x_\sigma, \mathsf{share}xd^{(0)}{}_\sigma, \ldots, \mathsf{share}xd^{(\ell-1)}{}_\sigma)$, and $\mathsf{M}^y_\sigma = (\mathsf{share}y_\sigma, \mathsf{share}yd^{(0)}{}_\sigma, \ldots, \mathsf{share}yd^{(\ell-1)}{}_\sigma)$.
    - Compute $\mathsf{share}z_\sigma = \mathsf{share}x_\sigma + \mathsf{share}y_\sigma$, and $\mathsf{share}zd^{(i)}{}_\sigma = \mathsf{share}xd^{(i)}{}_\sigma + \mathsf{share}yd^{(i)}{}_\sigma$ for $i \in [\ell]$.
    - Output $\mathsf{M}^z_\sigma = (\mathsf{share}z_\sigma, \mathsf{share}zd^{(0)}{}_\sigma, \ldots, \mathsf{share}zd^{(\ell-1)}{}_\sigma)$.
  - $\mathsf{Mult}(\sigma, \mathsf{ek}_\sigma, \mathsf{I}^x, \mathsf{M}^y_\sigma)$
    - Parse $\mathsf{I}^x = (X, X^{(0)}, \ldots, X^{(\ell-1)})$ and $\mathsf{M}^y_\sigma = (\mathsf{share}y_\sigma, \mathsf{share}yd^{(0)}{}_\sigma, \ldots, \mathsf{share}yd^{(\ell-1)}{}_\sigma)$.
    - Parse $X = (c_0, \ldots, c_{\ell-1}, \hat{c})$, and $X^{(i)} = (c^{(i)}_0, \ldots, c^{(i)}_{\ell-1}, \hat{c}^{(i)})$ for $i \in [\ell]$.
    - Compute $\mathsf{share}z_\sigma = \mathsf{DDLog}_N\left(\mathsf{ct}'_\sigma\right) \pmod N + F_{k_{\mathsf{PRF}}}(\mathsf{id})$, where
$$\mathsf{ct}'_\sigma = (\hat{c})^{\mathsf{share}y_\sigma} \cdot \left(\prod_{i=0}^{\ell-1} c_i^{-\mathsf{share}yd^{(i)}{}_\sigma}\right) \pmod{N^2}$$
    - For $j \in [\ell]$ compute $\mathsf{share}zd^{(j)}{}_\sigma = \mathsf{DDLog}_N\left(\mathsf{ct}'_{\sigma,j}\right) \pmod N + F_{k_{\mathsf{PRF}}}(\mathsf{id})$, where
$$\mathsf{ct}'_{\sigma,j} = (\hat{c}^{(j)})^{\mathsf{share}y_\sigma} \cdot \left(\prod_{i=0}^{\ell-1} (c^{(j)}_i)^{-\mathsf{share}yd^{(i)}{}_\sigma}\right) \pmod{N^2}$$

- Output $\mathsf{M}_\sigma^z = (\mathsf{share}z_\sigma, \mathsf{share}zd^{(0)}{}_\sigma, \ldots, \mathsf{share}zd^{(\ell-1)}{}_\sigma)$.

- $\mathsf{Output}(\sigma, \mathsf{ek}_\sigma, \mathsf{M}_\sigma^z, n_{\mathsf{out}})$
  - Parse $\mathsf{M}_\sigma^z = (\mathsf{share}z_\sigma, \mathsf{share}zd^{(0)}{}_\sigma, \ldots, \mathsf{share}zd^{(\ell-1)}{}_\sigma)$.
  - Output $\mathsf{share}z_\sigma \pmod{n_{\mathsf{out}}}$.

Figure 5.1: Homomorphic Secret Sharing for Branching Programmes from the security of Paillier's Encryption Scheme

5.3.1 HSS Following the RMS Template from DCR.

We show that fig. 5.1 satisfies definition 17.

Proof. We show how the MemGen algorithm of the template work in this construction. One can see that the other algorithms of the HSS construction exactly follow the template. We define the memory generation algorithm as follows:

- $\mathsf{MemGen}(\sigma, \mathsf{ek}_\sigma, x) \to \mathsf{M}_\sigma^x$
  - If $x = 1$, do:
    - Parse $\mathsf{ek}_\sigma = (k_{\mathsf{PRF}}, \mathsf{share}1_\sigma, \mathsf{share}d^{(0)}{}_\sigma, \ldots, \mathsf{share}d^{(\ell-1)}{}_\sigma)$.
    - Output $\mathsf{M}_\sigma^1 = (\mathsf{share}1_\sigma, \mathsf{share}d^{(0)}{}_\sigma, \ldots, \mathsf{share}d^{(\ell-1)}{}_\sigma)$.
  - Else, do:
    - Run $(\mathsf{I}_0^x, \mathsf{I}_1^x) \leftarrow \mathsf{Input}(\mathsf{pk}, x)$.
    - Run $\mathsf{M}_\sigma^x \leftarrow \mathsf{ConvertInput}(\sigma, \mathsf{ek}_\sigma, \mathsf{I}_\sigma^x)$.
    - Output M_σ^x.

It is easy to see that the outputs of this algorithm are additively homomorphic. This follows from the fact that for any $x \neq 1 \in \mathcal{I}$, this algorithm uses the Input and $\mathsf{Eval.ConvertInput}$ algorithms to generate the memory values. Thus, if the HSS scheme works correctly, the generated memory values are intrinsically homomorphic. More specifically, for an input $z \in \mathcal{I}$, the memory value M_σ^z is of the form $\mathsf{M}_\sigma^z = (\mathsf{share}z_\sigma, \mathsf{share}zd^{(0)}{}_\sigma, \ldots, \mathsf{share}zd^{(\ell-1)}{}_\sigma)$. Furthermore, when $x = 1$, this algorithm outputs a valid share for the vector $(1, d^{(0)}, \ldots, d^{(\ell-1)})$. □

5.3.2 HSS with Simulatable Memory Values from DCR.

We show that HSS-QSY21 satisfies definition 18.

Proof. Regarding definition 18, we need to show that there exist two algorithms Sim_0 and Sim_1 that simulate the output of MemGen. We define them as follows:

- $\mathsf{Sim}_0(1^\lambda) \to \mathsf{M}_0$
  - Sample a random vector $(t, t_0, \ldots, t_{\ell-1}) \xleftarrow{\$} [2^\kappa \cdot N]^{\ell+1}$.
  - Output $\mathsf{M}_0 = (t, t_0, \ldots, t_{\ell-1})$.
- $\mathsf{Sim}_1(\mathsf{M}, z, (\mathsf{ek}_0, \mathsf{ek}_1)) \to \mathsf{M}_1$
  - Parse $\mathsf{ek}_\sigma = (\mathsf{share}1_\sigma, \mathsf{share}d^{(0)}{}_\sigma, \ldots, \mathsf{share}d^{(\ell-1)}{}_\sigma)$ for both $\sigma \in \{0, 1\}$.

- For $i \in [\ell]$ reconstruct $d^{(i)} = \mathsf{share}d^{(i)}_0 - \mathsf{share}d^{(i)}_1 \bmod N$.
- Compute and output $\mathsf{M}_1 = \mathsf{M}_0 - (z, zd^{(0)}, \ldots, zd^{(\ell-1)})$.

We prove the following two properties regarding the simulation algorithms:

Simulation Correctness. For any $z \in \mathbb{Z}_N$, it holds that

$$\mathsf{M}_0 - \mathsf{M}_1 = (z, zd^{(0)}, \ldots, zd^{(\ell-1)}),$$

where $\mathsf{M}_0 \leftarrow \mathsf{Sim}_0(1^\lambda)$, and $\mathsf{M}_1 \leftarrow \mathsf{Sim}_1(\mathsf{M}, z, (\mathsf{ek}_0, \mathsf{ek}_1))$. Therefore, the simulated memory values of z are correctly formed as subtractive shares of vector $(z, zd^{(0)}, \ldots, zd^{(\ell-1)})$. Thus, they are valid shares. This guarantees the correctness of multiplication between this values and real input values, and finally the correctness of $\mathsf{ExtEval}$ in lemma 4 when M_σ is simulated.

Simulation Security. We need to prove that for any $x \in \mathcal{I}$, it holds that

$$(z, \mathsf{M}_0) \approx_s (z, \mathsf{M}_1),$$

where $\mathsf{M}_1 \leftarrow \mathsf{Sim}_1(\mathsf{M}_0, z, (\mathsf{ek}_0, \mathsf{ek}_1))$, and $\mathsf{M}_0 \leftarrow \mathsf{Sim}_0(1^\lambda)$.
Note that $\mathsf{M}_1 = \mathsf{M}_0 - (z, zd^{(0)}, \ldots, zd^{(\ell-1)})$, where each element of M_0 is chosen uniformly from $\mathbb{Z}_{2^\kappa N}$. Also, in a fixed vector $(z, zd^{(0)}, \ldots, zd^{(\ell-1)})$, x and each $zd^{(i)}$ for $i \in [\ell]$ are elements of $\mathbb{Z}_N$. Therefore, the distribution of each element of M_1 is within the statistical distance $2^{-\kappa}$ of the uniform distribution over $\mathbb{Z}_{2^\kappa N}$ which is the distribution of M_0. □

5.3.3 Staged HSS from DCR.

We prove that assuming the hardness of DCR, fig. 5.1 satisfies definition 19.

Proof. We explicitly define four algorithms $(\overline{\mathsf{Input}}_0, \overline{\mathsf{Input}}_1)$ and $(\overline{\mathsf{Eval}}_0, \overline{\mathsf{Eval}}_1)$ according to definition 19. We define the four algorithms as follows:

- $\overline{\mathsf{Input}}_0(\mathsf{pp}) \to (\bar{\mathsf{I}}_0, \mathsf{aux})$

  - Parse $\mathsf{pp} = (\mathsf{BG.pk}, D^{(0)}, \ldots, D^{(\ell-1)})$, and $\mathsf{BG.pk} = (N, \mathbf{g}, \hat{g})$.
  - Sample $r \xleftarrow{\$} \mathbb{Z}_N$ and compute $\mathsf{ct}_{\mathsf{ind}} = \mathbf{g}^r$.
  - For $i \in [\ell]$ do:
    * Sample $r_i \xleftarrow{\$} \mathbb{Z}_N$.
    * Compute $\mathsf{ct}^{(i)}_{\mathsf{ind}} = \mathbf{g}^{r_i}$.
  - Set $\bar{\mathsf{I}}_0 = (\mathsf{ct}_{\mathsf{ind}}, \mathsf{ct}^{(0)}_{\mathsf{ind}}, \ldots, \mathsf{ct}^{(\ell-1)}_{\mathsf{ind}})$.
  - Set $\mathsf{aux} = (\mathbf{g}^r, \hat{g}^r, \{\mathbf{g}^{r_i}\}_{i\in[\ell]}, \{\hat{g}^{r_i}\}_{i\in[\ell]})$.
  - Output $(\bar{\mathsf{I}}_0, \mathsf{aux})$.

- $\overline{\mathsf{Input}}_1(\mathsf{pp}, x, \mathsf{aux}, (\mathsf{ek}_0, \mathsf{ek}_1)) \to \bar{\mathsf{I}}_1$

- Parse $\mathsf{pp} = (\mathsf{BG.pk}, D^{(0)}, \ldots, D^{(\ell-1)})$, $\mathsf{BG.pk} = (N, \mathbf{g}, \hat{g})$, $\mathsf{aux} = (\mathbf{g}^r, \hat{g}^r, \{\mathbf{g}^{r_i}\}_{i\in[\ell]}, \{\hat{g}^{r_i}\}_{i\in[\ell]})$, and $\mathsf{ek}_\sigma = (k_{\mathsf{PRF}}, \mathsf{share}d^{(0)}{}_\sigma, \ldots, \mathsf{share}d^{(\ell-1)}{}_\sigma)$ for $\sigma \in \{0,1\}$.
- Compute $\mathsf{ct} = (\mathbf{g}^r, \hat{g}^r \cdot (1+N)^x)$.
- For $i \in [\ell]$ do
  * Reconstruct $d^{(i)} = \mathsf{share}d^{(i)}{}_0 - \mathsf{share}d^{(i)}{}_1 \bmod N$.
  * Compute $\mathsf{ct}^{(i)} = (\mathbf{g}^{r_i}, \hat{g}^{r_i} \cdot (1+N)^{xd^{(i)}})$.
- Output $\bar{\mathsf{I}}_1 = (\mathsf{ct}, \mathsf{ct}^{(0)}, \ldots, \mathsf{ct}^{(\ell-1)})$.

- $\overline{\mathsf{Eval}}_0(\mathsf{ek}_0, (\bar{\mathsf{I}}_0^{(1)}, \ldots, \bar{\mathsf{I}}_0^{(\rho)}), P) \to \mathsf{M}_0$

  - $\overline{\mathsf{ConvertInput}}_0(\mathsf{ek}_0, \bar{\mathsf{I}}_x)$ //same as in Eval
    - Parse $\mathsf{ek}_0 = (\mathsf{share}1_0, \mathsf{share}d^{(0)}{}_0, \ldots, \mathsf{share}d^{(\ell-1)}{}_0)$.
    - Set $\mathsf{M}_0^1 = (\mathsf{share}1_0, \mathsf{share}d^{(0)}{}_0, \ldots, \mathsf{share}d^{(\ell-1)}{}_0)$.
    - Compute $\mathsf{M}_0^x \leftarrow \overline{\mathsf{Mult}}_0(\mathsf{ek}_0, \bar{\mathsf{I}}_0, \mathsf{M}_0^1)$.

  - $\overline{\mathsf{Add}}_0(\mathsf{ek}_0, \mathsf{M}_0^x, \mathsf{M}_0^y)$ //same as in Eval
    - Parse $\mathsf{M}_0^x = (\mathsf{share}x_0, \mathsf{share}xd^{(0)}{}_0, \ldots, \mathsf{share}xd^{(\ell-1)}{}_0)$, and $\mathsf{M}_0^y = (\mathsf{share}y_0, \mathsf{share}yd^{(0)}{}_0, \ldots, \mathsf{share}yd^{(\ell-1)}{}_0)$.
    - Compute $\mathsf{share}z_0 = \mathsf{share}x_0 + \mathsf{share}y_0$, and $\mathsf{share}zd^{(i)}{}_0 = \mathsf{share}xd^{(i)}{}_0 + \mathsf{share}yd^{(i)}{}_0$ for $i \in [\ell]$.
    - Output $\mathsf{M}_0^z = (\mathsf{share}z_0, \mathsf{share}zd^{(0)}{}_0, \ldots, \mathsf{share}zd^{(\ell-1)}{}_0)$.

  - $\overline{\mathsf{Mult}}_0(\mathsf{ek}_0, \bar{\mathsf{I}}_0^x, \mathsf{M}_0^y)$
    - Parse $\bar{\mathsf{I}}_0^x = (\mathsf{ct}_{\mathsf{ind}}, \mathsf{ct}_{\mathsf{ind}}^{(0)}, \ldots, \mathsf{ct}_{\mathsf{ind}}^{(\ell-1)})$, and $\mathsf{M}_0^y = (\mathsf{share}y_0, \mathsf{share}yd^{(0)}{}_0, \ldots, \mathsf{share}yd^{(\ell-1)}{}_0)$.
    - Parse $\mathsf{ct}_{\mathsf{ind}} = (c_0, \ldots, c_{\ell-1})$, and $\mathsf{ct}_{\mathsf{ind}}^{(i)} = (c_0^{(i)}, \ldots, c_{\ell-1}^{(i)})$ for $i \in [\ell]$.
    - Compute $\mathsf{share}z_0 = \mathsf{DDLog}_N(\mathsf{ct}')(\bmod N) + F_{k_{\mathsf{PRF}}}(\mathsf{id})$, where
    $$\mathsf{ct}' = \prod_{i=0}^{\ell-1} (c_i)^{-\mathsf{share}yd^{(i)}{}_0} (\bmod N^2).$$
    - For $j \in [\ell]$, compute $\mathsf{share}zd^{(j)}{}_0 = \mathsf{DDLog}_N(\mathsf{ct}'_j)(\bmod N) + F_{k_{\mathsf{PRF}}}(\mathsf{id})$, where
    $$\mathsf{ct}'_j = \prod_{i=0}^{\ell-1} (c_i^{(j)})^{-\mathsf{share}yd^{(i)}{}_0} (\bmod N^2).$$
    - Output $\mathsf{M}_0^z = (\mathsf{share}z_0, \mathsf{share}zd^{(0)}{}_0, \ldots, \mathsf{share}zd^{(\ell-1)}{}_0)$.

- $\overline{\mathsf{Eval}}_1(\mathsf{ek}_1, (\bar{\mathsf{I}}_1^{(1)}, \ldots, \bar{\mathsf{I}}_1^{(\rho)}), (x^{(1)}, \ldots, x^{(\rho)}), P) \to \mathsf{M}_1$

  - $\overline{\mathsf{ConvertInput}}_1(\mathsf{ek}_1, \bar{\mathsf{I}}_x, x)$ //same as in Eval
    - Parse $\mathsf{ek}_1 = (\mathsf{share}1_1, \mathsf{share}d^{(0)}{}_1, \ldots, \mathsf{share}d^{(\ell-1)}{}_1)$.
    - Set $\mathsf{M}_1^1 = (\mathsf{share}1_1, \mathsf{share}d^{(0)}{}_1, \ldots, \mathsf{share}d^{(\ell-1)}{}_1)$.

- Compute $\mathsf{M}_1^x \leftarrow \overline{\mathsf{Mult}}_1(\mathsf{ek}_1, \bar{\mathsf{I}}_1, \mathsf{M}_1^1, x)$.

- $\overline{\mathsf{Add}}_1(\mathsf{ek}_1, \mathsf{M}_1^x, \mathsf{M}_1^y)$ //same as in Eval
  - Parse $\mathsf{M}_1^x = (\mathsf{share}x_1, \mathsf{share}xd^{(0)}{}_1, \ldots, \mathsf{share}xd^{(\ell-1)}{}_1)$, and $\mathsf{M}_1^y = (\mathsf{share}y_1, \mathsf{share}yd^{(0)}{}_1, \ldots, \mathsf{share}yd^{(\ell-1)}{}_1)$.
  - Compute $\mathsf{share}z_1 = \mathsf{share}x_1 + \mathsf{share}y_1$, and $\mathsf{share}zd^{(i)}{}_1 = \mathsf{share}xd^{(i)}{}_1 + \mathsf{share}yd^{(i)}{}_1$ for $i \in [\ell]$.
  - Output $\mathsf{M}_1^z = (\mathsf{share}z_1, \mathsf{share}zd^{(0)}{}_1, \ldots, \mathsf{share}zd^{(\ell-1)}{}_1)$.

- $\overline{\mathsf{Mult}}_1(\mathsf{ek}_0, \bar{\mathsf{I}}_0^x, \mathsf{M}_0^y, y)$
  - Parse $\bar{\mathsf{I}}_1^x = (\mathsf{ct}, \mathsf{ct}^{(0)}, \ldots, \mathsf{ct}^{(\ell-1)})$, and $\mathsf{M}_1^y = (\mathsf{share}y_1, \mathsf{share}yd^{(0)}{}_1, \ldots, \mathsf{share}yd^{(\ell-1)}{}_1)$.
  - Parse $\mathsf{ct} = (c_0, \ldots, c_{\ell-1}, \hat{c})$, and $\mathsf{ct}^{(i)} = (c_0^{(i)}, \ldots, c_{\ell-1}^{(i)}, \hat{c}^{(i)})$ for $i \in [\ell]$.
  - Compute $\mathsf{share}z_1 = \mathsf{DDLog}_N(\mathsf{ct}')(\mathrm{mod}\ N) + F_{k_{\mathsf{PRF}}}(\mathsf{id})$, where
$$\mathsf{ct}' = (\hat{c})^y \cdot \prod_{i=0}^{\ell-1} (c_i)^{-\mathsf{share}yd^{(i)}{}_1} (\mathrm{mod}\ N^2).$$
  - For $j \in [\ell]$, compute $\mathsf{share}zd^{(j)}{}_1 = \mathsf{DDLog}_N(\mathsf{ct}'_j)(\mathrm{mod}\ N) + F_{k_{\mathsf{PRF}}}(\mathsf{id})$, where
$$\mathsf{ct}'_j = (\hat{c}^{(j)})^y \cdot \prod_{i=0}^{\ell-1} (c_i^{(j)})^{-\mathsf{share}yd^{(i)}{}_1} (\mathrm{mod}\ N^2).$$
  - Output $\mathsf{M}_1^z = (\mathsf{share}z_1, \mathsf{share}zd^{(0)}{}_1, \ldots, \mathsf{share}zd^{(\ell-1)}{}_1)$.

Correctness. We show that a memory value M_σ^y outputted by $\overline{\mathsf{Eval}}_\sigma$ is in fact party σ's subtractive share of vector $(y, yd^{(0)}, \ldots, yd^{(\ell-1)})$, thus it's valid. This guarantees the correctness of ExtEval algorithm when given as input a staged memory value and a vector of original input values.

Since the new evaluation algorithms $\overline{\mathsf{Eval}}_0$ and $\overline{\mathsf{Eval}}_1$ work the same as the original evaluation algorithm Eval except for the multiplication instruction, we briefly prove the correctness of multiplication in the following. Let $x, y \in \mathcal{I}$ be any two arbitrary input values. We show that

$$\Pr\left[z_0 - z_1 = xy\right] \geq 1 - \mathsf{negl}(\lambda),$$

and

$$\Pr\left[(zd^{(i)})_0 - (zd^{(i)})_1 = xyd^{(i)}\right] \geq 1 - \mathsf{negl}(\lambda),$$

for all $i \in [\ell]$, where
$\mathsf{M}_0^z = \left(z_0, (zd^{(0)})_0, \ldots, (zd^{(\ell-1)})_0\right) \leftarrow \overline{\mathsf{Mult}}_0(\mathsf{ek}_0, \bar{\mathsf{I}}_0^x, \mathsf{M}_0^y)$,
$\mathsf{M}_1^z = \left(z_1, (zd^{(0)})_1, \ldots, (zd^{(\ell-1)})_1\right) \leftarrow \overline{\mathsf{Mult}}_1(\mathsf{ek}_1, \bar{\mathsf{I}}_1^x, \mathsf{M}_1^y, y)$,
$(\bar{\mathsf{I}}_0^b, \mathsf{aux}^b) \leftarrow \overline{\mathsf{Input}}_0(\mathsf{pk})$ for $b \in \{x, y\}$,
$\bar{\mathsf{I}}_1^b \leftarrow \overline{\mathsf{Input}}_1(\mathsf{pk}, b, \mathsf{aux}^b)$, for $b \in \{x, y\}$,
$\mathsf{M}_0^y \leftarrow \overline{\mathsf{ConvertInput}}_0(\mathsf{ek}_0, \bar{\mathsf{I}}_0^y)$,
$\mathsf{M}_1^y \leftarrow \overline{\mathsf{ConvertInput}}_1(\mathsf{ek}_1, \bar{\mathsf{I}}_1^y, y)$, and
$(\mathsf{pk}, (\mathsf{ek}_0, \mathsf{ek}_1)) \leftarrow \mathsf{Setup}(1^\lambda)$.

Regarding how $\overline{\mathsf{Mult}}_0$ and $\overline{\mathsf{Mult}}_1$ works, it holds that $z_b = \mathsf{DDLog}_N(\mathsf{ct}'_b)$, for $b \in \{0,1\}$. Thus, by lemma 5, it's enough to prove that $\mathsf{ct}'_0 \cdot \mathsf{ct}'_1 = (1+N)^{xy}$. We have

$$\begin{aligned}
\mathsf{ct}'_0 \cdot \mathsf{ct}'_1 &= \prod_{i=0}^{\ell-1} (c_i)^{-\mathsf{share}yd^{(i)}{}_0} \cdot (\hat{c})^y \cdot \prod_{i=0}^{\ell-1} (c_i)^{-\mathsf{share}yd^{(i)}{}_1} \\
&= (\hat{c})^y \cdot \prod_{i=1}^{\ell} (c_i)^{-yd^{(i)}} \\
&= (1+N)^{xy} \cdot \prod_{i=1}^{\ell} (c_i)^{yd^{(i)}} \cdot \prod_{i=1}^{\ell} (c_i)^{-yd^{(i)}} \\
&= (1+N)^{xy} (\text{mod } N^2).
\end{aligned}$$

The equation $\mathsf{ct}_0^{'(j)} \cdot \mathsf{ct}_1^{'(j)} = (1+N)^{xyd^{(j)}}$ for all $j \in [\ell]$ is proved similarly.

Security. Outputs of the $\overline{\mathsf{Input}}_1$ algorithm are in fact in the same form as the Input algorithm. More precisely, they are both Paillier encryptions of the vector $(x, xd^{(0)}, \ldots, xd^{(\ell-1)})$, where d is the secret key of the encryption scheme. Therefore, they are computationally indistinguishable. □

5.4 Sublinear-Communication Secure Two-Party Computation with One-Sided Statistical Security from Staged HSS

5.4.1 In the $\mathcal{F}^{\mathsf{HSS}}_{\mathsf{update}}$-Hybrid Model.

The main component (apart from the HSS scheme itself) in building sublinear secure computation from HSS is the low-communication distributed share generation. When using staged-HSS, the first party can simply sample its share locally, so the hard part is updating the second party so they can receive their share too. We formalize this task in Figure 5.2 as the ideal functionality $\mathcal{F}^{\mathsf{HSS}}_{\mathsf{update}}$. We prove in Lemma 6 that there exists sublinear two-party secure computation, provided this step can be performed with one-sided statistical security and with low-enough communication.

Functionality $\mathcal{F}^{\mathsf{HSS}}_{\mathsf{update}}$

The functionality is parameterised with a staged-HSS scheme $\mathsf{stagedHSS} = (\mathsf{stagedHSS.Setup}, \mathsf{stagedHSS.Input}, \mathsf{stagedHSS.MemGen}, \mathsf{stagedHSS.Eval})$.

Input: Wait to receive $(\mathtt{share}, \mathsf{stagedHSS.pk}, \bar{\mathsf{I}}_0, \mathsf{aux})$ from P_0 and $(\mathtt{input}, x_1)$ from P_1.
Output: Compute $\bar{\mathsf{I}}_1 \leftarrow \mathsf{stagedHSS}.\overline{\mathsf{Input}}_1(\mathsf{stagedHSS.pk}, x_1, \mathsf{aux})$; output $(\mathsf{stagedHSS.pk}, \bar{\mathsf{I}}_1)$ to P_1.

Figure 5.2: Ideal functionality $\mathcal{F}^{\mathsf{HSS}}_{\mathsf{update}}$, parameterized by a staged-HSS scheme, for generating the second input share given the first, precomputed, one.

Functionality $\mathcal{F}_{\mathsf{SFE}}(C)$

The functionality is parameterised with an arithmetic circuit $C\colon \mathcal{R}^{n_0} \times \mathcal{R}^{n_1} \to \mathcal{R}^m$ over a finite ring R.

Input: Wait to receive $(\mathtt{input}, \sigma, x_\sigma)$ from each party P_σ ($\sigma \in \{0,1\}$), where $x_\sigma \in \mathbb{F}^{n_\sigma}$, and set $\vec{x} \leftarrow (x_0, x_1)$.

Output: Compute $\vec{y} \leftarrow C(\vec{x})$; Sample $\vec{y}_0 \xleftarrow{\$} \mathcal{R}^m$; Set $\vec{y}_1 \leftarrow \vec{y} - \vec{y}_0$; Output $\vec{y}_0$ to P_0 and $\vec{y}_1$ to P_1.

Figure 5.3: Ideal functionality $\mathcal{F}_{\mathsf{SFE}}(C)$ for the two-party secure evaluation of an arithmetic circuit C.

Protocol Π_C

Parties: Alice and Bob
Parameters: The protocol is parameterized with:

- $C\colon \mathbb{F}^{n_0} \times \mathbb{F}^{n_1} \to \mathbb{F}^m$ is an arithmetic circuit over finite field $\mathbb{F}$.
- $\mathsf{HSS} = (\mathsf{HSS.Setup}, \mathsf{HSS.Input}, \mathsf{HSS.MemGen}, \mathsf{HSS.Eval})$ is a staged-HSS scheme with pseudorandom shares supporting the class of RMS programs over $\mathbb{F}$ (seen as a ring). We denote the staged input and evaluation algorithms by $(\mathsf{HSS.}\overline{\mathsf{Input}}_0, \mathsf{HSS.}\overline{\mathsf{Input}}_1)$ and $(\mathsf{HSS.}\overline{\mathsf{Eval}}_0, \mathsf{HSS.}\overline{\mathsf{Eval}}_1)$. Let $\mathsf{HSS.ExtEval}$ be defined as in Lemma 4.
- $F(\cdot,\cdot)$ is a PRF in NC^1 with domain $\{0,1\}^\lambda$, key space $\{0,1\}^\lambda$, and range $\mathbb{F}^{n_1}$.

Hybrid Model: The protocol is defined in the $\mathcal{F}^{\mathsf{HSS}}_{\mathsf{update}}$-hybrid model.
Input: Alice holds $x_0 \in \mathbb{F}^{n_0}$ and Bob holds $x_1 \in \mathbb{F}^{n_1}$.
The Protocol:

Alice's precomputation phase. Alice does the following:

1. $K \xleftarrow{\$} \{0,1\}^\lambda$
2. $(\mathsf{HSS.pk}, \mathsf{ek}_0, \mathsf{ek}_1) \leftarrow \mathsf{HSS.Setup}(1^\lambda)$
3. $(\bar{\mathsf{I}}_0, \mathsf{aux}) \leftarrow \mathsf{HSS.}\overline{\mathsf{Input}}_0(\mathsf{HSS.pk})$
4. $(\mathsf{I}_0, \mathsf{I}_1) \leftarrow \mathsf{HSS.Input}(1^\lambda, K)$
5. $\alpha \xleftarrow{\$} \{0,1\}^\lambda$, $c_{\mathsf{in}} \leftarrow x_0 + F(K, \alpha)$, and $r_{\mathsf{out}} \xleftarrow{\$} \mathbb{F}^m$
6. $M_0 \leftarrow \mathsf{HSS.}\overline{\mathsf{Eval}}(\mathsf{ek}_0, \bar{\mathsf{I}}_0, F(\cdot, \alpha))$
7. $y_0 \leftarrow \mathsf{HSS.ExtEval}(\mathsf{ek}_0, (M_0, I_0), f_{\alpha, c_{\mathsf{in}}})$,
   where $f_{\alpha, c_{\mathsf{in}}}\colon (X, Y) \mapsto C(c_{\mathsf{in}} - F(X, \alpha), Y)$

Online phase.

8. Alice sends $(\mathsf{ek}_1, \mathsf{I}_1, c_{\mathsf{in}}, \alpha, r_{\mathsf{out}})$ to Bob, who waits to receive it.

9. Alice sends $(\texttt{share}, \mathsf{HSS.pk}, \bar{\mathsf{I}}_0, \mathsf{aux})$ to $\mathcal{F}_{\mathsf{update}}^{\mathsf{HSS}}$;
   Bob sends $(\texttt{input}, x_1)$ to $\mathcal{F}_{\mathsf{update}}^{\mathsf{HSS}}$, and waits to receive $(\mathsf{HSS.pk}, \bar{\mathsf{I}}_1)$ from $\mathcal{F}_{\mathsf{update}}^{\mathsf{HSS}}$.

Bob's computation phase. Bob does the following:

1. $M_1 \leftarrow \mathsf{HSS.}\overline{\mathsf{Eval}}(\mathsf{ek}_1, \bar{\mathsf{I}}_1, F(\cdot, \alpha))$
2. $y_1 \leftarrow \mathsf{HSS.ExtEval}(\mathsf{ek}_1, (M_1, I_1), f_{\alpha, c_{\mathsf{in}}})$,
   where $f_{\alpha,c_{\mathsf{in}}} \colon (X, Y) \mapsto C(c_{\mathsf{in}} - F(X, \alpha), Y)$

Output phase. Alice outputs $y_0' \leftarrow y_0 + r_{\mathsf{out}}$; Bob outputs $y_1' \leftarrow y_1 - r_{\mathsf{out}}$.

Figure 5.4: (Sublinear) Secure Two-Party Computation with One-Sided Statistical Security from staged-HSS Supporting the Class of RMS Programs.

Lemma 6 (Secure Computation with One-Sided Statistical Security in the $\mathcal{F}_{\mathsf{update}}^{\mathsf{HSS}}$-hybrid model)**.** *Let $C \colon \mathbb{F}^{n_0} \times \mathbb{F}^{n_1} \to \mathbb{F}^m$ be an arithmetic circuit over a finite field $\mathbb{F}$. Let* stagedHSS *be a staged-HSS scheme with pseudorandom shares supporting the class of RMS programs over $\mathbb{F}$ (seen as a ring).*
The protocol Π_C provided in Figure 5.4 UC-securely implements the two-party functionality $\mathcal{F}_{\mathsf{SFE}}(C)$ in the $\mathcal{F}_{\mathsf{update}}^{\mathsf{HSS}}$-hybrid model, against a passive adversary statically corrupting at most one of the parties, with perfect security against Alice, and computational security against Bob. The protocol uses $\lambda^{\mathcal{O}(1)} + (n_1 + m) \cdot \log |\mathbb{F}|$ bits of communication.

Proof. First observe that if the PRF $F(\cdot, \cdot)$ and C have logarithmic depth then $f_{\alpha,c_{\mathsf{in}}}$ can indeed be expressed as an RMS program, and we cause the PRF from [NRR00]. The required amount of communication follows from inspection of the protocol, and we are left to prove security. Let $\mathcal{A}$ be a semi-honest, static adversary that interacts with parties Alice and Bob running protocol Π_C in the $\mathcal{F}_{\mathsf{update}}^{\mathsf{HSS}}$-hybrid model. We need to construct a simulator Sim such that no environment $\mathcal{Z}$ can distinguish with non-negligible probability whether it is interacting with $\mathcal{A}$ and parties running Π_C in the $\mathcal{F}_{\mathsf{update}}^{\mathsf{HSS}}$-hybrid model, or with Sim and $\mathcal{F}_{\mathsf{SFE}}(C)$. Because the adversary is static, we can assume that the set of corrupted parties is fixed before the start of the protocol, and we can consider the cases of a corrupted Alice and a corrupted Bob separately.

- **Perfect security against a corrupted Alice.** Alice receives no messages (from Bob or $\mathcal{F}_{\mathsf{update}}^{\mathsf{HSS}}$) in the real execution, therefore a simulator can trivially perfectly simulate the joint view of $\mathcal{Z}$ and $\mathcal{A}$ in the execution of Π_C.

- **Computational security against a corrupted Bob.** Consider the simulator Sim which is given as input the corrupted Bob's input $x_1 \in \mathbb{F}^{n_1}$, runs an internal copy of $\mathcal{A}$, and acts as follows:

  - **Simulating the communication with $\mathcal{Z}$:** Every input value that Sim receives from $\mathcal{Z}$ is written on $\mathcal{A}$'s input tape (as if coming from $\mathcal{A}$'s environment), and every output value written by $\mathcal{A}$ on its output tape is copied to Sim's own output tape (to be read by $\mathcal{Z}$).
  - **Simulating the protocol's execution:**
    1. Sim sends $(\texttt{input}, 1, x_1)$ to $\mathcal{F}_{\mathsf{SFE}}(C)$ and waits to receive y_1'.

2. Sim samples $K \xleftarrow{\$} \{0,1\}^\lambda$, $(\mathsf{pk}, \mathsf{ek}_0, \mathsf{ek}_1) \leftarrow \mathsf{HSS.Setup}(1^\lambda)$, $(\bar{\mathsf{I}}_0, \mathsf{aux}) \leftarrow \mathsf{HSS.}\overline{\mathsf{Input}}_0(\mathsf{pk})$, $(\mathsf{I}_0, \mathsf{I}_1) \leftarrow \mathsf{HSS.Input}(1^\lambda, K)$, $\alpha \xleftarrow{\$} \{0,1\}^\lambda$.

3. Sim sets $c_{\mathsf{in}} \leftarrow F(K, \alpha)$.

4. Sim computes $\bar{\mathsf{I}}_1 \leftarrow \mathsf{stagedHSS.Share}(\mathsf{pk}, x_1, \mathsf{aux})$.

5. Sim computes $M_1 \leftarrow \mathsf{HSS.}\overline{\mathsf{Eval}}(\mathsf{ek}_1, \bar{\mathsf{I}}_1, F(\cdot, \alpha))$.

6. Sim sets $r_{\mathsf{out}} \leftarrow \mathsf{HSS.Eval}'(\mathsf{ek}_1, (M_1, I_1), f_{\alpha, c_{\mathsf{in}}}) - y'_1$.

7. Sim writes $(\mathsf{ek}_1, \bar{\mathsf{I}}_1, c_{\mathsf{in}}, \alpha, r_{\mathsf{out}})$ on $\mathcal{A}$'s input tape (as if Alice sent it to Bob).

8. Sim proceeds with the execution of $\mathcal{A}$ until the latter writes $(\mathtt{input}, x_1)$ on its output tape (the message from Bob to $\mathcal{F}^{\mathsf{HSS}}_{\mathsf{update}}$), at which point it writes $(\mathsf{HSS.pk}, \bar{\mathsf{I}}_1)$ on $\mathcal{A}$'s input tape (as if sent to Bob by $\mathcal{F}^{\mathsf{HSS}}_{\mathsf{update}}$).

Observe that the difference in the joint view of $\mathcal{Z}$ and $\mathcal{A}$ in the real and ideal worlds boils down to how c_{in} and r_{out} are defined. However, because in the ideal world y'_1 is uniformly distributed and independent from the coins of Sim, r_{out} is uniformly distributed in the ideal world. Further, because there is no entropy in the single outgoing message of Bob (the message $(\mathtt{input}, 1, x_1)$ he sends to $\mathcal{F}_{\mathsf{OLE}}$), the internal coins of Bob are irrelevant in the joint view of $\mathcal{Z}$ and $\mathcal{A}$. Let $(x_0, x_1) \in \mathbb{F}^{n_0} \times \mathbb{F}^{n_1}$. From what precedes, it suffices to show that the distributions of outputs of the following probability experiments are computationally indistinguishable:

$\underline{\mathsf{RealView}(1^\lambda, x_0, x_1)}$:	$\underline{\mathsf{IdealView}(1^\lambda, x_1)}$:
$K \xleftarrow{\$} \{0,1\}^\lambda$	$K \xleftarrow{\$} \{0,1\}^\lambda$
$\alpha \xleftarrow{\$} \{0,1\}^\lambda$	$\alpha \xleftarrow{\$} \{0,1\}^\lambda$
$c_{\mathsf{in}} \leftarrow x_0 + F(K, \alpha)$	$c_{\mathsf{in}} \leftarrow F(K, \alpha)$
$(\mathsf{pk}, \mathsf{ek}_0, \mathsf{ek}_1) \leftarrow \mathsf{HSS.Setup}(1^\lambda)$	$(\mathsf{pk}, \mathsf{ek}_0, \mathsf{ek}_1) \leftarrow \mathsf{HSS.Setup}(1^\lambda)$
$(\bar{\mathsf{I}}_0, \mathsf{aux}) \leftarrow \mathsf{HSS.}\overline{\mathsf{Input}}_0(\mathsf{pk})$	$(\bar{\mathsf{I}}_0, \mathsf{aux}) \leftarrow \mathsf{HSS.}\overline{\mathsf{Input}}_0(\mathsf{pk})$
$(\mathsf{I}_0, \mathsf{I}_1) \leftarrow \mathsf{HSS.Input}(1^\lambda, K)$	$(\mathsf{I}_0, \mathsf{I}_1) \leftarrow \mathsf{HSS.Input}(1^\lambda, K)$
$\bar{\mathsf{I}}_1 \leftarrow \mathsf{stagedHSS.Share}(\mathsf{pk}, x_1, \mathsf{aux})$	$\bar{\mathsf{I}}_1 \leftarrow \mathsf{stagedHSS.Share}(\mathsf{pk}, x_1, \mathsf{aux})$
$r_{\mathsf{out}} \xleftarrow{\$} \mathbb{F}^m$	$r_{\mathsf{out}} \xleftarrow{\$} \mathbb{F}^m$
Output $(\mathsf{ek}_1, \mathsf{I}_1, c_{\mathsf{in}}, \alpha, r_{\mathsf{out}}, x_1, \mathsf{pk}, \bar{\mathsf{I}}_1)$	Output $(\mathsf{ek}_1, \mathsf{I}_1, c_{\mathsf{in}}, \alpha, r_{\mathsf{out}}, x_1, \mathsf{pk}, \bar{\mathsf{I}}_1)$

To that end, consider the following experiments (which differ with the above only in the law of $(\mathsf{I}_0, \mathsf{I}_1)$):

$\underline{\mathsf{HybridReal}}(1^\lambda, x_0, x_1)$:

$K \xleftarrow{\$} \{0,1\}^\lambda$

$\alpha \xleftarrow{\$} \{0,1\}^\lambda$

$c_{\mathsf{in}} \leftarrow x_0 + F(K, \alpha)$

$(\mathsf{pk}, \mathsf{ek}_0, \mathsf{ek}_1) \leftarrow \mathsf{HSS.Setup}(1^\lambda)$

$(\bar{\mathsf{I}}_0, \mathsf{aux}) \leftarrow \mathsf{HSS}.\overline{\mathsf{Input}}_0(\mathsf{pk})$

$(\mathsf{I}_0, \mathsf{I}_1) \leftarrow \mathsf{HSS.Input}(1^\lambda, 0^\lambda)$

$\bar{\mathsf{I}}_1 \leftarrow \mathsf{stagedHSS.Share}(\mathsf{pk}, x_1, \mathsf{aux})$

$r_{\mathsf{out}} \xleftarrow{\$} \mathbb{F}^m$

Output $(\mathsf{ek}_1, \mathsf{I}_1, c_{\mathsf{in}}, \alpha, r_{\mathsf{out}}, x_1, \mathsf{pk}, \bar{\mathsf{I}}_1)$

$\underline{\mathsf{HybridIdeal}}(1^\lambda, x_1)$:

$K \xleftarrow{\$} \{0,1\}^\lambda$

$\alpha \xleftarrow{\$} \{0,1\}^\lambda$

$c_{\mathsf{in}} \leftarrow F(K, \alpha)$

$(\mathsf{pk}, \mathsf{ek}_0, \mathsf{ek}_1) \leftarrow \mathsf{HSS.Setup}(1^\lambda)$

$(\bar{\mathsf{I}}_0, \mathsf{aux}) \leftarrow \mathsf{HSS}.\overline{\mathsf{Input}}_0(\mathsf{pk})$

$(\mathsf{I}_0, \mathsf{I}_1) \leftarrow \mathsf{HSS.Input}(1^\lambda, 0^\lambda)$

$\bar{\mathsf{I}}_1 \leftarrow \mathsf{stagedHSS.Share}(\mathsf{pk}, x_1, \mathsf{aux})$

$r_{\mathsf{out}} \xleftarrow{\$} \mathbb{F}^m$

Output $(\mathsf{ek}_1, \mathsf{I}_1, c_{\mathsf{in}}, \alpha, r_{\mathsf{out}}, x_1, \mathsf{pk}, \bar{\mathsf{I}}_1)$

Let $\mathcal{A}^{\mathsf{Real}}_{x_0,x_1}$ be a PPT adversary which distinguishes between the real view $\{\mathsf{RealView}(1^\lambda, x_0, x_1)\}$ and $\{\mathsf{HybridReal}(1^\lambda, x_0, x_1)\}$ with probability ϵ_{Real}. By the security of HSS, for every PPT adversaries $\mathcal{A}, \mathcal{A}'$, and any bit $\sigma \in \{0,1\}$:

$$\left| \Pr \left[b' = b : \begin{array}{l} (K_0, K_1, \mathsf{st}) \leftarrow \mathcal{A}(1^\lambda) \\ (\mathsf{pk}, (\mathsf{ek}_0, \mathsf{ek}_1)) \leftarrow \mathsf{Setup}(1^\lambda) \\ b \xleftarrow{\$} \{0,1\} \\ (\mathsf{I}_0, \mathsf{I}_1) \leftarrow \mathsf{Input}(K_b) \\ b' \leftarrow \mathcal{A}'(\mathsf{st}, \mathsf{pk}, \mathsf{ek}_\sigma, \mathsf{I}_\sigma) \end{array} \right] - \frac{1}{2} \right| \leq \mathsf{negl}(\lambda) \ .$$

Considering the following PPT algorithms $\mathcal{A}_{x_0}$ and $\mathcal{A}'_{x_1}$ proves that ϵ_{Real} must be negligible:

- $\mathcal{A}_{x_0}(1^\lambda)$: $K \xleftarrow{\$} \{0,1\}^\lambda$; $\alpha \xleftarrow{\$} \{0,1\}^\lambda$; $\mathsf{st} \leftarrow (x_0 + F(K,\alpha), \alpha)$; Output $(K, 0^\lambda, \mathsf{st})$.
- $\mathcal{A}'_{x_1}(\mathsf{st}, \mathsf{pk}, \mathsf{ek}_1, \mathsf{I}_1)$: $(\bar{\mathsf{I}}_0, \mathsf{aux}) \leftarrow \mathsf{HSS}.\overline{\mathsf{Input}}_0(\mathsf{pk})$;
  $\bar{\mathsf{I}}_1 \leftarrow \mathsf{stagedHSS.Share}(\mathsf{pk}, x_1, \mathsf{aux})$; $r_{\mathsf{out}} \xleftarrow{\$} \{0,1\}^\lambda$;
  Output $\mathcal{A}^{\mathsf{Real}}_{x_0,x_1}(\mathsf{ek}_1, \mathsf{I}_1, \mathsf{st.first}, \mathsf{st.second}, r_{\mathsf{out}}, x_1, \mathsf{pk}, \bar{\mathsf{I}}_1)$.

Therefore $\{\mathsf{RealView}(1^\lambda, x_0, x_1)\} \overset{\mathrm{c}}{\approx} \{\mathsf{HybridReal}(1^\lambda, x_0, x_1)\}$, and similarly $\{\mathsf{IdealView}(1^\lambda, x_1)\} \overset{\mathrm{c}}{\approx} \{\mathsf{HybridIdeal}(1^\lambda, x_1)\}$. Finally, because all other random variables than c_{in} are independent of K (both in $\mathsf{HybridReal}(1^\lambda, x_0, x_1)$ and in $\mathsf{HybridIdeal}(1^\lambda, x_1)$), we can conclude using a straightforward reduction to the security of the PRF $F(\cdot,\cdot)$ that the two distributions $\{\mathsf{HybridReal}(1^\lambda, x_0, x_1)\}$ and $\{\mathsf{HybridIdeal}(1^\lambda, x_1)\}$ are computationally indistinguishable. Wrapping up, this implies that the joint view of $\mathcal{Z}$ and $\mathcal{A}$ is indistinguishable in the real and ideal worlds.

□

5.4.2 Instantiating $\mathcal{F}^{\mathsf{HSS}}_{\mathsf{update}}$ under DCR.

We now show how to instantiate $\mathcal{F}^{\mathsf{HSS}}_{\mathsf{update}}$ for construction of staged-HSS from DCR (fig. 5.1). This instantiation is non black-box in the HSS scheme, and uses a combination of the Paillier-ElGamal encryption scheme, which is provably semantically secure under DCR, and oblivious linear evaluation (OLE) with one-sided statistical security, which is known from DCR.

Functionality $\mathcal{F}_{\mathsf{OLE}}$

The functionality $\mathcal{F}_{\mathsf{OLE}}$ for (batch) oblivious linear evaluation is parameterized by a finite field $\mathbb{F}$, and interacts with two parties P_0 and P_1.

Input: Wait to receive $(\texttt{input}, 0, \mathbf{u} = (u_1, \ldots, u_s))$ (where $u_1, \ldots, u_s \in \mathbb{F}$) from P_0 and $(\texttt{input}, 1, \mathbf{v} = (v_1, \ldots, v_t))$ (where $v_1, \ldots, v_t \in \mathbb{F}$) from P_1.
Output: Compute $\mathbf{z} \leftarrow (u_i \cdot v_j)_{i \in [s], j \in [t]}$, sample $\mathbf{z}_0 \xleftarrow{\$} \mathbb{F}^{s \cdot t}$, set $\mathbf{z}_1 \leftarrow \mathbf{z} - \mathbf{z}_0$; Output $\mathbf{z}_\sigma$ to P_σ for $\sigma \in \{0, 1\}$.

Figure 5.5: Ideal functionality $\mathcal{F}_{\mathsf{OLE}}$ for (batch) oblivious linear evaluation.

Protocol $\Pi^{\mathsf{HSS}}_{\mathsf{update}}$

Parties: Alice and Bob.
Parameters: $\mathbb{F}_{2^\lambda}$ is an exponential-size finite field; n_1 is an input size. $\mathsf{stagedHSS}$ is the staged-HSS scheme of fig. 5.1, instantiated from Paillier-ElGamal. The Paillier-ElGamal cryptosystem itself is parameterized by GenPQ, an algorithm that on input 1^λ, generates $(N = p \cdot q, p, q)$, where p and q are $\ell(\lambda)$-bit primes where $\ell \colon \mathbb{N}^\star \to \mathbb{N}^\star$ is a function such that $\forall \kappa \in \mathbb{N}^\star, \ell(\kappa) \geq 1.5\kappa$. $B_{\mathsf{sk}} \coloneqq 2^{2\ell(\lambda) - 2\log|\mathbb{F}|}$ is the base for the decomposition of the secret key into digits; $s \coloneqq 2\ell(\lambda) + 2\log|\mathbb{F}|$ is the number of cyphertexts needed to encrypt the secret key; $t \coloneqq \lceil n_1 \frac{\log|\mathbb{F}|}{2\ell(\lambda)} \rceil$.
Hybrid Model: The protocol is defined in the $\mathcal{F}_{\mathsf{OLE}}$-hybrid model.
Input: Alice holds $(\mathsf{HSS.pk}, \bar{\mathsf{I}}_0, \mathsf{aux})$ and Bob holds $x_1 = (x_1^{(1)}, \ldots, x_1^{(t)}) \in \mathcal{R}^{n_1} \approx [N]^t$.
The Protocol:

1. Alice does the following:

   - Parse $\mathsf{HSS.pk} = (\mathsf{pk}_{\mathsf{PaillierEG}}, D^{(0)}, \ldots, D^{(s-1)})$
     // $D^{(j)}$ is a Paillier-ElGamal encryption under pk of the j^{th} digit of the secret key in base B_{sk}
   - Parse $\bar{\mathsf{I}}_0 = (\mathsf{ct}_{\mathsf{ind}}, (\mathsf{ct}_{\mathsf{ind}}^{(i,j)})_{(i,j) \in [t] \times [s+1]})$
     // $\mathsf{ct}_{\mathsf{ind}}$ is of the form g^r, and $\mathsf{ct}_{\mathsf{ind}}^{(i,j)}$ is of the form $g^{r_{i,j}}$
   - Parse $\mathsf{aux} = (g^r, \mathsf{pk}^r_{\mathsf{PaillierEG}}, (g^{r_{i,j}})_{(i,j) \in [t] \times [s+1]}, (\mathsf{pk}^{r_{i,j}}_{\mathsf{PaillierEG}})_{(i,j) \in [t] \times [s+1]})$
     // $\mathsf{pk}_{\mathsf{PaillierEG}} = g^{\mathsf{sk}_{\mathsf{PaillierEG}}} \bmod N^2$

2. Alice sends $(N, \mathsf{pk}_{\mathsf{PaillierEG}}, \mathsf{ct}_{\mathsf{ind}})$ to Bob

3. Alice sends $(\texttt{input}, 0, (1 \| d))$ to $\mathcal{F}_{\mathsf{OLE}}$ and waits to receive $\mathbf{y}^{(0)} = (y_{i,j}^{(0)})_{(i,j) \in [t] \times [s+1]}$;
   Bob sends $(\texttt{input}, 1, x_1)$ to $\mathcal{F}_{\mathsf{OLE}}$ and waits to receive $\mathbf{y}^{(1)} = (y_{i,j}^{(1)})_{(i,j) \in [t] \times [s+1]}$.
   // Adding the digit 1 to the secret key d condenses the notations of the encryption of the input alone, and those of the input times each digit of the secret key, as $x \cdot (1, d_0, \ldots, d_{s-1}) = (x, x \cdot d_0, \ldots, x \cdot d_{s-1})$.

4. Alice does the following:

   For each $(i, j) \in [t] \times [s+1]$, $c_{i,j} \leftarrow (1+N)^{y_{i,j}^{(0)}} \cdot h^{r_{i,j}}$

5. Alice sends $\mathbf{c} = (c_{i,j})_{(i,j)\in[t]\times[s+1]}$ to Bob, who waits to receive it.

6. Bob sets $\mathsf{ct}_{\mathsf{dep}} \leftarrow (c_{i,j} \cdot (1+N)^{y^{(1)}_{i,j}})_{(i,j)\in[t]\times[s+1]}$ and outputs $\bar{\mathsf{I}}_1 \leftarrow (\mathsf{ct}_{\mathsf{ind}}, \mathsf{ct}_{\mathsf{dep}})$.

Figure 5.6: Protocol for securely realizing $\mathcal{F}^{\mathsf{HSS}}_{\mathsf{update}}$ under the circular security of the Paillier-ElGamal cryptosystem.

Lemma 7 (Instantiating Lemma 6 under DCR)**.** *Let* HSS *be the staged-HSS scheme of fig. 5.1. Assuming the* DCR *assumption holds, the protocol* $\Pi^{\mathsf{HSS}}_{\mathsf{update}}$ *provided in Figure 5.6 UC-securely implements the two-party functionality* $\mathcal{F}^{\mathsf{HSS}}_{\mathsf{update}}$ *in the* $\mathcal{F}_{\mathsf{OLE}}$*-hybrid model, against a passive adversary statically corrupting at most one of the parties, with perfect security against Alice and Bob. The protocol uses* $\mathcal{O}(\lambda \cdot n_1)$ *bits of communication.*

Proof. Set $\ell(\lambda) = \Theta(\lambda)$ (and therefore $s = \mathcal{O}(1)$) such that $\ell(\lambda) \geq \frac{3}{2}\lambda$. Sending N requires $2\ell(\lambda)$ bits of communication, h requires $4\ell(\lambda)$, $\mathsf{ct}_{\mathsf{ind}}$ and $\vec{c}$ require $\mathcal{O}(\ell(\lambda) \cdot n_1)$. We now analyse security. Let $\mathcal{A}$ be a semi-honest, static adversary that interacts with parties Alice and Bob running protocol Π_C in the $\mathcal{F}^{\mathsf{HSS}}_{\mathsf{update}}$-hybrid model. We need to construct a simulator Sim such that no environment $\mathcal{Z}$ can distinguish with non-negligible probability whether it is interacting with $\mathcal{A}$ and Π_C in the $\mathcal{F}^{\mathsf{HSS}}_{\mathsf{update}}$-hybrid model, or with Sim and $\mathcal{F}_{\mathsf{SFE}}(C)$. Because the adversary is static, we can assume that the set of corrupted parties is fixed before the start of the protocol, and we can consider the cases of a corrupted Alice and a corrupted Bob separately.

- **Perfect security against a corrupted Alice.** Alice receives no messages from Bob in the real execution and the single message Alice receives from $\mathcal{F}_{\mathsf{OLE}}$ is a uniformly random value in $(\mathbb{F}_{2^\lambda})^{t\cdot(s+1)}$, so it is easy to see that joint view of $\mathcal{Z}$ and $\mathcal{A}$ in the execution of $\Pi^{\mathsf{HSS}}_{\mathsf{update}}$ can be simulated perfectly.

- **Perfect security against a corrupted Bob.** Consider the simulator Sim which is given as input the corrupted Bob's input $x_1 \in (\mathbb{F}_{2^\lambda})^{n_1}$, internally runs a copy of $\mathcal{A}$, and acts as follows:

  - **Simulating the communication with $\mathcal{Z}$:** Every input value that Sim receives from $\mathcal{Z}$ is written on $\mathcal{A}$'s input tape (as if coming from $\mathcal{A}$'s environment), and every output value written by $\mathcal{A}$ on its output tape is copied to Sim's own output tape (to be read by $\mathcal{Z}$).
  - **Simulating the protocol's execution:**
    1. Sim sends $(\mathtt{input}, x_1)$ to $\mathcal{F}_{\mathsf{SFE}}(C)$ and waits to receive $(\mathsf{HSS.pk}, \bar{\mathsf{I}}_1 = (\mathsf{ct}_{\mathsf{ind}}, \mathsf{ct}_{\mathsf{dep}}))$.
    2. Parse $\mathsf{HSS.pk} = (\mathsf{pk}_{\mathsf{PaillierEG}}, D^{(0)}, \ldots, D^{(s-1)})$ and recover N from $\mathsf{pk}_{\mathsf{PaillierEG}}$.
    3. Parse $\mathsf{ct}_{\mathsf{dep}} = (\mathsf{ct}^{(i,j)}_{\mathsf{dep}})_{(i,j)\in[t]\times[s+1]}$
    4. Sim writes $(N, \mathsf{pk}_{\mathsf{PaillierEG}}, \mathsf{ct}_{\mathsf{ind}})$ on $\mathcal{A}$'s input tape (as if sent by Alice to Bob).
    5. Sample $\vec{y}^{(1)} = (y^{(1)}_{(i,j)})_{(i,j)\in[t]\times[s+1]} \xleftarrow{\$} (\mathbb{F}_{2^\lambda})^{[t]\times[s+1]}$
    6. For $(i,j) \in [t] \times [s+1]$, set $c_{(i,j)} \leftarrow \mathsf{ct}^{(i,j)}_{\mathsf{dep}} \cdot (1+N)^{-y^{(1)}_{i,j}}$.
    7. Sim proceeds with the execution of $\mathcal{A}$ until the latter writes $(\mathtt{input}, 1, x_1)$ on its output tape (the message from Bob to $\mathcal{F}_{\mathsf{OLE}}$), at which point it writes $\vec{y}^{(1)}$ on $\mathcal{A}$'s input tape (as if sent to Bob by $\mathcal{F}_{\mathsf{OLE}}$).

8. Sim writes $\vec{c}$ on $\mathcal{A}$'s input tape (as if sent by Alice to Bob).

If follows from inspection that this simulation is perfect.

□

We then obtain our final claim.

Theorem 8 (Computation for NC^1 with Circuit-Independent-Communication and One-Sided Statistical Security from Circular Security of Paillier-ElGamal)**.** *Let C be an RMS program with $n = n_0 + n_1$ inputs and m outputs over $\mathbb{F}_{2^\lambda}$. Assuming DCR and the circular security of the Paillier-ElGamal encryption scheme, there exists a protocol that UC-securely implements the two-party functionality $\mathcal{F}_{\mathsf{SFE}}(C)$, against a passive adversary that statically corrupts at most one of the parties, with perfect security against a corrupted Alice, and computational security against a corrupted Bob. The protocol uses $\lambda^{\mathcal{O}(1)} + \mathcal{O}((n+m) \cdot \log |\mathbb{F}|)$ bits of communication.*

Proof. The proof of Theorem 8 follows from a combination of Lemmas 6 and 7, as well as a linear-communication protocol for $\mathbb{F}_{2^\lambda}$-OLE (which we batch "naively" in order to instantiate $\mathcal{F}_{\mathsf{OLE}}$, which in fact corresponds to $t \cdot (s+1)$-batch OLE). Such a protocol is folklore, and can be achieved *e.g.* by using Gilboa's [Gil99] information-theoretic reduction of OLE to string-OT and OT-extension [IKNP03, ALSZ17]. □

Chapter 6

Towards a Complete Primitive for Sublinear-Communication Two-Party Computation

This chapter describes results which have been communicated previously in [BCM22].

Based on Joint works with Elette Boyle and Geoffroy Couteau.

We present a new approach toward secure two-party computation protocols for general layered circuits, with communication complexity that scales sublinearly in the circuit size. As opposed to building FHE or HSS, our approach begins with protocols for "batch Oblivious Transfer" with low communication.

Oblivious Transfer (OT) is an atomic functionality in which sender and receiver parties begin with inputs $m_0, m_1 \in \{0,1\}$ and $b \in \{0,1\}$, respectively; at the conclusion the receiver learns the selected message m_b; and neither party learns further information about one another's inputs. OT was shown to be a complete functionality for general secure computation [Kil00], where OT protocol execution(s) take place for each nonlinear gate of the corresponding circuit.

OT protocols are known from a number of standard assumptions, in just two rounds of communication (i.e., one message from receiver to sender, and one message in return); but, the communication complexity for all such solutions is (inherently) significantly larger than the input size. Very recently, it was shown by Brakerski et al. [BBDP22] how to achieve a *batched* version of OT, still in two rounds, and with *rate-1* communication. That is, for a collection of message pairs $(\{m_0^{(i)}, m_1^{(i)}\})_{i\in[\ell]}$ and selection bits $(b^{(i)})_{i\in[\ell]}$, a sender and receiver could perform ℓ parallel batched executions of OT in communication roughly ℓ.

We prove that any such protocol which satisfies an additional *decomposability* property suffices to imply secure computation protocols for general layered circuits with sublinear communication complexity. To define decomposability, consider the communication structure of any 2-round rate-1 batch OT protocol. In the first round, the receiver sends $\ell + o(\ell)$ bits to the sender,[1] somehow encoding its selection bits $b^{(i)}$. In response, the sender performs some computation as a function of its message pairs $\{m_0^{(i)}, m_1^{(i)}\}$, and returns $\ell + o(\ell)$ bits in response, somehow encoding the k selected messages, $m_{b^{(i)}}^{(i)}$. For the constructions of [BBDP22], the sender's message size is just $\ell + \mathsf{polylog}(\ell)$.

[1] Our construction can actually handle arbitrary *constant* client-to-server upload rate, as long as the sender-to-receiver download rate is 1.

We say that the (2-round, rate-1) batch OT protocol is *decomposable* if for any agreed subset $S \subset [\ell]$ of indices, the sender can choose a corresponding subset of $|S|+\mathsf{polylog}(\ell)$ of its return message bits, such that sending this partial sender response reveals *exactly* the corresponding subset of selected messages $(m^{(i)}_{b^{(i)}})_{i \in S}$ to the receiver. Namely, given the partial response, these $|S|$ messages can be recovered, and no information is revealed about $m^{(i)}_{b^{(i)}}$ for $i \notin S$.

Theorem 9 (Sublinear 2PC from Decomposable Batch OT - informal)**.** *Assume existence of 2-round rate-1 batch OT with the above "decomposability" property. Then for any k, we can securely compute layered (synchronous) circuits of depth d and size s using $\mathsf{poly}(2^{2^k}, s)$ computation and $O(2^{2^k} \cdot d \cdot \mathsf{poly}(\lambda) + s/k)$ communication.*
In particular, for $k = O(\log \log s)$, we obtain communication $O(s/\log \log s + d^{1/3} \cdot s^{2(1+\varepsilon)/3} \cdot \mathsf{poly}(\lambda))$, for an arbitrary small constant ε. The latter is sublinear in s whenever $d = o(s^{1-\varepsilon}/\mathsf{poly}(\lambda))$, i.e. , the circuit is not too "tall and skinny".

This decomposability property is not simply hypothetical, but rather was inspired by the batch-OT protocols of Brakerski et al. [BBDP22], which we show to satisfy the requirement. At a high level, the sender's message in their protocols consists of an encryption of the selected message bits (computed homomorphically as a function of receiver-sent ciphertexts of its selection bits, together with the message pairs $\{m^{(i)}_0, m^{(i)}_1\}$), compressed *à la* [DGI+19] to rate 1. The resulting rate-1 ciphertexts have the structure of a $\mathsf{polylog}(\ell)$-size "header" string, independent of the messages, together with a *single bit* of information for each encrypted message bit. Decomposability thus follows (almost) directly, by simply omitting those information bits corresponding to encrypted messages the sender wishes to drop (i.e., $[\ell] \setminus S$).[2]
In turn, we obtain the following corollary.

Corollary 3 (Sublinear 2PC from QR+LPN - informal)**.** *The conclusion of Theorem 9 holds based on Quadratic Residuosity (QR) and Learning Parity with Noise (LPN) for any inverse-polynomial noise rate.*

We finally mention that this result is also not implied by the constructions of pseudorandom correlation functions (PCF) [BCG+20] from QR+LPN of [OSY21] (or in fact any of the line of work on pseudorandom correlation generators (PCG) [BCG+19b]). While PCG/PCFs enable the generation of large quantities of *random* instances of OT with sublinear communication, the best known approaches for utilizing these random correlations within an actual secure computation protocol require communication that scales linearly with the circuit size.

6.1 Overview of this Chapter's Results

We consider Boolean circuits over any base of gates with fan-in two. Toward our sublinear 2PC result for layered circuits, we begin by focusing on circuits of low depth k (e.g., think of $k = \log \log \log s$), and devise a secure protocol with communication $n + m + (2^{2^k} \cdot \mathsf{poly}(\lambda))$, for input size n, output size m, circuit size s, and security parameter λ. Given such a tool, we can appropriately divide a larger layered circuit into depth-k blocks where the sum of all block input and output sizes is s/k, and then iteratively compute (secret shares of) each layer output via the sub-protocol. Combined, this yields a secure computation for the layered circuit with overall communication $O(s/k + 2^{2^k} \cdot d \cdot \mathsf{poly}(\lambda))$, as desired.

[2] We are of course sweeping details under the rug here, and refer the reader to the main body for a more complete treatment.

6.1.1 Starting point: An SPIR viewpoint.

Consider a circuit with input size n, output size m, and low depth k. Given fan-in 2, each output bit is computed as a function of at most 2^k input bits. We may thus view the circuit output as dictated by m separate truth tables, each of size 2^{2^k}, indexed by the values of the corresponding relevant 2^k input bits. More concretely, think of one party as holding the (partially collapsed) truth tables incorporating its known inputs, and the second party as holding its own input string, dictating the relevant position of each truth table. We will refer to the first party as "sender" and second as "receiver."
Given this perspective, protocols for (Symmetric) Private Information Retrieval (SPIR) immediately come to mind. An SPIR protocol is a strengthened version of PIR, where the client additionally learns nothing beyond its queried value of the database. Secure computation of our circuit precisely amounts to m instances of SPIR, where the receiver party learns exactly the desired indexed values of the m truth tables.
However, the situation is not so simple: Even the best known (S)PIR protocols have communication polylogarithmic in the database size. Applying m instances of SPIR for the m outputs would thus yield communication $\mathsf{polylog}(2^k) \cdot m \in \Omega(km)$, killing sublinearity.
In order to obtain sublinear communication, we must somehow leverage that the m SPIR instances are not completely independent, but rather are made with *correlated* queries. That is, although there are m instances each with (2^k)-bit index values, the $m \cdot 2^k$ selection bits have several repeats, collectively coming from different subsets of only $n < m \cdot 2^k$ input bits.

6.1.2 Toward batch SPIR with correlated queries.

Our task becomes precisely to construct such an object: m-instance batch SPIR, with significantly lower communication complexity given correlated queries.
For purposes of discussion, suppose there existed a 2-round rate-1 protocol for oblivious transfer, where each sender and receiver (magically) sends only a single bit. Given access to such a tool, then by leveraging ideas from the literature (e.g., achieving PIR from linearly homomorphic encryption [KO97]), we would be set. Indeed, the receiver would simply send 1 bit for each input bit, corresponding to the first OT message using this value as a selector bit. These first messages could then be *reused* by the sender in multiple, recursive executions.
More concretely, suppose the server holds a database of N bits and that the receiver wants to retrieve the element stored at index $x = (x_1, \ldots, x_{\log N})$. If the receiver sends a message otr_1 generated as its first-round OT-receiver message for the first bit x_1 of the desired index, the server can take the database, pair up elements whose indices differ only on the first bit, then apply the OT-server computation with respect to otr_1 on each pair in order to retrieve a single-bit response for each, creating a new "database" of half the number of elements, each corresponding to a 1-bit sender answer message.
If instead the receiver sends messages $(\mathsf{otr}_1, \ldots, \mathsf{otr}_{\log N})$, one for each bit of the desired index, the server can now iteratively compress the database down to a single bit by building a "Merkle tree" where in each recursive iteration corresponding to input index bit x_i, the new "database" is split into pairs of messages whose indices differ only in this index, and performing the OT-server computation on each pair produces a new list of 1-bit sender answer messages of again half the length. At the conclusion, the server will be left with a single message value remaining, which by construction precisely enables the receiver to recover the target value stored at index x. This approach extends directly for m distinct databases with the *same* total receiver message $(\mathsf{otr}_1, \ldots, \mathsf{otr}_{\log N})$, since

the corresponding OT-receiver messages can be used independently in any mix and match format across databases. In turn, the sender would need to send only m total bits response, one bit for each database query.
Of course, unfortunately, we do not have such a strong rate-1 OT. We thus turn to the next closest alternative which does exist: 2-round rate-1 *batch* OT, as recently achieved by Brakerski et al. [BBDP22]. Batch OT considers a collection of ℓ message pairs $(\{m_0^{(i)}, m_1^{(i)}\})_{i\in[\ell]}$ and selection bits $(b^{(i)})_{i\in[\ell]}$, and enables a sender and receiver to perform ℓ parallel batched executions of OT with communication roughly ℓ.
Attempting to apply the above strategy with rate-1 batch OT, however, poses significant challenges.

- The batching structure restricts the "mix and match" abilities of the sender when using the receiver's OT message. The sender must respond to the *entire* batched vector of receiver's selection bits at any stage, without freely accessing subsets of selection bits. Instead, the above approach involves using each selection bit $b^{(i)}$ within a *different* number $(N/2^i)$ of message pairs.

- Even worse, the sender's (batch) response in general is only defined given *all* ℓ *pairs of messages* to be selected by the bits $b^{(1)}, \ldots, b^{(\ell)}$.

  In contrast, the above approach crucially relies on the ability to choose the message pairs for selection bit $b^{(i)}$ *dynamically* as a function of the server's responses given the previous selection bits $b^{(1)}, \ldots, b^{(i-1)}$.

- Finally, it is no longer the case that for each selected message the sender has a *single* corresponding response bit. In fact, rate 1 here does not even mean that for ℓ instances that exactly ℓ bits are sent in each direction, but rather just asymptotically $\ell + o(\ell)$. This means that in each recursive OT execution, the sender's messages (and thus "database entry" size) may grow, leading to large growth and ultimately large communication upon further recursions.

6.1.3 Decomposable batch OT.

With this motivation, we introduce the notion of *decomposable (2-round, rate-1) batch oblivious transfer*, which can be seen as a strengthening of two-round batch OT with constant upload-rate (i.e. the size of the receiver message is linear in the batch size ℓ) and download-rate asymptotically one (i.e. the size of the sender message is $\ell + o(\ell)$). The differences boil down to a notion of *decomposability* which we impose on the sender message.
At a high level, what we want to capture is the fact that the receiver should be able to retrieve the i^{th} selected message in the batch if and only it also has access to the i^{th} bit of the sender message (using its own internal state saved from generating the receiver message). More generally, given only a subset of the bits of the sender message, the receiver should able to retrieve the corresponding subset of selected messages in the batch.
Slightly more formally, we say that the (2-round, rate-1) batch OT protocol is *decomposable* if for any agreed subset $S \subset [\ell]$ of indices, the sender can choose a corresponding subset of $|S| + \mathsf{polylog}(\ell)$ of its return message bits, such that sending this partial sender response reveals *exactly* the corresponding subset of selected messages $(m^{(i)}_{b^{(i)}})_{i\in S}$ to the receiver. Namely, given the partial response, these $|S|$ messages can be recovered, and no information is revealed about $m^{(i)}_{b^{(i)}}$ for $i \notin S$.

For our purposes, it will suffice to consider a relaxation of the notion we just described, and allow the sender message to have some small overhead rather than having a one-to-one correspondence between the bits on the sender message and the ℓ selected messages. In this relaxed form, we require that the sender message be comprised of two parts: a "reusable" part (of size $o(\ell)$), and a "decomposable" part (of size ℓ). On its own, the reusable part should reveal nothing about the messages, but can be used to "decode" each bit of the decomposable part so as to retrieve (exactly) the corresponding selected message in the batch. Among other benefits of this relaxation, it allows us to consider constructions whose download-rate is only asymptotically one.
This decomposability property is not only enough for our needs, but perhaps more importantly, is *achievable*, in fact achieved by the batch OT constructions of [BBDP22]. Roughly speaking, the sender message in their construction is composed of a rate-1 encryption of the vector of requested message bits, with structure consisting of a short "header" independent of the message bits, together with a single ciphertext bit encoding each message bit separately. Decomposability can then be achieved by sending only those ciphertext bits encoding the desired subset of messages.
Slightly more accurately, this describes the situation for all but an inverse-polynomial fraction of message bits (corresponding to noisy coordinates of an LPN ciphertext sent by the receiver), which actually encode the *incorrect* messages. In order to separately address these values, they employ a "co-PIR" (or "punctured OT" [BGI17]) to efficiently mask out the undesired values from the receiver, and a separate PIR to learn the correct values for these positions. The separate PIR query responses appear as part of the short "header" information of the server's response, which may sound like an issue, as this portion should not reveal information directly about any message bits. However, this problem does not occur, because the extra PIR queries are set up to actually reveal the *difference* between the masked-out incorrect message ($r_i \oplus m_{1-b}$) and the target message m_b. Because of the mask, this difference value (revealed in the header) provides no information about any message in the absence of the corresponding value ($r_i \oplus m_{1-b}$) from the payload portion of the ciphertext, as required by decomposability.

6.1.4 Sublinear 2PC from decomposable batch OT.

This decomposability property directly allows us to address one of the above challenges of batch OT: we will not have issues with exponential growth of the database entry size in the recursive OT executions. Instead, the result of one iteration of the batch OT on n inputs will result in a short $o(n)$-size header together with n bits that each provide information about a distinct queried message. The header string we will put to the side (ultimately we will send the collection of all the headers, which is still sufficiently short). The remaining n bits induce the recursive sender-message database that, as desired, consists of exactly 1 bit per message.
In fact, if we temporarily suppose that the assignment graph structure of n input bits to $m = n$ output bits can be decomposed as the disjoint union of 2^k matchings, then we have a solution. Each disjoint matching will correspond directly to a different instance of n-input batch OT, where each of the n inputs is simultaneously used to index a different database. Applying the recursive solution as above, the sender will ultimately compute a single bit for each output, as well as a collection of header strings from each of the batch OT executions.
The remaining challenge is that general circuits do *not* have such nice regular structure, instead with inputs appearing in different numbers of output computations, with inconvenient correlations, demanding a stronger form of "mix and match" of batched

OT queries beyond a direct approach.
To address this issue, we modify the structure of batch OT receiver queries, effectively extending the batch size (say from n to $2n$), and employing a careful choice of how to pack extra copies of more highly influential input bits into the queried vector, so that the overall total number of batch OT instances remains sufficiently small that the overhead of extra header strings does not negatively impact the final communication complexity. We refer the reader to the remainder of this chapter for a formal and detailed treatment of this procedure.

6.2 Correlated Symmetric PIR

6.2.1 Correlated Symmetric PIR with "Mix and Match" Queries

Definition 20 ("Mix and Match" Functions)**.** *A "mix and match" function* $\mathsf{MixAndMatch}\colon \{0,1\}^w \to [N]^k$ *is one parameterised by* k *ordered subsets of* $n \coloneqq \log N$ *elements of* $[w]$*,* $S_j = (s_{j,1}, \ldots, s_{j,n}) \in [w]^n$ *for* $j \in [k]$ *such that:*

$$\forall \vec{\alpha} = (\alpha_1, \ldots, \alpha_w) \in \{0,1\}^w, \mathsf{MixAndMatch}(\alpha_1, \ldots, \alpha_w) \coloneqq (x_1, \ldots, x_k), \\ \text{with } x_j \coloneqq \alpha_{s_{j,1}} \cdots \alpha_{s_{j,n}} \in [N].$$

Such a function is associated with an occurrence function, which counts the occurrences of each input position in the outputs:

$$\begin{array}{rccl} t.\colon & [w] & \to & [k] \\ & i & \mapsto & t_i = \sum_{j=1}^{k} \mathbf{1}_{i \in S_j} \end{array}$$

Each t_i *(*$i \in [w]$*) can be decomposed as* $t_i = t_{i,1} + \cdots + t_{i,n}$*, where* $t_{i,j'}$ *is equal to the number of values of* $j \in [k]$ *such that* $s_{j,j'} = i$*.*

- $\mathsf{MixAndMatch}$ *is said to be* T-balanced *if* $\forall i \in [w], \forall j' \in [n], t_{i,j'} \leq T$*.*
- $\mathsf{MixAndMatch}$ *is said to be* T-balanceable *if it can be expressed as the function* $\mathsf{MixAndMatch} = (\mathsf{MixAndMatch}' \circ \mathsf{replicate})$*, where* $\mathsf{MixAndMatch}\colon \{0,1\}^{w'} \to [N]^k$ *is a* T*-balanced mix-and-match function and* $\mathsf{replicate}$ *is defined as:*

$$\begin{array}{rccl} \mathsf{replicate}\colon & \{0,1\}^w & \to & \{0,1\}^{w'} \\ & (b_1, \ldots, b_w) & \mapsto & (b_1^{\|\lceil t_1/T \rceil} \| \ldots \| b_w^{\|\lceil t_w/T \rceil}) \end{array} \qquad \text{where } w' \coloneqq \sum_{i \in [w]} \lceil t_i/T \rceil.$$

Lemma 8. *Let* $w, n \in \mathbb{N}$ *be a sufficiently large integers. For any family of unordered subsets* $S_1, \ldots, S_k \in \binom{[w]}{n}$ *there exists an ordering of each subset* S_j *such that the mix-and-match function induced by the resulting* $(\tilde{S}_j)_{j \in [k]}$ *is* $\mathsf{polylog}(w)$*-balanceable. Furthermore, such orderings can be found in expected constant time.*

Proof. Let us prove, via a Chernoff bound, that reordering (after replication) each subset independently and uniformly at random, yields a $\mathsf{polylog}(w)$-balanced mix-and-match function with high probability. It follows that any family of subsets characterises a T balanceable mix-and-match function.

Preliminary Notations. Let $S_1 = \{s_{1,1}, \ldots, s_{1,n}\}, \ldots, S_k = \{s_{k,1}, \ldots, s_{k,n}\} \in \binom{[w]}{n}$ be a family of unordered sets, and consider the associated occurrence function, which counts the occurrences of each input position in each subset:

$$\begin{array}{rccl} t.: & [w] & \to & [k] \\ & i & \mapsto & t_i = \sum_{j=1}^{k} \mathbf{1}_{i \in S_j} \end{array}$$

Define $w' \coloneqq \sum_{i \in [w]} \lceil t_i / T \rceil$.

Analysis of Randomized Construction. Consider the i.i.d. random variables $\pi_1, \ldots, \pi_k \hookleftarrow \mathcal{U}(\mathfrak{S}_{w'})$ (where $\mathcal{U}(\mathfrak{S}_{[w']})$ is the uniform distribution on all permutations of $[w']$). Define the random variables $\tilde{S}_1, \ldots \tilde{S}_k$ as the following deterministic functions of the random variables $\pi_1, \ldots, \pi_k$: for each $j \in [k]$, $\tilde{S}_j \coloneqq (s_{j,\pi_j(1)}, \ldots, s_{j,\pi_j(w')})$. Finally, define the indicator random variables $(t_{i,j,d})_{i \in [w'], j \in [k], d \in [n]}$ as the following deterministic functions of the $(\pi_j)_{j \in [k]}$: for each $i \in [w'], j \in [k], d \in [n]$, $t_{i,j,d} \coloneqq \mathbf{1}_{i == s_{j,d}}$.

Observe that the event "$\tilde{S}_1, \ldots \tilde{S}_k$ characterizes a T-balanced mix-and-match function" (for any T) is equivalent to the event "$\forall d \in [n], \forall i \in [w'], \sum_{j \in k} t_{i,j,d} \leq T$". Since the $(\pi_j)_{j \in [k]}$ are independent, so are the $(t_{i,j,d})_{j \in [k], i \in [w']}$ for any fixed $d \in [k]$. Further note that $\forall d \in [k], \mu_d \coloneqq \mathbb{E}(\sum_{j \in k, i \in [w']} t_{i,j,d}) = \sum_{j \in k, i \in [w']} \mathbb{E}(t_{i,j,d}) = k \cdot \frac{\sum_{i \in [w]} t_i}{d}$. Therefore, by a Chernoff bound[3], for every $d \in [n]$, $\Pr(\sum_{j \in k} t_{i,j,d} > T) < 1/\lambda^{\omega(1)}$. By union-bound, $\Pr[\forall d \in [n], \forall i \in [w'], \sum_{j \in k} t_{i,j,d} \leq T] \leq n \cdot w' / \lambda^{\omega(1)} = 1/\lambda^{\omega(1)}$. □

Definition 21 (Two-Round Batch Computational Batch **SPIR** with Correlated "Mix and Match" Queries)**.** *A semi-honest two-round batch SPIR protocol with correlated "mix and match" queries between a sender and a receiver can be defined as a triple of PPT algorithms* $\mathsf{corrSPIR} = (\mathsf{corrSPIR}_R, \mathsf{corrSPIR}_S, \mathsf{corrSPIR}_D)$ *parameterised by a public* T-balanceable *"mix and match" function (definition 28)* $\mathsf{MixAndMatch}\colon \{0,1\}^w \to [N]^k$ *with the following syntax and properties:*

- **Syntax.**

  $\mathsf{corrSPIR}_R$: *On input the security parameter* 1^λ *and a vector of selection bits* $\vec{b} = (b_1, \ldots, b_w) \in \{0,1\}^w$, $\mathsf{corrSPIR}_R$ *outputs a receiver message* $\mathsf{spir}_R \in \{0,1\}^{\mathcal{O}(w)}$ *and an internal state* st*; without loss of generality, we assume* st *contains all the coins used by* $\mathsf{corrSPIR}_R$ *as well as* $\vec{b}$*.*

  $\mathsf{corrSPIR}_S$: *On input the security parameter* 1^λ*, a receiver message* spir_R*, and* k *N-bit databases* $\vec{m}_1, \ldots, \vec{m}_k \in \{0,1\}^N$, $\mathsf{corrSPIR}_S$ *outputs a sender message* $\mathsf{spir}_S \in \{0,1\}^{\mathcal{O}(k)}$*.*

  $\mathsf{corrSPIR}_D$: *On input a sender message* spir_S *and an internal state* st, $\mathsf{corrSPIR}_D$ *outputs a vector of messages* $(\widetilde{m}_1, \ldots, \widetilde{m}_k) \in \{0,1\}^k$*.*

- **Correctness.**

[3]Specifically, we are using the Chernoff bound in the form which is standardly denoted "$\Pr[X > (1+\delta)\mu] < \exp(-\delta^2 \mu(2+\delta))$".

$$\forall \vec{b} = (b_1, \ldots, b_w) \in \{0,1\}^w, \forall \vec{M} = (\vec{m}_1, \ldots, \vec{m}_k) \in \{0,1\}^{N \cdot k},$$

$$\Pr\left[(\widetilde{m}_1, \ldots, \widetilde{m}_k) = (\vec{m}_1[x_1], \ldots, \vec{m}_k[x_k]) : \begin{array}{rl} (\mathsf{spir}_R, \mathsf{st}) & \xleftarrow{\$} \mathsf{corrSPIR}_R(1^\lambda, \vec{b}) \\ \mathsf{spir}_S & \xleftarrow{\$} \mathsf{corrSPIR}_S(1^\lambda, \mathsf{spir}_R, \vec{M}) \\ (\widetilde{m}_1, \ldots, \widetilde{m}_k) & \xleftarrow{\$} \mathsf{corrSPIR}_D(\mathsf{spir}_S, \mathsf{st}) \end{array} \right] = 1$$

where $(x_1, \ldots, x_k) \coloneqq \mathsf{MixAndMatch}(\vec{b})$.

- **Security.**

  The following protocol, Π_{corrSPIR} *(fig. 6.1), securely realises* $\mathcal{F}_{\mathsf{corrSPIR}}$ *(fig. 7.7) in the presence of a semi-honest adversary. The receiver computes* $(\mathsf{spir}_R, \mathsf{st}) \xleftarrow{\$} \mathsf{corrSPIR}_R(1^\lambda, \vec{b})$ *and sends* spir_R *to the sender, who in turn computes* $\mathsf{spir}_S \xleftarrow{\$} \mathsf{corrSPIR}_S(1^\lambda, \mathsf{spir}_R, \vec{M})$ *and returns* spir_S*; finally, the receiver computes and outputs* $(\widetilde{m}_1, \ldots, \widetilde{m}_k) \xleftarrow{\$} \mathsf{corrSPIR}_D(\mathsf{spir}_S, \mathsf{st})$.

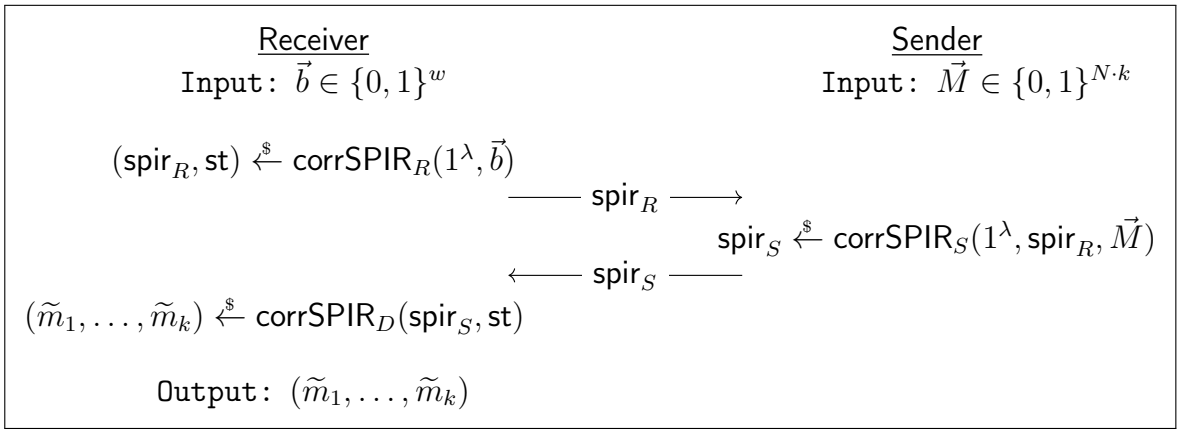


Figure 6.1: Two-Round corrSPIR Protocol Π_{corrSPIR}.

Functionality $\mathcal{F}_{\mathsf{corrSPIR}}$

The functionality $\mathcal{F}_{\mathsf{corrSPIR}}$ is parameterised by the number k of SPIRs in the batch, the size N of each database, and the number w of selection bits. Furthermore, it is parameterised by a public T-balanceable "mix and match" function (definition 28) $\mathsf{MixAndMatch}\colon \{0,1\}^w \to [N]^k$. $\mathcal{F}_{\mathsf{corrSPIR}}$ interacts with an ideal sender $\mathbf{S}$ and an ideal receiver $\mathbf{R}$ via the following queries.

1. On input $(\texttt{sender}, \vec{M} = (\vec{m}_i)_{i \in [k]})$ from $\mathbf{S}$, with $\vec{m}_i = (m_{i,j})_{j \in [N]} \in \{0,1\}^N$ store $\vec{M}$.
2. On input $(\texttt{receiver}, (b_j)_{j \in [w]})$ from $\mathbf{R}$, check if a tuple of inputs $\vec{M}$ has already been recorded; if so, compute $(x_1, \ldots, x_k) \coloneqq \mathsf{MixAndMatch}(b_1, \ldots, b_w) \in [N]^k$, send $(m_{i,x_i})_{i \in [k]}$ to $\mathbf{R}$, and halt.

If the functionality receives an incorrectly formatted input, it aborts.

Figure 6.2: Ideal Functionality $\mathcal{F}_{\mathsf{corrSPIR}}$ for Batch SPIR with Correlated "Mix and Match" Queries

6.3 Sublinear-Communication Secure Computation from Correlated SPIR

6.3.1 Sublinear Computation of $\log\log$-Depth Circuits from corrSPIR

In this section theorem 10 shows how to build sublinear secure computation for shallow (roughly $\log\log$-depth) circuits from $\mathsf{corrSPIR}$, with an explicit protocol provided in fig. 6.3. Main theorem 3 combines all of the previous theorems and shows that sublinear secure computation for shallow circuits can be based on $\mathsf{QR} + \mathsf{LPN}$.

Protocol Π_{2PC}

Functionality:

- **Parameters:** $C\colon \{0,1\}^n \to \{0,1\}^m$ is a boolean circuit of depth k. For $j \in [m]$, $S_j = \{s_{j,1}, \ldots, s_{j,2^k}\}$ is the subset[a] of the inputs on which depends the j^{th} output of f, and for $i \in [n]$ we denote t_i the number of outputs of C on which the i^{th} variable depends. $(\pi_j)_{j\in[m]} \in (\mathfrak{S}_{2^k})^m$ is a family of m permutations on $[2^k]$, such that the following is a $(T = \mathsf{polylog}(n))$-balanced "mix and match" function:

$$\begin{array}{rccc} \mathsf{MixAndMatch}_C\colon & \{0,1\}^w & \to & [2^k]^m \\ & (x_1, \ldots, x_w) & \mapsto & (x_{s_{j,\pi_j(1)}} \| \ldots \| x_{s_{j,\pi_j(2^k)}})_{j\in[m]} \end{array}$$

- **Inputs:** Parties P_0 and P_1 hold additive shares $(\vec{x}_0, \vec{x}_1)$ of an input $\vec{x} \in \{0,1\}^n$.
- **Outputs:** The parties output $C(\vec{x})$.
- **Requires:** $\mathsf{corrSPIR} = (\mathsf{corrSPIR}_R, \mathsf{corrSPIR}_S, \mathsf{corrSPIR}_D)$ is a two-round batch SPIR protocol with correlated "mix and match" queries.

Protocol:

1. P_0 samples $\vec{y}_0 \xleftarrow{\$} \{0,1\}^m$ and for $j \in [m]$ sets $\mathtt{DB}_j \in \{0,1\}^{2^{2^k}}$ to be the truth table of the following function:

$$\begin{array}{rccc} g_j\colon & \{0,1\}^{2^k} & \to & \{0,1\} \\ & (X_1, \ldots, X_{2^k}) & \mapsto & C_j((X_{\pi_j(1)} \oplus \vec{x}_0[\pi_j(1)] \| \ldots \| X_{\pi_j(2^k)} \oplus \vec{x}_0[\pi_j(2^k)])) \oplus \vec{y}_0[j] \end{array}$$

where C_j is the j^{th} output of C.

2. P_1 sets $\vec{x}'_1 \leftarrow (\vec{x}_1[1])^{\|\lceil t'_1/T\rceil} \| \ldots \| b_w^{\|\lceil t'_w/T\rceil}$.
   // $\vec{b'}$ is the vector whose first $\lceil t'_1/T\rceil$ coordinates are equal to b_1, followed by $\lceil t'_2/T\rceil$ coordinates equal to b_2, and so on. Note that the total size of this "selection vector with redundancies" is $\sum_{i=1}^{w} \lceil t'_i/T\rceil$.
3. P_1 samples $(\mathsf{spir}_R, \mathsf{st}) \xleftarrow{\$} \mathsf{corrSPIR}_R(1^\lambda, \vec{x}_1)$ and sends spir_R to P_0.
4. P_0 samples $\mathsf{spir}_S \xleftarrow{\$} \mathsf{corrSPIR}_S(1^\lambda, \mathsf{spir}_R, (\mathtt{DB}_j)_{j\in[m]})$ and sends $(\mathsf{spir}_S, \vec{y}_0)$ to P_1.

5. P_1 recovers $\vec{y}_1 \leftarrow \mathsf{corrSPIR}_D(\mathsf{spir}_S, \mathsf{st})$.

6. P_1 sets $\vec{y} \leftarrow \vec{y}_0 \oplus \vec{y}_1$, and sends $\vec{y}$ to P_0.

7. Each party P_σ outputs $\vec{y}$.

[a]Because C has depth k and each of its gate has fan-in at most 2, each output value only depends on at most 2^k inputs. Without loss of generality we can assume each output depends on exactly 2^k (by allowing for trivial "dependencies").

Figure 6.3: **Secure Computation of Low-Depth Circuits from** corrSPIR

Theorem 10. *If* corrSPIR *is a two-round batch* SPIR *protocol with correlated "mix and match" queries, then* Π_{2PC} *from fig. 6.3 securely computes the randomized functionality* $(\vec{x}_0, \vec{x}_1) \mapsto \{(\vec{r}, C(\vec{x}_0 \oplus \vec{x}_1) \oplus \vec{r}) : \vec{r} \xleftarrow{\$} \{0,1\}^m\}$ *in the presence of a semi-honest adversary corrupting (at most) one of the two parties.*

Proof. First note that by lemma 8, there indeed exists a family of permutations $(\pi_j)_{j \in [k]}$ such as the one required by Π_{2PC} (and that such a family can be found in expected constant time by simply sampling random permutations and testing for the "T-balanced" property). Therefore the protocol is well-defined. Correctness follows from correctness of corrSPIR, with the observation that if $(\alpha_1, \ldots, \alpha_k) \coloneqq \mathsf{MixAndMatch}_C(\vec{x}_1)$ then by construction $\mathtt{DB}_j[\alpha_j] = C_j(\vec{x}_1 \oplus \vec{x}_0) \oplus \vec{y}_0[j]$. It follows that $(\mathtt{DB}_1[\alpha_1], \ldots, \mathtt{DB}_m[\alpha_m]) \oplus \vec{y}_0 = C(\vec{x}_1 \oplus \vec{x}_0)$. Π_{2PC} essentially consists in a single call to Π_{corrSPIR}, and security follows from the security of corrSPIR via a standard hybrid argument. □

Our first main theorem follows from the combination of corollary 7 (which instantiates dec-OT from QR + LPN), lemma 9 (which provides a construction of rep-OT from dec-OT), theorem 11 (which provides a construction of corrSPIR from rep-OT), and theorem 10 (which provides a secure computation protocol from corrSPIR).

Main Theorem 3 (Sublinear Secure Computation from QR + LPN)**.** *Assume the* QR *assumption and the binary LPN assumption* $\mathsf{LPN}(\mathsf{dim}, \mathsf{num}, \rho)$ *with dimension* $\mathsf{dim} = \mathsf{poly}(\lambda)$*, number of samples* $\mathsf{num} = (n+m)^{1/3} \cdot \mathsf{poly}(\lambda)$*, and noise rate* $\rho = \mathsf{num}^{\varepsilon - 1}$ *(for some constant* $\varepsilon < 1$*). Then for any* n*-input* m*-output boolean circuit* C *of size* s *and depth* k*, there is a two-party protocol for securely computing* C *using only* $\mathcal{O}(n + m + 2^{k+2^k} \cdot \mathsf{polylog}(n) \cdot \mathsf{poly}(\lambda) \cdot ((n+m)^{2/3} + (n+m)^{(1+2\varepsilon)/3}))$ *bits of communication, and computation* $\mathsf{poly}(\lambda, 2^{2^k})$.

The numbers are obtained by setting $\mathsf{dim} = \mathsf{poly}(\lambda)$, $\ell = (n+m)^{1/3}$, and $\mathsf{num} = \ell^2 \cdot \mathsf{dim}$ in the statement of corollary 7. The $\mathsf{polylog}(n)$ term stems from the bounded query repetition property and the 2^{2^k+k} stems from the fact that the batch computational SPIR does, in essence, apply $\approx 2^{2^k}$ instances of the sender OT function in each interative compression step, and there are 2^k such steps. For example, absorbing the $\mathsf{polylog}(n)$ term in the $\mathsf{poly}(\lambda)$ term and setting $\varepsilon = 1/2$, and $k = (\log\log s)/4$ (implying $2^{k+2^k} \ll \sqrt{s}$), we get corollary 4:

Corollary 4 (Sublinear Secure Computation of log log-Depth Circuits)**.** *Assume the* QR *assumption and the binary LPN assumption* $\mathsf{LPN}(\mathsf{dim}, \mathsf{num}, \rho)$ *with dimension* $\mathsf{dim} = \mathsf{poly}(\lambda)$*, number of samples* $\mathsf{num} = (n+m)^{1/3} \cdot \mathsf{poly}(\lambda)$*, and noise rate* $\rho = \mathsf{num}^{-1/2}$*. Then for any* n*-input* m*-output boolean circuit* C *of polynomial size* s *and depth* $\log\log s/4$*, there is a two-party protocol for securely computing* C *using only* $\mathcal{O}(n + m + \sqrt{s} \cdot \mathsf{poly}(\lambda) \cdot (n+m)^{2/3})$ *bits of communication, and polynomial computation.*

6.3.2 Extension to Layered Circuits

Layered circuits are boolean circuits whose gates can be arranged into layers such that any wire connects adjacent layers. It is well-known from previous works [BGI16a, Cou19, CM21] that sublinear protocols for low-depth circuits translate to sublinear protocols for general layered circuits: the parties simply cut the layered circuit into low-depth "chunks", and securely evaluate it chunk-by-chunk. For each chunk, a sublinear secure protocol is invoked to compute the low-depth function which maps shares of the values on the first layer to shares of the values on the first layer of the next chunk. In particular, we get as a corollary from Theorem 3:

Corollary 5 (Sublinear Secure Computation of Layered Circuits)**.** *Assume the* QR *or* DDH *assumption. Then for any n-input m-output layered boolean circuit C of polynomial size s and depth d, and any k, assuming the binary LPN assumption* $\mathsf{LPN}(\mathsf{dim}, \mathsf{num}, \rho)$ *with dimension* $\mathsf{dim} = \mathsf{poly}(\lambda)$*, number of samples* $\mathsf{num} = ((s/d)^2/2^{2^k})^{1/3} \cdot \mathsf{poly}(\lambda)$*, and noise rate* $\rho = \mathsf{num}^{-1/2}$*, there is a two-party protocol for securely computing C using communication*

$$\mathcal{O}\left(n + m + \frac{d}{k} \cdot \left(2^{2^k} \cdot \frac{s}{d}\right)^{2/3} \cdot \mathsf{poly}(\lambda) + \frac{s}{k}\right),$$

and computation $\mathsf{poly}(\lambda, 2^{2^k})$*.*

In the above corollary, "layered" refers to layered circuits whose inputs are on the first layer; this type of layered circuit is sometimes called *synchronous* in the literature. Furthermore, the corollary also uses a slightly optimized choice of parameters (compared to a naive application of Theorem 3): we set ℓ such that $\ell^2 = 2^{2^k+k} \cdot \mathsf{num}$ and $\mathsf{num} = ((s/d)^2/2^{2^k})^{1/3} \cdot \mathsf{poly}(\lambda)$ in the statement of corollary 7. The above leads to a sublinear secure computation protocol for layered circuit whenever the circuit is not too "tall and skinny", i.e. , d is not too close to s. For example:

Corollary 6 ($s/\log\log s$-Secure Computation of Layered Circuits)**.** *Assume the* QR *or* DDH *assumption. Then for any n-input m-output layered boolean circuit C of polynomial size s and depth d, for any constant $\varepsilon \in (0, 1)$, assuming the binary LPN assumption* $\mathsf{LPN}(\mathsf{dim}, \mathsf{num}, \rho)$ *with dimension* $\mathsf{dim} = \mathsf{poly}(\lambda)$*, number of samples* $\mathsf{num} = ((s/d)^2/s^{\varepsilon})^{1/3} \cdot \mathsf{poly}(\lambda)$*, and noise rate* $\rho = \mathsf{num}^{-1/2}$*, there is a two-party protocol for securely computing C using communication*

$$\mathcal{O}\left(n + m + d^{1/3} \cdot s^{2(1+\varepsilon)/3} \cdot \mathsf{poly}(\lambda) + \frac{s}{\log\log s}\right),$$

and computation $\mathsf{poly}(\lambda)$*.*

The above is sublinear in s as soon as $d = o(s^{1-\varepsilon}/\mathsf{poly}(\lambda))$.

6.4 A "Generic" Construction from Decomposable Batch OT

6.4.1 Decomposable Two-Round Batch Oblivious Transfer

In this section, definition 22 defines the notion of *decomposable two-round batch oblivious transfer* ($\mathsf{dec\text{-}OT}$).

We introduce the notion of *decomposable two-round batch oblivious transfer*, which can be seen as a strengthening of two-round batch OT with constant upload-rate (i.e. the size of the receiver message is linear in the batch size k) and download-rate asymptotically one (i.e. the size of the sender message is $k + o(k)$). The differences boil down to a notion of *decomposability* which we impose on the sender message. At a high level, what we want to capture is the fact that the receiver should be able to retrieve the i^{th} selected message in the batch if and only it also has access to the i^{th} bit of the sender message (using its own internal state saved from generating the receiver message). More generally, given only a subset of the bits of the sender message, the receiver should able to retrieve the corresponding subset of selected messages in the batch. For our purposes, it will suffice to consider a relaxation of the notion we just described, and allow the sender message to have some small overhead rather than having a one-to-one correspondence between the bits on the sender message and the k selected messages. In this relaxed form, we require that the sender message be comprised of two parts: a "reusable" one (of size $o(k)$), and a "decomposable" one (of size k). On its own, the reusable part should reveal nothing about the messages, but can be used to "decode" each bit of the decomposable part so as to retrieve (exactly) the corresponding selected message in the batch. Among other benefits of this relaxation, it allows us to consider constructions whose download-rate is only asymptotically, and not necessarily exactly, optimal. We now formalize this notion in definition 22.

Definition 22 (Decomposable Two-Round Batch Oblivious Transfer)**.** *Let $k \in \mathbb{N}^\star$ and let $\alpha(\cdot)$ be a sublinear function (i.e. $\alpha(n) = o(n)$). A semi-honest two-round decomposable batch OT protocol with $\alpha(\cdot)$-overhead between a sender and a receiver is defined as a triple of PPT algorithms $\mathsf{dec\text{-}OT} = (\mathsf{dec\text{-}OTR}, \mathsf{dec\text{-}OTS}, \mathsf{dec\text{-}OTD})$ with the following syntax and properties:*

- **Syntax.**

  $\mathsf{dec\text{-}OTR}$: *On input the security parameter 1^λ and a vector of selection bits $\vec{b} = (b_1, \ldots, b_k) \in \{0,1\}^k$, $\mathsf{dec\text{-}OTR}$ outputs a receiver message $\mathsf{otr} \in \{0,1\}^{\mathcal{O}(k)}$ and an internal state st; without loss of generality we assume that st contains all the random coins used by $\mathsf{dec\text{-}OTR}$ as well as $\vec{b}$.*

  $\mathsf{dec\text{-}OTS}$: *On input the security parameter 1^λ, a receiver message otr, and a database of k pairs of bits $((m_0^{(i)}, m_1^{(i)}))_{i\in[k]} \in \{0,1\}^{2k}$, $\mathsf{dec\text{-}OTS}$ outputs a sender message $\mathsf{ots} = (\mathsf{ots}^\star, \mathsf{ots}^{\mathsf{dec}})$, which is comprised of a reusable part $\mathsf{ots}^\star \in \{0,1\}^{\alpha(k)}$ and a decomposable part $\mathsf{ots}^{\mathsf{dec}} \in \{0,1\}^k$.*

  $\mathsf{dec\text{-}OTD}$: *On input a batch subset $K \subseteq [k]$, a partial sender message $\mathsf{ots}' \in \{0,1\}^{\alpha(k)+|K|}$, and an internal state st, $\mathsf{dec\text{-}OTD}$ outputs a vector of messages $(\widetilde{m}_i)_{i\in K} \in \{0,1\}^{|K|}$.*

- **Decomposable Correctness.** *For every $\lambda \in \mathbb{N}^\star$, $K \subseteq [k]$, every $\vec{b} = (b_1, \ldots, b_k) \in \{0,1\}^k$, and every $\vec{m} = ((m_0^{(i)}, m_1^{(i)}))_{i\in[k]} \in \{0,1\}^{2k}$,*

$$\Pr\left[(\widetilde{m}_1, \ldots, \widetilde{m}_{|K|}) = (m_{b_i}^{(i)})_{i\in K} : \begin{array}{rcl} (\mathsf{otr}, \mathsf{st}) & \xleftarrow{\$} & \mathsf{dec\text{-}OTR}(1^\lambda, \vec{b}) \\ (\mathsf{ots}^\star, \mathsf{ots}^{DB}) & \xleftarrow{\$} & \mathsf{dec\text{-}OTS}(1^\lambda, \mathsf{otr}, \vec{m}) \\ (\widetilde{m}_1, \ldots, \widetilde{m}_{|K|}) & \xleftarrow{\$} & \mathsf{dec\text{-}OTD}(K, (\mathsf{ots}^\star, [\mathsf{ots}^{\mathsf{dec}}]_K), \mathsf{st}) \end{array}\right] = 1 \ .$$

- **Receiver Security (against Semi-Honest Sender).** *There exists an expected polynomial time simulator Sim_S such that for every $\lambda \in \mathbb{N}^\star$ and every*

$\vec{b} = (b_1, \ldots, b_k) \in \{0,1\}^k$,

$$\left\{\mathsf{otr} \colon (\mathsf{otr}, \mathsf{st}) \xleftarrow{\$} \mathsf{dec\text{-}OTR}(1^\lambda, \vec{b})\right\} \overset{\mathrm{c}}{\approx} \left\{\mathsf{Sim}_S(1^\lambda)\right\} .$$

- **Decomposable Sender Security (against Semi-Honest Receiver).** *There exists an expected polynomial time simulator* Sim_R *such that for every* $\lambda \in \mathbb{N}^\star$, *every* $K \subseteq [k]$, *every* $\vec{b} = (b_1, \ldots, b_k) \in \{0,1\}^k$, *and every* $\vec{m} = ((m_0^{(i)}, m_1^{(i)}))_{i \in [k]} \in \{0,1\}^{2k}$,

$$\left\{(\mathsf{ots}^\star, [\mathsf{ots}^{\mathsf{dec}}]_K, \mathsf{otr}, \mathsf{st}) \colon \begin{array}{rcl} (\mathsf{otr}, \mathsf{st}) & \xleftarrow{\$} & \mathsf{dec\text{-}OTR}(1^\lambda, \vec{b}) \\ (\mathsf{ots}^\star, \mathsf{ots}^{BD}) & \xleftarrow{\$} & \mathsf{dec\text{-}OTS}(1^\lambda, \mathsf{otr}, \vec{m}) \end{array}\right\} \overset{\mathrm{c}}{\approx}$$

$$\left\{(\mathsf{sim}^\star, \mathsf{sim}^{\mathsf{dec}}, \mathsf{otr}, \mathsf{st}) \colon \begin{array}{rcl} (\mathsf{otr}, \mathsf{st}) & \xleftarrow{\$} & \mathsf{dec\text{-}OTR}(1^\lambda, \vec{b}) \\ (\mathsf{sim}^\star, \mathsf{sim}^{\mathsf{dec}}) & \xleftarrow{\$} & \mathsf{Sim}_R(1^\lambda, K, (m_{b_i}^{(i)})_{i \in K}, \vec{b}, \mathsf{otr}, \mathsf{st}) \end{array}\right\} .$$

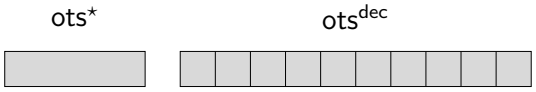
Note that in definition 22 the rate is baked into the syntax: dec-OTR outputs a receiver message of size $\mathcal{O}(k)$ and dec-OTS outputs a sender message of size $\alpha(k) + k$.

6.4.2 Bounded Query Repetitions

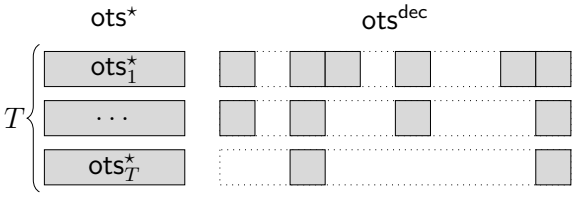
At a high level the goal of this section is to show how a receiver message of dec-OT can be re-used, possibly with imbalances in how many times each selection bit in the batch is re-used, while asymptotically preserving upload- and download-rate. definition 23 defines the notion of *decomposable two-round batch oblivious transfer with bounded query repetitions* (rep-OT), and lemma 9 establishes a generic reduction from dec-OT to rep-OT, with an explicit transformation in fig. 6.6.

A central property of two-round OT protocols is that the receiver message can be re-used multiple times by the sender on different (and even adaptively crafted) databases. If the receiver were to prepare messages for different selection bits, the sender could use each one a different number of times and still only have to send an answer whose size is proportional to the number of "useful" messages. When using two-round batch OT, the sender cannot freely re-use some of the selection bits more than the others: if the receiver sends a message corresponding to the selection vector $(b_1, \ldots, b_k)$, and the sender wishes to use each selection bit b_i a number t_i of times then it has little choice but to pad their database up to size $(\max_{i \in [k]} t_i) \times k$ using dummy messages then send $(\max_{i \in [k]} t_i)$ different sender messages. However, if the two-round batch OT is decomposable, then the sender can drop the bits corresponding to the dummy messages before sending them to the receiver. More specifically, if the two-round decomposable batch OT has α-overhead, then the size of the sender message is $(\max_{i \in [k]} t_i) \cdot \alpha + \sum_{i=1}^k t_i$. Since the amount of "useful" bits is $\sum_{i=1}^k t_i$, the download rate is still asymptotically one provided α is sufficiently small. This informal discussion is illustrated in fig. 6.4.

We introduce a stronger notion of two-round decomposable batch OT in definition 23, one which allows each selection bit to be used a different but bounded number of times while preserving the asymptotic rate. We provide in fig. 6.6 a black-box transformation which allows any two-round decomposable batch OT to gain this property. Before we proceed, let us observe that this transformation has an inherent limitation: because the size of the sender message grows with $(\max_{i \in [k]} t_i) \times \alpha$ the number of repetitions must be bounded by $\|t\|_\infty = o(k/\alpha)$. In particular, none of the t_i's can be linear in k (and skipping ahead, our application to sublinear computation will require us to bypass this restriction). In other words, while we can tolerate $\|\cdot\|_\infty$-bounded



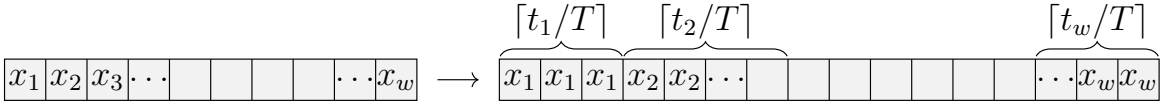
(a) The reusable part $\mathsf{ots}^\star$ is monolithic, while the decomposable part $\mathsf{ots}^{\mathsf{dec}}$ can be decomposed.



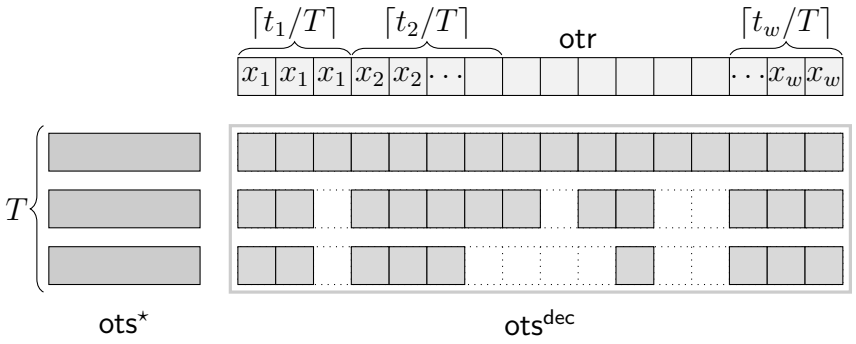
(b) Having invoked T parallel instances of $\mathsf{dec\text{-}OT}$, the reusable part is now T times larger, but each selection bit can be used up to T times without the size of the decomposable part blowing up by a factor of T.

Figure 6.4: Using decomposability to achieve $\|\cdot\|_\infty$-bounded query repetitions.

repetitions, we cannot tolerate $\|\cdot\|_1$-bounded repetitions; the difference boils down to the fact that we require the repetitions to be somewhat "balanced" in that no selection bit is solicited too much (e.g. a linear number of times). This can be addressed by having the receiver replicate the selection bits proportionally to the number of times the sender will want to use it. This way, we are reduced to the case where the repetitions are balanced: if each selection bit b_i is replicated across $\lceil t_i/T \rceil$ copies, then each copy of each selection bits is only used $\leq T$ times, which means the repetition vector is $\|\cdot\|_1$-bounded by T. In the parameter range of our application, T can be chosen so that $T \times \alpha$ is sublinear but the total number of copies sent by the receiver, which is $\sum_{i=1}^{w} \lceil t_i/T \rceil \leq \frac{1}{T}\|t\|_1 + w$, is less than $2w$. What this achieves is keeping the sender message small by making the receiver message only slightly larger. This transformation is illustrated in fig. 6.5 and implicitly used in the construction of fig. 6.8.



(a) The overall size of the receiver message with duplicated queries $\Sigma_{i=1}^{w} \lceil t_i/T \rceil \leq 2w$.



(b) If $\|t\|_1 = w$ we now have $|\mathsf{otr}| = \mathcal{O}(w)$ and $|\mathsf{ots}| = w \cdot (1 + o(1))$.

Figure 6.5: Going from $\|\cdot\|_\infty$-bounded repetitions to $\|\cdot\|_1$-bounded repetitions by doubling the size of the receiver message.

Definition 23 (Decomposable Two-Round Batch Oblivious Transfer with Bounded Query Repetitions)**.** *Let $k \in \mathbb{N}^\star$ and $\alpha = o(n)$. A semi-honest two-round decomposable batch OT protocol with $\alpha(\cdot)$-overhead and T-bounded query repetitions between*

a sender and a receiver can be defined as a triple of PPT algorithms $\mathsf{rep\text{-}OT} = (\mathsf{rep\text{-}OTR}, \mathsf{rep\text{-}OTS}, \mathsf{rep\text{-}OTD})$ *with the following syntax and properties:*

- **Syntax.**

  $\mathsf{rep\text{-}OTR}$: *On input the security parameter* 1^λ *and a vector of selection bits* $\vec{b} = (b_1, \ldots, b_k) \in \{0,1\}^k$, $\mathsf{rep\text{-}OTR}$ *outputs a receiver message* $\mathsf{otr} \in \{0,1\}^{\mathcal{O}(k)}$ *and an internal state* st*; without loss of generality we assume that* st *contains all the random coins used by* $\mathsf{rep\text{-}OTR}$ *as well as* $\vec{b}$*.*

  $\mathsf{rep\text{-}OTS}$: *On input the security parameter* 1^λ*, a query* otr*, a database* $((m_0^{(i)}, m_1^{(i)}))_{i\in[k']} \in \{0,1\}^{2k'}$ *(where* $k \le k' \le k \cdot T$*), and a vector of repetitions* $\mathsf{rep} = (\mathsf{rep}_1, \ldots, \mathsf{rep}_k) \in [0,T]^k$ *such that* $\sum_{i=1}^k \mathsf{rep}_i = k'$*,* $\mathsf{rep\text{-}OTS}$ *outputs a sender message* $\mathsf{ots} = (\mathsf{ots}^\star, \mathsf{ots}^{\mathsf{dec}})$*, which is comprised of a reusable part* $\mathsf{ots}^\star \in \{0,1\}^{\alpha(k)}$ *and a decomposable part* $\mathsf{ots}^{\mathsf{dec}} \in \{0,1\}^{k'}$*, as well as* rep*.*

  $\mathsf{rep\text{-}OTD}$: *On input a batch subset* $K \subseteq [k']$*, a partial sender message* $\mathsf{ots}' \in \{0,1\}^{\alpha(k)+|K|}$*, a vector of repetitions* $\mathsf{rep} = (\mathsf{rep}_1, \ldots, \mathsf{rep}_k) \in [0,T]^k$ *such that* $\sum_{i=1}^k \mathsf{rep}_i = k'$*, and an internal state* st*,* $\mathsf{rep\text{-}OTD}$ *outputs a vector of messages* $(\widetilde{m}_i)_{i\in K} \in \{0,1\}^{|K|}$*.*

- **Decomposable Correctness.** *For every* $\lambda \in \mathbb{N}^\star$*,* $K \subseteq [k']$*, every* $\vec{b} = (b_1, \ldots, b_k) \in \{0,1\}^k$*, and every* $\vec{m} = ((m_0^{(i)}, m_1^{(i)}))_{i\in[k']} \in \{0,1\}^{2k'}$*,*

$$\Pr\left[(\widetilde{m}_1, \ldots, \widetilde{m}_{|K|}) = (m_{\sigma_i}^{(i)})_{i\in K} \colon \begin{array}{rcl} (\mathsf{otr}, \mathsf{st}) & \xleftarrow{\$} & \mathsf{rep\text{-}OTR}(1^\lambda, \vec{b}) \\ ((\mathsf{ots}^\star, \mathsf{ots}^{\mathsf{dec}}), \mathsf{rep}) & \xleftarrow{\$} & \mathsf{rep\text{-}OTS}(1^\lambda, \mathsf{otr}, \vec{m}, \mathsf{rep}) \\ (\widetilde{m}_1, \ldots, \widetilde{m}_{|K|}) & \xleftarrow{\$} & \mathsf{rep\text{-}OTD}(K, (\mathsf{ots}^\star, [\mathsf{ots}^{\mathsf{dec}}]_K), \mathsf{rep}, \mathsf{st}) \end{array}\right] = 1 \ ,$$

$$\text{where } \sigma_i \coloneqq b_{\max\{j \colon (\sum_{j'<j} \mathsf{rep}_{j'}) \le i\}} \ .$$

- **Receiver Security (against Semi-Honest Sender).** *There exists an expected polynomial time simulator* Sim_S *such that for every* $\lambda \in \mathbb{N}^\star$ *and every* $\vec{b} = (b_1, \ldots, b_k) \in \{0,1\}^k$*,*

$$\left\{\mathsf{otr} \colon (\mathsf{otr}, \mathsf{st}) \xleftarrow{\$} \mathsf{rep\text{-}OTR}(1^\lambda, \vec{b})\right\} \overset{\mathrm{c}}{\approx} \left\{\mathsf{Sim}_S(1^\lambda)\right\} \ .$$

- **Decomposable Sender Security (against Semi-Honest Receiver).** *There exists an expected polynomial time simulator* Sim_R *such that for every* $\lambda \in \mathbb{N}^\star$*, every* $\mathsf{rep} = (\mathsf{rep}_1, \ldots, \mathsf{rep}_k) \in [0,T]^k$ *such that* $\|\mathsf{rep}\|_1 = k'$*, every* $K \subseteq [k']$*, every* $\vec{b} = (b_1, \ldots, b_k) \in \{0,1\}^k$*, and every* $\vec{m} = ((m_0^{(i)}, m_1^{(i)}))_{i\in[k']} \in \{0,1\}^{2k'}$*,*

$$\left\{(\mathsf{ots}^\star, [\mathsf{ots}^{\mathsf{dec}}]_K, \mathsf{otr}, \mathsf{st}) \colon \begin{array}{rcl} (\mathsf{otr}, \mathsf{st}) & \xleftarrow{\$} & \mathsf{rep\text{-}OTR}(1^\lambda, \vec{b}) \\ (\mathsf{ots}^\star, \mathsf{ots}^{\mathsf{dec}}) & \xleftarrow{\$} & \mathsf{rep\text{-}OTS}(1^\lambda, \mathsf{otr}, \vec{m}, \mathsf{rep}) \end{array}\right\} \overset{\mathrm{c}}{\approx}$$

$$\left\{(\mathsf{sim}^\star, \mathsf{sim}^{\mathsf{dec}}, \mathsf{otr}, \mathsf{st}) \colon \begin{array}{rcl} (\mathsf{otr}, \mathsf{st}) & \xleftarrow{\$} & \mathsf{rep\text{-}OTR}(1^\lambda, \vec{b}) \\ (\mathsf{sim}^\star, \mathsf{sim}^{\mathsf{dec}}) & \xleftarrow{\$} & \mathsf{Sim}_R(1^\lambda, K, (m_{\sigma_i}^{(i)})_{i\in K}, \vec{b}, \mathsf{rep}, \mathsf{otr}, \mathsf{st}) \end{array}\right\}$$

$$\text{where } \sigma_i \coloneqq b_{\max\{j \colon (\sum_{j'<j} \mathsf{rep}_{j'}) \le i\}} \ .$$

Decomposable Two-Round Batch Oblivious Transfer with Bounded Query Repetition

Parameters: Batch number k, Repetition bound T.
Requires: A two-round decomposable batch $\mathsf{dec\text{-}OT}$ protocol $\mathsf{dec\text{-}OT} = (\mathsf{dec\text{-}OTR}, \mathsf{dec\text{-}OTS}, \mathsf{dec\text{-}OTD})$ with α-overhead such that $\alpha(k) = o(k/T)$.

$\mathsf{rep\text{-}OTR}$**:** On input the security parameter 1^λ and a vector of selection bits $\vec{b} = (b_1, \ldots, b_k) \in \{0,1\}^k$:

1. Compute $(\mathsf{otr}, \mathsf{st}) \xleftarrow{\$} \mathsf{dec\text{-}OTR}(1^\lambda, \vec{b})$.
2. Output $(\mathsf{otr}, \mathsf{st})$.

$\mathsf{rep\text{-}OTS}$**:** On input the security parameter 1^λ, a receiver message otr, a database $\vec{m} \in \{0,1\}^{2k'}$, and a vector of repetitions $\mathsf{rep} = (\mathsf{rep}_1, \ldots, \mathsf{rep}_k) \in [0,T]^k$ such that $\|\mathsf{rep}\|_1 = k'$:

1. Parse $\vec{m}$ as $((m_0^{(j,i)}, m_1^{(j,i)}))_{j\in[k], i\in[\mathsf{rep}_j]}$.
   // The first rep_1 pairs are indexed by $j = 1$, the next rep_2 by $j = 2$, and so on.
2. For $j \in [k]$ and $i \in [\mathsf{rep}_j + 1, T]$, set $(m_0^{(j,i)}, m_1^{(j,i)}) \leftarrow (0,0)$.
   // The database is padded with "dummy" elements so there are exactly T pairs associated with each $j \in [k]$.
3. For $i \in [T]$ set $\vec{m}_i \coloneqq ((m_0^{(j,i)}, m_1^{(j,i)}))_{j\in[k]}$.
   // Each "sub-database" $\vec{m}_i$ contains the i^{th} pair associated with each $j \in [k]$.
4. For $j = 1 \ldots T$:

   Compute $(\mathsf{ot}^\star_{S,j}, \mathsf{ot}^{\mathsf{dec}}_{S,j}) \xleftarrow{\$} \mathsf{dec\text{-}OTS}(1^\lambda, \mathsf{otr}, \vec{m}_j)$.
5. Set $\mathsf{ots}^\star \leftarrow \mathsf{ot}^\star_{S,1} \| \ldots \| \mathsf{ot}^\star_{S,T}$ and $\mathsf{ot}^{\mathsf{dec}}_S \leftarrow ([\mathsf{ot}^{\mathsf{dec}}_{S,1}]_1 \| \ldots \| [\mathsf{ot}^{\mathsf{dec}}_{S,\mathsf{rep}_1}]_1) \| \ldots \| ([\mathsf{ot}^{\mathsf{dec}}_{S,1}]_k \| \ldots \| [\mathsf{ot}^{\mathsf{dec}}_{S,\mathsf{rep}_k}]_k)$.
   // Filter out the bits of the decomposable parts associated with dummy elements, and reorder the remaining bits according to the original database.
6. Output $(\mathsf{ot}^\star_S, \mathsf{ot}^{\mathsf{dec}}_S)$.

$\mathsf{rep\text{-}OTD}$**:** On input a sender message ots, a vector of repetitions $\mathsf{rep} = (\mathsf{rep}_1, \ldots, \mathsf{rep}_k) \in [0,T]^k$ such that $\|\mathsf{rep}\|_1 = k'$, and an internal state st:

1. Parse ots as $(\mathsf{ot}^\star_S, \mathsf{ot}^{\mathsf{dec}}_S)$.
2. Parse $\mathsf{ot}^\star_S$ as $\mathsf{ot}^\star_{S,1} \| \ldots \| \mathsf{ot}^\star_{S,T}$.
3. Parse $\mathsf{ot}^{\mathsf{dec}}_S$ as $([\mathsf{ot}^{\mathsf{dec}}_{S,1}]_1 \| \ldots \| [\mathsf{ot}^{\mathsf{dec}}_{S,\mathsf{rep}_1}]_1) \| \ldots \| ([\mathsf{ot}^{\mathsf{dec}}_{S,1}]_k \| \ldots \| [\mathsf{ot}^{\mathsf{dec}}_{S,\mathsf{rep}_k}]_k)$.
4. For $i = 1 \ldots T$:
   (a) Set $K_i \coloneqq \{j \colon j \in [k], \mathsf{rep}_j \le i\}$, ordered according to the natural order on $\mathbb{N}$.

(b) Set $\vec{v}_i \leftarrow \|_{j \in K_i} [\mathsf{ot}^{\mathsf{dec}}_{S,i}]_j$.

(c) Compute $(\widetilde{m}_{i,j})_{j \in K_i} \xleftarrow{\$} \mathsf{dec\text{-}OTD}(K_i, (\mathsf{ot}^{\star}_{S,i}, v_i), \mathsf{st})$.
//Note that the size-$|K_j|$ vector output by dec-OTD is indexed here by elements of K_i, not by $1, \ldots, |K_i|$.

5. Output $(\widetilde{m}_{1,1}, \ldots, \widetilde{m}_{\mathsf{rep}_1,1}, \widetilde{m}_{1,2}, \ldots, \widetilde{m}_{\mathsf{rep}_2,2}, \ldots, \widetilde{m}_{\mathsf{rep}_k,k})$.

Figure 6.6: From dec-OT with α overhead to rep-OT with $\alpha \cdot T$ overhead.

Lemma 9 (From dec-OT to rep-OT)**.** *If* dec-OT *is a semi-honest two-round decomposable batch OT protocol with α overhead, then the construction* rep-OT *from fig. 6.6 is a semi-honest two-round decomposable batch OT protocol with $\alpha \cdot T$ overhead and T-bounded repetitions.*

Proof.

- *Syntax, Size, and Correctness:* The fact that rep-OT fulfills the syntactic requirements as well as correctness follows from inspection. In particular, note that the receiver message output by rep-OTR has indeed size $\mathcal{O}(k)$ and that the sender message output by rep-OTS is indeed comprised of a reusable part of size $T \cdot \alpha(k)$ and a decomposable part of size k'.

- *Receiver Security:* By receiver security of dec-OT there exists an expected polynomial time simulator Sim_S such that for every $\lambda \in \mathbb{N}^{\star}$ and every $\vec{b} = (b_1, \ldots, b_k) \in \{0,1\}^k$,
$$\left\{\mathsf{otr} \colon (\mathsf{otr}, \mathsf{st}) \xleftarrow{\$} \mathsf{dec\text{-}OTR}(1^\lambda, \vec{b})\right\} \overset{\mathrm{c}}{\approx} \mathsf{Sim}_S(1^\lambda) \ .$$
By construction rep-OTR = dec-OTR, so we also have that
$$\left\{\mathsf{otr} \colon (\mathsf{otr}, \mathsf{st}) \xleftarrow{\$} \mathsf{rep\text{-}OTR}(1^\lambda, \vec{b})\right\} \overset{\mathrm{c}}{\approx} \mathsf{Sim}_S(1^\lambda) \ ,$$
which concludes this part of the proof.

- *Sender Security:* By sender security of dec-OT there exists an expected polynomial time simulator $\mathsf{Sim}^{\mathsf{dec}}_R$ such that for every $\lambda \in \mathbb{N}^{\star}$, every $K \subseteq [k]$, every $\vec{b} = (b_1, \ldots, b_k) \in \{0,1\}^k$, and every $\vec{m} = ((m_0^{(i)}, m_1^{(i)}))_{i \in [k]} \in \{0,1\}^{2k}$,
$$\left\{(\mathsf{ots}^{\star}, [\mathsf{ots}^{\mathsf{dec}}]_K, \mathsf{otr}, \mathsf{st}) \colon \begin{array}{rcl}(\mathsf{otr}, \mathsf{st}) & \xleftarrow{\$} & \mathsf{dec\text{-}OTR}(1^\lambda, \vec{b}) \\ (\mathsf{ots}^{\star}, \mathsf{ots}^{\mathsf{BD}}) & \xleftarrow{\$} & \mathsf{dec\text{-}OTS}(1^\lambda, \mathsf{otr}, \vec{m})\end{array}\right\} \overset{\mathrm{c}}{\approx}$$
$$\left\{(\mathsf{sim}^{\star}, \mathsf{sim}^{\mathsf{dec}}, \mathsf{otr}, \mathsf{st}) \colon \begin{array}{rcl}(\mathsf{otr}, \mathsf{st}) & \xleftarrow{\$} & \mathsf{dec\text{-}OTR}(1^\lambda, \vec{b}) \\ (\mathsf{sim}^{\star}, \mathsf{sim}^{\mathsf{dec}}) & \xleftarrow{\$} & \mathsf{Sim}^{\mathsf{dec}}_R(1^\lambda, K, (m^{(i)}_{b_i})_{i \in K}, \vec{b}, \mathsf{otr}, \mathsf{st})\end{array}\right\}. \tag{6.1}$$
The construction of rep-OTS makes T parallel calls to dec-OTS. Sender security follows from a straightforward hybrid argument, replacing these calls one by one

with $\mathsf{Sim}_R^{\mathsf{dec}}$; indistinguishability of these hybrids follows by invoking eq. (6.1) once at each step (and therefore a polynomial number overall).

In a bit more detail, let $\mathsf{rep} = (\mathsf{rep}_1, \ldots, \mathsf{rep}_k) \in [0,T]^k$ such that $\sum_{i=1}^k \mathsf{rep}_i = k'$, and consider the following family of simulators $(\mathsf{Sim}_{R,t}^{\mathsf{rep}})_{t\in[0,T]}$ (fig. 6.7).

Simulator $\mathsf{Sim}_{R,t}^{\mathsf{rep}}$

Parameters: For $j \in [k]$, $\mathsf{ind}(j) := \sum_{j'<j} \mathsf{rep}_{j'}$. We define $\mathsf{First}_t := \bigcup_{j=1}^k [\mathsf{ind}(j),\ \mathsf{ind}(j) + \min(\mathsf{rep}_j, t-1)]$. For $i \in [k']$, $\sigma_i := b_{\max\{j:\ (\sum_{j'<j} \mathsf{rep}_{j'}) \le i\}}$.

On input $(1^\lambda, K \subseteq [k'], (m_{\sigma_i}^{(i)})_{i\in K\cap \mathsf{First}_t}, (m_0^{(i)}, m_1^{(i)})_{i\in[k']\setminus\mathsf{First}_t}, \mathsf{otr}, \mathsf{st})$ where :
// The datase is comprised of rep_j pairs "to be selected according to b_j", for each $j \in [k]$. For each j, the simulator is only given the selected message from the first $\min(\mathsf{rep}_j, t)$ pairs, but is given both messages (a "complete pair") from the other $\min(\mathsf{rep}_j, t) - t$ pairs.

1. Parse the family $(m_{\sigma_i}^{(i)})_{i\in K\cap\mathsf{First}_t}$ as $(m_{\sigma_{j,i}}^{(j,i)})_{(j,i)\in K'}$ where K' is the subset of $\{(j,i)\colon j \in [k], i \in [\mathsf{rep}_j]\}$ defined by $(j,i) \in K' \Leftrightarrow ((\sum_{j'<j} \mathsf{rep}_j) + i) \in K \cap \mathsf{First}_t$.
   // Each $i \in [k']$ can uniquely be associated with a pair (j,i) where $j \in [k]$ and $i \in [\mathsf{rep}_j]$. The simulator computes this re-indexing for the database elements for which it does not have a complete pair...
2. Parse the family $(m_0^{(i)}, m_1^{(i)})_{i\in[k']\setminus\mathsf{First}_t}$ as $((m_0^{(j,i)}, m_1^{(j,i)}))_{j\in[k],i\in[t+1,rep_j]}$ where $m_b^{(j,i)} := m_b^{(\mathsf{ind}(j)+i)}$.
   // ...and for those where he is given a complete pair.
3. Complete the family $(m_{\sigma_{j,i}}^{(j,i)})_{(j,i)\in K'}$ as $((m_0^{(j,i)}, m_1^{(j,i)}))_{j\in[k],i\in[1,t]}$, where $m_{1-\sigma_{j,i}}^{(j,i)} := 0$ for $(j,i) \in K'$ and $(m_0^{(i)}, m_1^{(i)}) := (0,0)$ for $(j,i) \notin K'$.
4. For $j \in [k]$ and $i \in [\mathsf{rep}_j + 1, T]$, set $(m_0^{(j,i)}, m_1^{(j,i)}) \leftarrow (0,0)$.
   // This "padding" of the database is also performed in the real execution and is not due to the simulator not being given the entire database.
5. For $i \in [T]$ set $\vec{m}_i := ((m_0^{(j,i)}, m_1^{(j,i)}))_{j\in[k]}$.
6. For $i = 1 \ldots t$:
   Compute $(\mathsf{sim}_i^\star, \mathsf{sim}_i^{\mathsf{dec}}) \xleftarrow{\$} \mathsf{Sim}_R^{\mathsf{dec}}(1^\lambda, [T], \vec{m}_i, \vec{b}, \mathsf{otr}, \mathsf{st})$
7. For $i = (t+1) \ldots T$:
   Compute $(\mathsf{ot}_{S,i}^\star, \mathsf{ot}_{S,i}^{\mathsf{dec}}) \xleftarrow{\$} \mathsf{dec\text{-}OTS}(1^\lambda, \mathsf{otr}, \vec{m}_i)$.
8. Set $\mathsf{ots}^\star \leftarrow \mathsf{sim}_1^\star \| \ldots \| \mathsf{sim}_t^\star \| \mathsf{ot}_{S,t+1}^\star \| \ldots \| \mathsf{ot}_{S,T}^\star$.
9. Set $\mathsf{ot}_S^{\mathsf{dec}} \leftarrow \|_{j=1}^k \left([\mathsf{sim}_1^{\mathsf{dec}}]_j \| \ldots \| [\mathsf{sim}_{\min(t,\mathsf{rep}_j)}^{\mathsf{dec}}]_j \| [\mathsf{ot}_{S,\min(t,\mathsf{rep}_j)+1}^{\mathsf{dec}}]_j \| \ldots \| [\mathsf{ot}_{S,\mathsf{rep}_j}^{\mathsf{dec}}]_j \right)$.
10. Output $(\mathsf{ot}_S^\star, [\mathsf{ot}_S^{\mathsf{dec}}]_K)$.

Figure 6.7: Simulator $\mathsf{Sim}^{\mathsf{rep}}_{R,t}$ replacing the first t calls to dec-OTS by calls to $\mathsf{Sim}^{\mathsf{dec}}_{S}$.

Consider the following distributions:

$$\Delta^{\mathsf{real}} \coloneqq \left\{ (\mathsf{ots}^\star, [\mathsf{ots}^{\mathsf{dec}}]_K, \mathsf{otr}, \mathsf{st}) : \begin{array}{rcl} (\mathsf{otr}, \mathsf{st}) & \xleftarrow{\$} & \mathsf{rep\text{-}OTR}(1^\lambda, \vec{b}) \\ (\mathsf{ots}^\star, \mathsf{ots}^{\mathsf{dec}}) & \xleftarrow{\$} & \mathsf{rep\text{-}OTS}(1^\lambda, \mathsf{otr}, \vec{m}, \mathsf{rep}) \end{array} \right\}$$

$$\forall t \in [T],\ \Delta^{\mathsf{Sim}}_t \coloneqq \left\{ (\mathsf{sim}^\star, \mathsf{sim}^{\mathsf{dec}}, \mathsf{otr}, \mathsf{st}) : \begin{array}{rcl} (\mathsf{otr}, \mathsf{st}) & \xleftarrow{\$} & \mathsf{rep\text{-}OTR}(1^\lambda, \vec{b}) \\ (\mathsf{sim}^\star, \mathsf{sim}^{\mathsf{dec}}) & \xleftarrow{\$} & \mathsf{Sim}^{\mathsf{rep}}_{R,t}(\ 1^\lambda, K, (m^{(i)}_{\sigma_i})_{i \in K \cap \mathsf{First}_t}, \\ & & \qquad (m^{(i)}_0, m^{(i)}_1)_{i \in [k'] \setminus \mathsf{First}_t}, \vec{b}, \mathsf{rep}, \mathsf{otr}, \mathsf{st}) \end{array} \right\}$$

where $\sigma_i \coloneqq b_{\max\{j\,:\,(\sum_{j'<j} \mathsf{rep}_{j'}) \le i\}}$.

For all $t \in [T]$, it holds that $\Delta^{\mathsf{Sim}}_{t-1} \overset{\mathrm{c}}{\approx} \Delta^{\mathsf{Sim}}_t$ by decomposable sender security of dec-OT. Indeed, if a PPT $\mathcal{D}$ could distinguish $\Delta^{\mathsf{Sim}}_{t-1} \overset{\mathrm{c}}{\approx} \Delta^{\mathsf{Sim}}_t$ with probability ϵ then it could distinguish the following distributions with the same probability:

$$\left\{ (\mathsf{ots}^\star, [\mathsf{ots}^{\mathsf{dec}}]_K, \mathsf{otr}, \mathsf{st}) : \begin{array}{rcl} (\mathsf{otr}, \mathsf{st}) & \xleftarrow{\$} & \mathsf{dec\text{-}OTR}(1^\lambda, \vec{b}) \\ (\mathsf{ots}^\star, \mathsf{ots}^{\mathsf{BD}}) & \xleftarrow{\$} & \mathsf{dec\text{-}OTS}(1^\lambda, \mathsf{otr}, \vec{m}_t) \end{array} \right\}$$

where $\vec{m}_t$ is defined in line 6. of $\mathsf{Sim}^{\mathsf{rep}}_{R,t}$

and

$$\left\{ (\mathsf{sim}^\star, \mathsf{sim}^{\mathsf{dec}}, \mathsf{otr}, \mathsf{st}) : \begin{array}{rcl} (\mathsf{otr}, \mathsf{st}) & \xleftarrow{\$} & \mathsf{dec\text{-}OTR}(1^\lambda, \vec{b}) \\ (\mathsf{sim}^\star, \mathsf{sim}^{\mathsf{dec}}) & \xleftarrow{\$} & \mathsf{Sim}^{\mathsf{dec}}_{R}(1^\lambda, K, (m^{(i)}_{\sigma_i})_{i \in K \cap [\mathsf{ind}(j), \mathsf{ind}(j) + \min(t-1, \mathsf{rep}_j)]}, \vec{b}, \mathsf{otr}, \mathsf{st}) \end{array} \right\}.$$

But by eq. (6.1), they are computationally indistinguishable and therefore ϵ must be negligible.

Observe that $\Delta^{\mathsf{real}} = \Delta^{\mathsf{Sim}}_0 \overset{\mathrm{c}}{\approx} \ldots \overset{\mathrm{c}}{\approx} \Delta^{\mathsf{Sim}}_T$. Since $[k'] \setminus \mathsf{First}_T = \varnothing$, $\mathsf{Sim}^{\mathsf{rep}}_{R,T}$ only takes as input $(1^\lambda, K, (m^{(i)}_{\sigma_i})_{i \in K}, \vec{b}, \mathsf{rep}, \mathsf{otr}, \mathsf{st})$ and therefore we have shown our construction has decomposable sender security.

□

6.4.3 Two-Round Batch SPIR with Correlated Queries from Two-Round Decomposable Batch OT (with Bounded Query Repetitions)

In this section, definition 28 defines *"mix-and-match functions"* and lemma 8 shows how they can be built. Definition 21 introduces the notion of *two-round batch SPIR protocol with correlated "mix and match" queries* (corrSPIR), and theorem 11 provides a reduction from rep-OT to corrSPIR, with an explicit transformation in fig. 6.8.

We next introduce and achieve a notion of batch symmetric PIR with correlated queries. This corresponds to batch SPIR where the queries are not independent; rather, the total entropy w used to describe the queries is small, and the queried indices can

be reconstructed via a public function that "mixes and matches" the individual bits of entropy in a public manner. In more detail, if the w bits of entropy are $\alpha_1, \ldots, \alpha_w$, "mixing and matching" means that each of the $(n = \log N)$-bit queries to a single database can be obtained by concatenating n of the bits α_i, possibly permuted. In the notation below, the j^{th} query is given by vector $(\alpha_{s_{j,1}}, \ldots, \alpha_{s_{j,n}})$ (in other words, the j^{th} query is associated with the ordered subset $S_j = \{s_{j,1}, \ldots, s_{j,n}\}$ of the bits of entropy). This notion is tailor-made for our application to sublinear computation, but may be of independent interest. Let us now sketch the construction, and highlight both the need for decomposability and why we need the queries to be correlated.

Our starting point is the observation originally present in [KO97] (and later re-used explicitly in [IP07, DGI+19]) using the following Merkle tree abstraction that a rate-1 two-round 1-out-of-2 string OT can be seen as a hash function with a compression factor of two, and can be used to build (block) symmetric PIR. Let us sketch the construction under the (idealised) assumption that we have access to a rate-1 two-round 1-out-of-2 bit OT primitive, which is better suited to our purposes than its string variant. Suppose the server holds a database of $N = 2^n$ bits and that the client wants to retrieve the element stored at index $x = (x_1, \ldots, x_n)$. If the client sends a receiver message $\mathsf{otr}_1 \leftarrow \mathsf{OTR}(x_1)$ for the first bit of the desired index, the server can take the database, pair up elements whose indices differ only on the first bit, then apply the "hash function" $\mathsf{OTS}(\mathsf{otr}_1, \cdot)$ in order to retrieve a single-bit "hashed value" for each pair. If the server were to send all $N/2$ "hashes", the client could retrieve exactly the elements of the database whose indices start with x_1 by applying $\mathsf{OTD}(\mathsf{st}_1, \cdot)$ (st_1 was generated alongside otr_1). If instead the client sends receiver messages $(\mathsf{otr}_1, \ldots, \mathsf{otr}_n)$, one for each bit of the desired index, the server can now iteratively compress the database down to a single bit by building a "Merkle tree" using $\mathsf{OTS}(\mathsf{otr}_d, \cdot)$ at every node of depth $n - d$. If the server sends this single-bit root of the tree, the client can retrieve the element at index $x_1 \ldots x_n$ by iteratively applying $\mathsf{OTD}(\mathsf{st}_d, \cdot)$ for $d = n, \ldots, 1$. In fact, this is the only element of the database which the client can recover from the root of the Merkle tree (intuitively, when the server applied $\mathsf{OTS}(\mathsf{otr}_d, \cdot)$ they removed the client's ability to retrieve any information about elements whose indices have $1 - x_d$ in position d).

The above construction achieves SPIR with optimal communication, from the idealised primitive of rate-1 two-round 1-out-of-2 bit OT. We may ask whether we can replace this primitive with a more realistic batched version, and have the client send $\mathsf{BatchOTR}(x_1, \ldots, x_n)$ for instance in the hopes the client can batch the OT sender messages it has to compute. Unfortunately, while the server has to compute N OT sender messages with a first selection bit, then $N/2$ OT sender messages with a second, and so on, the messages at each layer are crafted adaptively and therefore cannot be batched.

Now consider the setting where the server holds a batch of k databases. If the sender is to compress each database down to a single bit using the "Merkle tree" approach, it has to compute $N/2^d$ OT sender messages for each layer of $d = 1, \ldots, n$ of each of the k Merkle trees. While messages across layers cannot be batched, OTs from the same layer of different trees can! The main challenge is that we can only afford (in order to keep communication low) to use a single batched receiver message in order to compute all of the sender messages. This requires a special assumption on the queries, which need to be highly correlated for this approach to work. We will be interested in

how many times a given α_i appears within the k queries (counted by the occurrence function t_i below), as well as how many times it appears in specific position $j' \in [n]$ within the k queries (denoted below by $t_{i,j'}$). If all $t_{i,j'}$ are bounded by T, then for each level $j' \in [n]$ in the "Merkle forest" we can achieve the desired length-halving compression by using at most T batch OT sender computations on the original batch OT selection vector $\vec{\alpha}$.

Batch SPIR with Correlated "Mix and Match" Queries

Parameters: k, N, $n := \log N$, w, T, a T-balanceable $\mathsf{MixAndMatch}\colon \{0,1\}^w \to [N]^k$ (parameterised by subsets $S_j = (s_{j,1}, \ldots, s_{j,n}) \in [w]^n$ for $j \in [k]$) and an associated list of number of occurrences $(t_1, \ldots, t_w)$ with $t_i = t_{i,1} + \cdots + t_{i,n}$, a two-round batch $\mathsf{rep\text{-}OT}$ protocol $\mathsf{rep\text{-}OT} = (\mathsf{rep\text{-}OTR}, \mathsf{rep\text{-}OTS}, \mathsf{rep\text{-}OTD})$.

$\mathsf{corrSPIR}_R$**:** On input the security parameter 1^λ and a vector of selection bits $\vec{b} = (b_1, \ldots, b_w) \in \{0,1\}^w$:

1. Set $\vec{b'} \leftarrow b_1^{\|\lceil t_1/T \rceil} \| \ldots \| b_w^{\|\lceil t_w/T \rceil}$.
   // $\vec{b'}$ is the vector whose first $\lceil t_1/T \rceil$ coordinates are equal to b_1, followed by $\lceil t_2/T \rceil$ coordinates equal to b_2, and so on. Note that the total size of this "selection vector with redundancies" is $\sum_{i=1}^{w} \lceil t_i/T \rceil$.
2. Compute $(\mathsf{spir}_R, \mathsf{st}) \xleftarrow{\$} \mathsf{OTR}(1^\lambda, \vec{b'})$, and output $(\mathsf{spir}_R, \mathsf{st} \| \vec{b})$.

$\mathsf{corrSPIR}_S$**:** On input the security parameter 1^λ, a receiver message spir_R, and k databases $\vec{m}_1, \ldots, \vec{m}_k \in [N]$:

1. Set $(\mathtt{DB}_{1,1}, \ldots, \mathtt{DB}_{1,k}) := (\vec{m}_1, \ldots, \vec{m}_k)$.
   // Throughout, $\mathtt{DB}_{d,k}$ will correspond to the values of the d^{th} layer of the k^{th} Merkle tree.
2. For $d = 1, \ldots, n$:
   (a) For $i = 1, \ldots, w$:
   $$\text{Set } \mathsf{rep}_{d,i} \leftarrow \begin{cases} T^{\|\lceil t_{i,d}/T \rceil} \| 0^{\|\lceil t_i/T \rceil - \lceil t_{i,d}/T \rceil} & \text{if } T | t_{i,d} \\ T^{\|\lfloor t_{i,d}/T \rfloor} \| t_{i,d} \% T \| 0^{\|\lceil t_i/T \rceil - \lceil t_{i,d}/T \rceil} & \text{if } T \nmid t_{i,d} \end{cases}$$
   (b) Set $\mathsf{rep}_d \leftarrow \mathsf{rep}_{d,1} \| \ldots \| \mathsf{rep}_{d,w}$.
   // Note that rep_d is a vector of size $\sum_{i=1}^{w} \lceil t_i/T \rceil$ with elements in $[0, T]$, and such that $\|\mathsf{rep}_d\|_1 = \sum_{i=1}^{w} t_{i,d}$.
   (c) Initialise $X_d \leftarrow \varnothing$.
   (d) For $j = 1, \ldots, k$:
   For $x = 0, \ldots, N/2^d - 1$:
   $X_d.\mathsf{append}(((\mathtt{DB}_{d,j}[2x], \mathtt{DB}_{d,j}[2x+1]),\ s_{j,d},\ x,\ j))$.
   // Note that X_d now contains $k \cdot N/2^d$ elements. For each $i \in [w]$, exactly $t_{i,d} \cdot N/2^d$ of the form $((\cdot,\cdot),\cdot,i,\cdot)$. Indeed, by definition, $t_{i,d} = |\{j \in [k] \colon s_{j,d} = i\}|$.
   (e) Sort X_d according to the lexicographic order which first sorts by increasing fourth element (the "$j \in [k]$") and then, in case of equality, by increasing third element (the "$x \in [0, N/2^d - 1]$").

(f) Greedily partition X_d as $X_d = X_{d,1} \sqcup \cdots \sqcup X_{d,(N/2^d)}$ such that for each $\ell \in [N/2^d]$ and each $i \in [w]$, $X_{d,\ell}$ contains (up to) $t_{i,d}$ elements of the form $((\cdot,\cdot), i, \cdot, \cdot)$; "greedily" is here taken to mean that the first $t_{i,d}$ elements of the form $((\cdot,\cdot), i, \cdot, \cdot)$ are placed in $X_{d,1}$, the next $t_{i,d}$ in $X_{d,2}$, and so on.
// Note that each $X_{d,\ell}$ can contain up to $t_{i,d}$ elements of the form $((\cdot,\cdot), i, \cdot, \cdot)$, of which there are a total of $(N/2^d) \cdot t_{i,d}$. Therefore X_d can indeed be decomposed into $(N/2^d)$ such partitions.
// Further note that each $X_{d,\ell}$ ($\ell \in [N/2^d]$) is a set of size $\sum_{i=1}^{w} t'_i$.

(g) For $\ell = 1, \ldots, N/2^d$:

- Sort $X_{d,\ell}$ according to the second element in increasing order, breaking ties with the fourth, and then if necessary the third element of the 4-tuples.
  // After this re-ordering, the first t_1 tuples are of the form $((\cdot,\cdot), 1, \cdot, \cdot)$, followed by t_2 tuples of the form $((\cdot,\cdot), 2, \cdot, \cdot)$, and so on.
- Set $\mathtt{DB}'_{d,\ell} \leftarrow (S_{d,\ell}[0].\mathsf{first}, \ldots, S_{d,\ell}[(\sum_{i=1}^{w} t_{i,d}) - 1].\mathsf{first}) \in \{0,1\}^{2|S_{d,\ell}|}$.
  // $\mathtt{DB}'_{d,\ell}$ is obtained by only considering the first of the four entries (which is a pair of bits from some $\mathtt{DB}_{d,j}$) of every element of $X_{d,\ell}$.
- Set $(\mathsf{ots}^{\star}_{d,\ell}, \mathsf{ots}^{\mathsf{dec}}_{d,\ell}) \xleftarrow{\$} \mathsf{rep\text{-}OTS}(1^\lambda, \mathsf{spir}_R, \mathtt{DB}'_{d,\ell}, \mathsf{rep}_d)$.

(h) If $d < n$:

- For $j = 1, \ldots, k$:
  Initialise $\mathtt{DB}_{d+1,j} \leftarrow 0^{\|N/2^d}$.
- For $\ell = 1, \ldots, N/2^d$:
  For $\ell' = 0, \ldots, (\sum_{i=1}^{w} t_{i,d}) - 1$:
  Parse $X_{d,\ell}[\ell']$ as $((\cdot,\cdot), \cdot, x, j)$, with $x \in [N/2^d]$ and $j \in [k]$.
  Set $\mathtt{DB}_{d+1,j}[x] \leftarrow \mathsf{ots}^{\mathsf{dec}}_{d,\ell}[\ell']$.

(i) Set $\mathsf{ots}^{\star}_d \leftarrow (\mathsf{ots}^{\star}_{d,1}, \ldots, \mathsf{ots}^{\star}_{d,N/2^d})$.

3. Set $\mathsf{spir}_S \coloneqq ((\mathsf{ots}^{\star}_1, \ldots, \mathsf{ots}^{\star}_n), \mathsf{ots}^{\mathsf{dec}}_n)$, and output spir_S.

corrSPIR$_D$: On input a sender message spir_S and an internal state st:

1. Parse spir_S as $\mathsf{spir}_S = ((\mathsf{ots}^{\star}_1, \ldots, \mathsf{ots}^{\star}_n), \mathsf{ots}^{\mathsf{dec}}_n)$, and parse st as $\mathsf{st}' \| \vec{b}$.
2. Set $(y_1, \ldots, y_k) \leftarrow \mathsf{MixAndMatch}(\vec{b})$ (i.e. $y_j \leftarrow b_{s_{j,1}} \ldots b_{s_{j,n}}$ for $j \in [n]$).
3. Initialise $(\widetilde{m}_1, \ldots, \widetilde{m}_k) \leftarrow \mathsf{ots}^{\mathsf{dec}}_n$.
4. For $d = 1, \ldots, n$:
   // The goal of this step is to identify which intermediary nodes of the Merkle tree can be recovered.

   (a) Initialise $X_d \leftarrow ((\perp,\ s_{j,d},\ x,\ j))_{j \in [k], x \in [0, N/2^d - 1]}$

   (b) Sort X_d according to the lexicographic order which first sorts by increasing fourth element (the "$j \in [k]$") and then, in case of equality, by increasing third element (the "$x \in [0, N/2^d - 1]$").

(c) Greedily partition X_d as $X_d = X_{d,1} \sqcup \cdots \sqcup X_{d,N/2^d}$ such that for each $\ell \in [N/2^d]$ and each $i \in [w]$, $X_{d,\ell}$ contains exactly $t_{i,d}$ elements of the form $(\cdot, i, \cdot, \cdot)$; "greedily" is here taken to mean that the first $t_{i,d}$ elements of the form $(\cdot, i, \cdot, \cdot)$ are placed in $X_{d,1}$, the next $t_{i,d}$ in $X_{d,2}$, and so on.

(d) For $\ell = 1, \ldots, N/2^d$:

Sort $X_{d,\ell}$ according to the second element in increasing order, breaking ties with the fourth, and then if necessary the third element of the 4-tuples.

(e) Parse $\mathsf{ots}^\star_d$ as $\mathsf{ots}^\star_d = (\mathsf{ots}^\star_{d,1}, \ldots, \mathsf{ots}^\star_{d,N/2^d})$

(f) For $j = 1, \ldots, k$:

- Set $\ell_{j,d}$ to be the unique $\ell \in [N/2^d]$ such that $(\bot,\ s_{j,d},\ (b_{s_{j,n}} \ldots b_{s_{j,d}}),\ j) \in X_{d,\ell}$.
- Set $\mathsf{ind}_{j,d}$ to be the index of $(\bot,\ s_{j,d},\ (b_{s_{j,n}} \ldots b_{s_{j,d}}),\ j)$ in $X_{d,\ell}$.
- Update $\widetilde{m}_j \leftarrow \mathsf{rep\text{-}OTD}(\{\mathsf{ind}_{j,d}\}, (\mathsf{ots}^\star_{d,\ell_{j,d}}, \widetilde{m}_j), \mathsf{rep}, \mathsf{st})$

5. Output $(\widetilde{m}_1, \ldots, \widetilde{m}_k)$.

Figure 6.8: corrSPIR from rep-OT.

Theorem 11. *Assume that* rep-OT *is a semi-honest two-round decomposable batch OT protocol with $\alpha(\cdot)$-overhead and T-bounded query repetitions. Then construction* $(\mathsf{corrSPIR}_R, \mathsf{corrSPIR}_S, \mathsf{corrSPIR}_D)$ *from fig. 6.8 is a two-round batch SPIR protocol with correlated "mix and match" queries. Furthermore the size of the receiver message is linear in $w + k \cdot n/T$ and the size of the sender message is upper bounded by $k + (\log N) \cdot (N-1) \cdot \alpha(w + k \cdot n/T)$ (where k is the batch number and N is the size of each of the k databases).*

Proof.

- *Size:* The receiver message is a rep-OT receiver message with $\sum_{i=1}^{w} \lceil t_i/T \rceil \leq \sum_{i=1}^{w}(1 + t_i/T) \leq w + (\sum_{i=1}^{w} t_i)/T = w + k \cdot n/T$ selection bits Since rep-OT has upload rate asymptotically one by definition, the receiver message is indeed linear in $w + k \cdot n/T$. The sender message is comprised a single bit per database, as well as $\sum_{d=1}^{n} N/2^d = N - 1$ different "reusable parts" of size $\alpha(w + k \cdot n/T)$. The sender message is therefore of size $k + (\log N) \cdot (N-1) \cdot \alpha(w + k \cdot n/T)$.

- *Correctness:* Correctness mostly follows from inspection, keeping in mind the following description of the instructions (alongside the comments in the pseudocode). Let $d \in [n]$. The pair $(\mathtt{DB}_{d,j}[x'_0], \mathtt{DB}_{d,j}[x'_1])$ corresponds to a pair of elements of $\mathtt{DB}_{d,j}$ whose indices only differ in bit $s_{j,d}$, wish we wish to "hash" down to a single bit using rate-1 OT. Because we only have access to a batched version the OT primitive, the pair will need to be batched with others (in fact, taken from different databases) in such a way that it corresponds to the $s^{\text{th}}_{j,d}$ selection bit. We therefore tag the pair with $s_{j,d}$. Furthermore, we will need to place the "hashed value" (which can be extracted by sender-message decomposability of the batched OT) thus obtained at the correct place in the Merkle tree, *i.e.* in the next level database. For this reason, we additionally tag the pair with (x, j),

so as to remember whence it came, and be able to deduce where it should be placed.

- *Security (Standalone Simulation):* We need to show that Π_{corrSPIR} from fig. 6.1, when instantiated with $\mathsf{corrSPIR} = (\mathsf{corrSPIR}_R, \mathsf{corrSPIR}_S, \mathsf{corrSPIR}_D)$ as defined in fig. 6.8 securely computes the functionality $f((\vec{m}_i)_{i\in[k]},\ \vec{b}) = (\perp,\ (\vec{m}_i[x_i])_{i\in[k]})$ where $(x_1, \ldots, x_k) \coloneqq \mathsf{MixAndMatch}(\vec{b})$ in the presence of static semi-honest adversaries.

  - *Corrupted Sender and Honest Receiver.* Because the functionality is deterministic, it suffices to show that there exist a PPT simulator $\mathsf{Sim}_S^{\mathsf{corrSPIR}}$ such that:

$$\{\mathsf{view}_S^{\Pi_{\mathsf{corrSPIR}}}(1^\lambda, (\vec{m}_i)_{i\in[k]}, \vec{b})\} \overset{\mathrm{c}}{\approx} \{\mathsf{Sim}_S^{\mathsf{corrSPIR}}(1^\lambda, (\vec{m}_i)_{i\in[k]}, \perp)\} \ .$$

    The sender's view in Π_{corrSPIR} consists of its input, internal random tape, and the messages it receives from the receiver. Note that the sender receives a single message, before it sends anything, and therefore its view can be split into two independent parts: the input and coins on one side, and the incoming transcript on the other. Therefore it suffices to show that we can simulate the incoming message given the security parameter and the sender's input. Since $\mathsf{rep\text{-}OT}$ is secure against a semi-honest sender, by definition there exists an expected polynomial time simulator $\mathsf{Sim}_S^{\mathsf{rep}}$ such that for every $\lambda \in \mathbb{N}^\star$ and every $\vec{b} = (b_1, \ldots, b_k) \in \{0,1\}^k$, and if further we define $\vec{b'} \leftarrow b_1^{\|\lceil t_1/T\rceil} \| \ldots \| b_w^{\|\lceil t_w/T\rceil}$,

$$\left\{\mathsf{otr} \colon (\mathsf{otr}, \mathsf{st}) \xleftarrow{\$} \mathsf{rep\text{-}OTR}(1^\lambda, \vec{b'})\right\} \overset{\mathrm{c}}{\approx} \left\{\mathsf{Sim}_S^{\mathsf{rep}}(1^\lambda)\right\} \ .$$

    Since the left hand side is exactly the distribution of the unique message received by the sender, it follows that the view of a semi-honest sender can be simulated.

  - *Corrupted Receiver and Honest Sender.* Because the functionality is deterministic, it suffices to show that there exists a PPT simulator $\mathsf{Sim}_R^{\mathsf{corrSPIR}}$ such that:

$$\{\mathsf{view}_R^{\Pi_{\mathsf{corrSPIR}}}(1^\lambda, (\vec{m}_j)_{j\in[k]}, \vec{b})\} \overset{\mathrm{c}}{\approx} \{\mathsf{Sim}_R^{\mathsf{corrSPIR}}(1^\lambda, \vec{b}, (\vec{m}_j[x_j])_{j\in[k]})\},$$
$$\text{where } (x_1, \ldots, x_k) \coloneqq \mathsf{MixAndMatch}(\vec{b}) \ .$$

    Note that the view of the corrupted receiver in Π_{corrSPIR} consists of its input (the vector of selection bits $\vec{b}$), its internal coins, and the single message spir_S it receives from the sender. Consider the simulator $\mathsf{Sim}_R^{\mathsf{corrSPIR}}$ which acts as follows:

    1. $\mathsf{Sim}_R^{\mathsf{corrSPIR}}$ starts by running the protocol as the receiver would, sampling random coins $\vec{r}_R$ and crafting $\mathsf{spir}_R \leftarrow \mathsf{corrSPIR}_R(1^\lambda, \vec{b}; \vec{r}_R)$;
    2. $\mathsf{Sim}_R^{\mathsf{corrSPIR}}$ builds k databases $\vec{m}'_1, \ldots, \vec{m}'_k$ of size N each, containing 0 everywhere except for one position each: the j^{th} database has $\vec{m}[x_j]$ in position x_j (recall that with the knowledge of $\vec{b}$, $\mathsf{Sim}_R^{\mathsf{corrSPIR}}$ is able to compute $(x_1, \ldots, x_k) = \mathsf{MixAndMatch}(\vec{b})$);

3. $\mathsf{Sim}_R^{\mathsf{corrSPIR}}$ runs $\mathsf{spir}_S \xleftarrow{\$} \mathsf{corrSPIR}_R(1^\lambda, \mathsf{spir}_R, (\vec{m}'_j)_{j\in[k]})$.
4. $\mathsf{Sim}_R^{\mathsf{corrSPIR}}$ outputs $(\vec{r}_R, \vec{b}, \mathsf{spir}_S)$.

In other words, $\mathsf{Sim}_R^{\mathsf{corrSPIR}}$ runs in its head an instance of the protocol of fig. 6.1, but replacing all of the unknown values (i.e. all except the $(\vec{m}_j[x_j])_{j\in[k]}$) in the databases $(\vec{m}_j)_{j\in[k]}$ with zeroes.
We will now prove that the view of the corrupted receiver in Π_{corrSPIR} is indistinguishable from the output of the above simulator $\mathsf{Sim}_R^{\mathsf{corrSPIR}}$ via a hybrid argument.

For $j \in [k]$ and $d \in [n]$, let $Y_{j,d} \coloneqq \{x = \overline{x_1 \dots x_n}^{(2)} \in [N] \colon (\forall d' < d, x_{d'} = b_{s_{j,d'}}) \wedge (x_d \neq b_{s_{j,d}})\}$ (in other words, $Y_{j,d}$ is the set of all elements of $[N]$ whose first $d-1$ digits in base two, but not the d^{th}, are the same as those of $\overline{b_{s_{j,1}} \dots b_{s_{j,d}}}^{(2)}$). Observe that for each $j \in [k]$, the $(Y_{j,d})_{d\in[n]}$ form a partition of $[N] \setminus \{\overline{b_{s_{j,1}} \dots b_{s_{j,d}}}^{(2)}\}$. Now, for $d \in [n]$ consider the simulator $\mathsf{Sim}_{R,d}^{\mathsf{corrSPIR}}$ which acts as follows:

1. $\mathsf{Sim}_{R,d}^{\mathsf{corrSPIR}}$ starts by running the protocol as the receiver would, sampling random coins $\vec{r}_R$ and crafting $\mathsf{spir}_R \leftarrow \mathsf{corrSPIR}_R(1^\lambda, \vec{b}; \vec{r}_R)$;
2. $\mathsf{Sim}_{R,d}^{\mathsf{corrSPIR}}$ builds k databases $\vec{m}'_1, \dots, \vec{m}'_k$ of size N each, as follows:

$$\forall j \in [k], \forall x \in [N],\ \vec{m}'_j[x] \coloneqq \begin{cases} \vec{m}_j[x] & \text{if } x \in Y_{j,d} \\ 0 & \text{otherwise} \end{cases}$$

   (recall that with the knowledge of $\vec{b}$, $\mathsf{Sim}_{R,d}^{\mathsf{corrSPIR}}$ is able to compute $(x_1, \dots, x_k) = \mathsf{MixAndMatch}(\vec{b})$, and therefore also $Y_{j,d}$);
3. $\mathsf{Sim}_{R,d}^{\mathsf{corrSPIR}}$ runs $\mathsf{spir}_S \xleftarrow{\$} \mathsf{corrSPIR}_R(1^\lambda, \mathsf{spir}_R, (\vec{m}'_j)_{j\in[k]})$.
4. $\mathsf{Sim}_{R,d}^{\mathsf{corrSPIR}}$ outputs $(\vec{r}_R, \vec{b}, \mathsf{spir}_S)$.

In other words, $\mathsf{Sim}_{R,d}^{\mathsf{corrSPIR}}$ runs in its head an instance of the protocol of fig. 6.1, but replacing all of the unknown values (i.e. all except the $(\vec{m}_j[x_j])_{j\in Y_{j,d}}$) in the databases $(\vec{m}_j)_{j\in[k]}$ with zeroes.
Observe that for all $j \in [k]$ we have that $Y_{j,1} = [N]$ and $Y_{j,n} = \{x_j\}$, and that $\mathsf{Sim}_{R,1}^{\mathsf{corrSPIR}}$ perfectly simulates the real world, while $\mathsf{Sim}_{R,n}^{\mathsf{corrSPIR}} = \mathsf{Sim}_R^{\mathsf{corrSPIR}}$ lives in the ideal world. For $j \in [1, n-1]$, indistinguishability of the distributions of outputs of $\mathsf{Sim}_{R,j}^{\mathsf{corrSPIR}}$ and $\mathsf{Sim}_{R,j+1}^{\mathsf{corrSPIR}}$ holds by invoking sender security of $\mathsf{rep\text{-}OT}$ k times. By construction, the process of running $\mathsf{corrSPIR}_S(1^\lambda, \mathsf{corrSPIR}_R(1^\lambda, \vec{b}), \cdot)$ compresses k size-N databases down to a single element each using a "Merkle-like forest".

Observe that $I_{j,d}$ (resp. $J_{j,d}$) corresponds to the 2^{n-d} (resp. 2^{n-d-1}) indices of database $\vec{m}_j$ whose lowest common ancestor with index x_j in the Merkle tree computed on $\vec{m}_j$ is in the layer[4] at most (resp. exactly) d. We will prove security via a sequence of hybrids, each giving the simulator access to the database elements $([\vec{m}_j]_{X_{j,d}})_{j\in[k]}$ for some $d \in [n]$. Proving indistinguishability of consecutive hybrids boils down to a reduction to the sender security of $\mathsf{rep\text{-}OT}$.

[4]We use the convention to count the *layers* of a Merkle tree from 1 to $\log N$, where layer 1 consists in the hashes computed directly on the original database elements and layer N is the final hash value (i.e. the root of the tree).

□

6.5 Instantiation from Standard Assumptions

For completeness, we provide a full description of the 2-round rate-1 batch OT construction from [BBDP22, Section 7], which we cast as decomposable. Their construction is centered around packed linearly homomorphic encryption, whose definition we recall in definition 24. Our contribution is to observe that if this packed LHE satisfies an additional property of "shrunken ciphertext decomposability", which we define in definition 25 and show in lemma 11 to be a property held by many concrete instantiations, then this two-round batch oblivious transfer is in fact decomposable.

6.5.1 Decomposable Packed Linearly Homomorphic Encryption

We recall in definition 24 the definition of packed linearly homomorphic encryption, and introduce in definition 25 the notion of *decomposability* for such an encryption scheme.

Definition 24 ((Packed) Linearly Homomorphic Encryption, [BBDP22])**.** *A* packed linearly homomorphic encryption *(LHE) scheme* $\mathcal{LHE}$ *over a finite group* $\mathbb{G}$ *is a tuple of* p.p.t. *algorithms* $\mathcal{LHE} = (\mathcal{LHE}.\mathsf{KeyGen}, \mathcal{LHE}.\mathsf{Enc}, \mathcal{LHE}.\mathsf{Shrink}, \mathcal{LHE}.\mathsf{DecShrink})$ *with the following syntax and properties:*

- $\mathsf{KeyGen}(1^\lambda, k)$*: On input a security parameter* 1^λ *and a plaintext length* $k \in \mathbb{N}^\star$*,* KeyGen *outputs a public key* pk *and a secret key* sk*. The size of the public key output by* $\mathsf{KeyGen}(1^\lambda, k)$ *is bounded by* $k \cdot \mathsf{poly}(\lambda)$*.*
- $\mathsf{Enc}(\mathsf{pk}, \vec{m} = (m_1, \ldots, m_k))$*: On input a public key* pk *and a message* $\vec{m} = (m_1, \ldots, m_k) \in \mathbb{G}^k$*,* Enc *outputs a ciphertext* ct*.*
- $\mathsf{Eval}(\mathsf{pk}, f, (\mathsf{ct}_1, \ldots, \mathsf{ct}_\ell))$*: On input a public key* pk*, a linear function* $f\colon (\mathbb{G}^k)^\ell \to \mathbb{G}^k$*, and a batch of* ℓ *ciphertexts* $(\mathsf{ct}_1, \ldots, \mathsf{ct}_\ell)$*,* Eval *outputs a ciphertext* $\tilde{\mathsf{ct}}$*.*
- $\mathsf{Shrink}(\mathsf{pk}, \mathsf{ct})$*: On input a public key* pk *and a ciphertext* ct*,* Shrink *outputs a shrunken ciphertext* ct'*.*
- $\mathsf{DecShrink}(\mathsf{sk}, \mathsf{ct})$*: On input a secret key* sk *and a shrunken ciphertext* ct*,* $\mathsf{DecShrink}$ *outputs a message* $\vec{m}$*.*
- Correctness. *For any* $\ell \in \mathbb{N}$*, any messages* $\vec{m}_1, \ldots, \vec{m}_\ell \in \mathbb{G}^k$*, and any linear function* $f\colon (\mathbb{G}^k)^\ell \to \mathbb{G}^k$*,*

$$\Pr\left[f(\vec{m}_1, \ldots, \vec{m}_\ell) = \tilde{m} \;:\; \begin{array}{l} (\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda, k) \\ \mathsf{ct}_i \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk}, \vec{m}_i) \text{ for } i \in [\ell] \\ \tilde{\mathsf{ct}} \xleftarrow{\$} \mathsf{Eval\&Shrink}(\mathsf{pk}, f, (\mathsf{ct}_1, \ldots, \mathsf{ct}_\ell)) \\ \tilde{m} \xleftarrow{\$} \mathsf{DecShrink}(\mathsf{sk}, \tilde{\mathsf{ct}}) \end{array} \right] = 1$$

where $\mathsf{Eval\&Shrink}$ *is an additional algorithm defined for convenience: On input a public key* pk*, a linear function* f*, and a batch of* ℓ *ciphertexts* $(\mathsf{ct}_1, \ldots, \mathsf{ct}_\ell)$*,* $\mathsf{Eval\&Shrink}$ *runs* $\tilde{\mathsf{ct}} \xleftarrow{\$} \mathsf{Shrink}(\mathsf{pk}, \mathsf{Eval}(\mathsf{pk}, f, (\mathsf{ct}_1, \ldots, \mathsf{ct}_\ell)))$ *and outputs the ciphertext* $\tilde{\mathsf{ct}}$*.*

- Semantic Security. *For all* $\lambda \in \mathbb{N}$, $k = \mathsf{poly}(\lambda)$, *and* p.p.t. *adversaries* $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$,

$$\Pr\left[b' = b \ : \ \begin{array}{l}(\mathsf{pk},\mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda, k) \\ (\vec{m}_0, \vec{m}_1, \mathsf{st}) \xleftarrow{\$} \mathcal{A}_0(\mathsf{pk}) \\ b \xleftarrow{\$} \{0,1\} \\ \mathsf{ct} \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk}, \vec{m}_b) \\ b' \xleftarrow{\$} \mathcal{A}_1(\mathsf{st}, \mathsf{ct})\end{array}\right] \leq \mathsf{negl}(\lambda) \ .$$

- Compactness.

  - *For sufficiently large* $k \in \mathbb{N}$, *any* $(\mathsf{pk},\mathsf{sk}) \in \mathsf{Supp}(\mathsf{KeyGen}(1^\lambda, k))$, *the size of the public key, i.e.* $|\mathsf{pk}|$, *is upper bounded by* $k \cdot \mathsf{poly}(n)$.
  - Rate-1. *For sufficiently large* $k \in \mathbb{N}$, *any linear function* $f\colon (\mathbb{G}^k)^\ell \to \mathbb{G}^k$, *and any* $(\vec{m}_1, \ldots, \vec{m}_\ell) \in (\mathbb{G}^k)^\ell$, *for all* $(\mathsf{pk},\mathsf{sk}) \in \mathsf{Supp}(\mathsf{KeyGen}(1^\lambda, k))$ *and* $\mathsf{ct}_i \in \mathsf{Supp}(\mathsf{Enc}(\mathsf{pk}, \vec{m}_i))$, $i \in [\ell]$*:*

$$|\mathsf{Eval\&Shrink}(\mathsf{pk}, f, (\mathsf{ct}_1, \ldots, \mathsf{ct}_\ell))| = |f(\vec{m}_1, \ldots, \vec{m}_\ell)| \cdot (1 + o(1)) = (k \cdot \log|\mathbb{G}|) \cdot (1 + o(1)) \ .$$

    When convenient, we will parse a rate-1 shrunken ciphertext as $\mathsf{ct} = (\mathsf{ct}_0, \mathsf{ct}_1)$, *where* $|\mathsf{ct}_0| = o(k)$ *and* $|\mathsf{ct}_1| = k$.

- Function Privacy. *There exists a simulator* $\mathsf{Sim}^{\mathsf{fn\text{-}priv}}_{\mathcal{LHE}}$ *such that for all messages* $(\vec{m}_1, \ldots, \vec{m}_\ell) \in (\mathbb{G}^k)^\ell$ *and all linear functions* $f\colon (\mathbb{G}^k)^\ell \to \mathbb{G}^k$, *for all adversaries* $\mathcal{A}$,

$$\left|\Pr\left[b = 1 \ : \ \begin{array}{l}(\mathsf{pk},\mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda, k) \\ \mathsf{ct}_i \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk}, \vec{m}_i) \ \textit{for} \ i \in [\ell] \\ \tilde{\mathsf{ct}} \xleftarrow{\$} \mathsf{Eval\&Shrink}(\mathsf{pk}, f, (\mathsf{ct}_i)_{i\in[\ell]}) \\ b \xleftarrow{\$} \mathcal{A}(\mathsf{pk}, \mathsf{sk}, \tilde{\mathsf{ct}})\end{array}\right] - \Pr\left[b = 1 \ : \ \begin{array}{l}(\mathsf{pk},\mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda, k) \\ \tilde{\mathsf{ct}} \xleftarrow{\$} \mathsf{Sim}^{\mathsf{fn\text{-}priv}}_{\mathcal{LHE}}(\mathsf{pk}, f((\vec{m}_i)_{i\in[\ell]})) \\ b \xleftarrow{\$} \mathcal{A}(\mathsf{pk}, \mathsf{sk}, \tilde{\mathsf{ct}})\end{array}\right]\right| \leq \mathsf{negl}(\lambda) \ .$$

  In other words, since $\mathsf{Sim}^{\mathsf{fn\text{-}priv}}_{\mathcal{LHE}}$ *does not use the function* f *to compute* $\tilde{\mathsf{ct}}$, *no non-trivial information about it is leaked from* $\tilde{\mathsf{ct}}$.

Informally, a packed LHE scheme is *decomposable* if, given the secret key and a shrunken ciphertext (which has size $k + o(k)$) missing some or all of the last k bits (note that the set of erased positions is assumed to be known), there is a way to recover the corresponding subset of the (homomorphically evaluated) plaintext vector but no information about the rest of the plaintext. Note that if any bit of the shrunken ciphertext is dropped other than the last k, then there is no correctness guarantee on recovering any information about the plaintext. We formalise this notion in definition 25.

Definition 25 (Decomposable Linearly Homomorphic Encryption, LHE)**.** *A packed linearly homomorphic encryption (LHE) scheme (definition 24)* $\mathcal{LHE} = (\mathcal{LHE}.\mathsf{KeyGen}, \mathcal{LHE}.\mathsf{Enc}, \mathcal{LHE}.\mathsf{Shrink}, \mathcal{LHE}.\mathsf{DecShrink})$ *over a finite group* $\mathbb{G}$ *is said to be* decomposable *if there exists a probabilistic polynomial time* partial decryption algorithm $\mathcal{LHE}.\mathsf{pDecShrink}$ *with the following syntax and properties:*

- Decomposability – Syntax. *On input a batch subset* $S \subseteq [k]$, *a secret key* sk, *a partial shrunken ciphertext* $\tilde{c} = (\tilde{c}_0, \tilde{c}_1)$ *where* $|\tilde{c}_0| = o(k \cdot \log|\mathbb{G}|)$ *and* $|\tilde{c}_1| = |S| \cdot \log|\mathbb{G}|$, $\mathcal{LHE}.\mathsf{pDecShrink}$ *outputs a partial message* $\tilde{m} \in \mathbb{G}^{|S|}$.

- Decomposability – Correctness. *For any $\ell \in \mathbb{N}$, any batch size $k \in \mathbb{N}^\star$, any messages $\vec{m}_1, \ldots, \vec{m}_\ell \in \mathbb{G}^k$, any linear function $f\colon (\mathbb{G}^k)^\ell \to \mathbb{G}^k$, and any batch subset $S \subseteq [k]$,*

$$\Pr\left[(f(\vec{m}_1,\ldots,\vec{m}_\ell))[S] = \tilde{m} \;:\; \begin{array}{l}(\mathsf{pk},\mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda, k)\\ \mathsf{ct}_i \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk}, \vec{m}_i) \text{ for } i \in [\ell]\\ \tilde{\mathsf{ct}} = (\tilde{\mathsf{ct}}_0, \tilde{\mathsf{ct}}_1) \xleftarrow{\$} \mathsf{Eval\&Shrink}(\mathsf{pk}, f, (\mathsf{ct}_1,\ldots,\mathsf{ct}_\ell))\\ \tilde{m} \xleftarrow{\$} \mathsf{pDecShrink}(S, \mathsf{sk}, (\tilde{\mathsf{ct}}_0, \tilde{\mathsf{ct}}_1[I_S]))\end{array}\right] = 1 \;,$$

$$\text{where } I_S \coloneqq \bigcup_{i\in S} [(i-1)|\mathbb{G}|, i|\mathbb{G}|] \;.$$

- Decomposability – Security. *There exists an expected polynomial time simulator $\mathsf{Sim}^{\mathsf{dec}}_{\mathcal{LHE}}$ such that for every $\lambda \in \mathbb{N}^\star$, any $\ell \in \mathbb{N}$, any batch size $k \in \mathbb{N}^\star$, any messages $\vec{m}_1, \ldots, \vec{m}_\ell \in \mathbb{G}^k$, any linear function $f\colon (\mathbb{G}^k)^\ell \to \mathbb{G}^k$, and any batch subset $S \subseteq [k]$,*

$$\left\{(\mathsf{pk},\mathsf{sk},\tilde{\mathsf{ct}}_0,\tilde{\mathsf{ct}}_1[I_S]) \;:\; \begin{array}{l}(\mathsf{pk},\mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda, k)\\ \mathsf{ct}_i \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk}, \vec{m}_i) \text{ for } i \in [\ell]\\ \tilde{\mathsf{ct}} = (\tilde{\mathsf{ct}}_0, \tilde{\mathsf{ct}}_1) \xleftarrow{\$} \mathsf{Eval\&Shrink}(\mathsf{pk}, f, (\mathsf{ct}_1,\ldots,\mathsf{ct}_\ell))\end{array}\right\} \overset{\mathrm{c}}{\approx}$$

$$\left\{(\mathsf{pk},\mathsf{sk},\mathsf{sim}_0,\mathsf{sim}_1) \;:\; \begin{array}{l}(\mathsf{pk},\mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda, k)\\ (\mathsf{sim}_0,\mathsf{sim}_1) \xleftarrow{\$} \mathsf{Sim}^{\mathsf{dec}}_{\mathcal{LHE}}(1^\lambda, \mathsf{pk}, k, S, f, (f(\vec{m}_1,\ldots,\vec{m}_\ell))[S])\end{array}\right\} \;,$$

$$\text{where } I_S \coloneqq \bigcup_{i\in S} [(i-1)|\mathbb{G}|, i|\mathbb{G}|] \;.$$

At a high level, function-privacy guarantees that, even given the secret key, a post-homomorphism shrunken ciphertext does not leak the function while decomposability guarantees that dropping selected parts of this ciphertext conceals information about the corresponding (post-homomorphism) plaintext. It makes intuitive sense that these properties should be achievable simultaneously, however this may not be clear *a priori* from the formalism since the simulator $\mathsf{Sim}^{\mathsf{dec}}_{\mathcal{LHE}}$ in definition 25 is given as input the function. We nevertheless establish this fact in lemma 10.

Lemma 10 (Function-Private Decomposability)**.** *If $\mathcal{LHE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Shrink}, \mathsf{DecShrink})$ is a decomposable packed linearly homomorphic encryption scheme over $\mathbb{G}$, then there exists an expected polynomial time simulator $\mathsf{Sim}^{\mathsf{priv\text{-}dec}}_{\mathcal{LHE}}$ such that for every $\lambda \in \mathbb{N}^\star$, any $\ell \in \mathbb{N}$, any batch size $k \in \mathbb{N}^\star$, any messages $\vec{m}_1, \ldots, \vec{m}_\ell \in \mathbb{G}^k$, any linear function $f\colon (\mathbb{G}^k)^\ell \to \mathbb{G}^k$, and any batch subset $S \subseteq [k]$,*

$$\left\{(\mathsf{pk},\mathsf{sk},\tilde{\mathsf{ct}}_0,\tilde{\mathsf{ct}}_1[I_S]) \;:\; \begin{array}{l}(\mathsf{pk},\mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda, k)\\ \mathsf{ct}_i \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk}, \vec{m}_i) \text{ for } i \in [\ell]\\ \tilde{\mathsf{ct}} = (\tilde{\mathsf{ct}}_0, \tilde{\mathsf{ct}}_1) \xleftarrow{\$} \mathsf{Eval\&Shrink}(\mathsf{pk}, f, (\mathsf{ct}_1,\ldots,\mathsf{ct}_\ell))\end{array}\right\} \overset{\mathrm{c}}{\approx}$$

$$\left\{(\mathsf{pk},\mathsf{sk},\mathsf{sim}_0,\mathsf{sim}_1) \;:\; \begin{array}{l}(\mathsf{pk},\mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda, k)\\ (\mathsf{sim}_0,\mathsf{sim}_1) \xleftarrow{\$} \mathsf{Sim}^{\mathsf{priv\text{-}dec}}_{\mathcal{LHE}}(1^\lambda, \mathsf{pk}, k, S, (f(\vec{m}_1,\ldots,\vec{m}_\ell))[S])\end{array}\right\} \;,$$

$$\text{where } I_S \coloneqq \bigcup_{i\in S} [(i-1)|\mathbb{G}|, i|\mathbb{G}|] \;. \quad (6.2)$$

In particular, note that f is not given as input to the simulator.

Proof. Let $\mathsf{Sim}^{\mathsf{dec}}_{\mathcal{LHE}}$ be defined as in definition 25 and consider the following algorithm $\mathsf{Sim}^{\mathsf{priv\text{-}dec}}_{\mathcal{LHE}}$: On input $(1^\lambda, \mathsf{pk}, k, S, y)$, parse pk so as to retrieve ℓ, define $y' \in \mathbb{G}^k$ as $y'[x] \coloneqq (y[x]$ if $x \in S$, and 0 otherwise), run $(\mathsf{ct}_0, \mathsf{ct}_1) \xleftarrow{\$} \mathsf{Sim}^{\mathsf{dec}}_{\mathcal{LHE}}(1^\lambda, \mathsf{pk}, k, S, \mathsf{cst}_{y'}, y)$ where $\mathsf{cst}_{y'}$ is the constant function on $(\mathbb{G}^k)^\ell$ equal to y', and output $(\mathsf{ct}_0, \mathsf{ct}_1[I_S])$ where $I_S \coloneqq \bigcup_{i\in S}[(i-1)|\mathbb{G}|, i|\mathbb{G}|]$. Let us now show that it satisfies the required property for lemma 10.

Let $\mathsf{Sim}^{\mathsf{fn\text{-}priv}}_{\mathcal{LHE}}$ be defined as in definition 24. Before we proceed, note that:

$$\begin{aligned}
&\forall f\colon (\mathbb{G}^k)^\ell \to \mathbb{G}^k \text{ linear}, \forall \vec{m}_1, \ldots, \vec{m}_\ell \in \mathbb{G}^k, \forall S \subseteq [k],\\
&\quad \left\{ (\mathsf{pk}, \mathsf{sk}, \tilde{\mathsf{ct}}_0, \tilde{\mathsf{ct}}_1[I_S]) \;:\; \begin{array}{l} (\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda, k) \\ \mathsf{ct}_i \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk}, \vec{m}_i) \text{ for } i \in [\ell] \\ \tilde{\mathsf{ct}} = (\tilde{\mathsf{ct}}_0, \tilde{\mathsf{ct}}_1) \xleftarrow{\$} \mathsf{Eval\&Shrink}(\mathsf{pk}, f, (\mathsf{ct}_1, \ldots, \mathsf{ct}_\ell)) \end{array} \right\}\\
&\quad \overset{\mathrm{c}}{\approx} \left\{ (\mathsf{pk}, \mathsf{sk}, \tilde{\mathsf{ct}}_0, \tilde{\mathsf{ct}}_1[I_S]) \;:\; \begin{array}{l} (\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda, k) \\ \tilde{\mathsf{ct}} = (\tilde{\mathsf{ct}}_0, \tilde{\mathsf{ct}}_1) \xleftarrow{\$} \mathsf{Sim}^{\mathsf{fn\text{-}priv}}_{\mathcal{LHE}}(\mathsf{pk}, f(\vec{m}_1, \ldots, \vec{m}_\ell)) \end{array} \right\},\\
&\qquad\qquad \text{where } I_S \coloneqq \bigcup_{i\in S}[(i-1)|\mathbb{G}|, i|\mathbb{G}|]\ . \quad (6.3)
\end{aligned}$$

Indeed, should there exist $f, \vec{m}_1, \ldots, \vec{m}_\ell, S$ such that there existed a *p.p.t.* adversary $\mathcal{A}$ with non-negligeable advantage in distinguishing the two above distributions, then $\mathcal{A}'$ defined as $\mathcal{A}'(x_1, x_2, x_3, x_4) \xleftarrow{\$} \mathcal{A}(x_1, x_2, x_3, x_4[I_S])$ (where $I_S \coloneqq \bigcup_{i\in S}[(i-1)|\mathbb{G}|, i|\mathbb{G}|]$) would distinguish the following two distributions with non-negligeable probability, thereby contradicting function-privacy:

$$\left\{ (\mathsf{pk}, \mathsf{sk}, \tilde{\mathsf{ct}}_0, \tilde{\mathsf{ct}}_1) \;:\; \begin{array}{l} (\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda, k) \\ \mathsf{ct}_i \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk}, \vec{m}_i) \text{ for } i \in [\ell] \\ \tilde{\mathsf{ct}} = (\tilde{\mathsf{ct}}_0, \tilde{\mathsf{ct}}_1) \xleftarrow{\$} \mathsf{Eval\&Shrink}(\mathsf{pk}, f, (\mathsf{ct}_1, \ldots, \mathsf{ct}_\ell)) \end{array} \right\}$$
$$\text{and } \left\{ (\mathsf{pk}, \mathsf{sk}, \tilde{\mathsf{ct}}_0, \tilde{\mathsf{ct}}_1) \;:\; \begin{array}{l} (\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda, k) \\ \tilde{\mathsf{ct}} = (\tilde{\mathsf{ct}}_0, \tilde{\mathsf{ct}}_1) \xleftarrow{\$} \mathsf{Sim}^{\mathsf{fn\text{-}priv}}_{\mathcal{LHE}}(\mathsf{pk}, f(\vec{m}_1, \ldots, \vec{m}_\ell)) \end{array} \right\}.$$

We are now ready to prove that our candidate $\mathsf{Sim}^{\mathsf{priv\text{-}dec}}_{\mathcal{LHE}}$ indeed satisfies the requirements of eq. (6.2). Let $\lambda \in \mathbb{N}^\star$, $\ell \in \mathbb{N}$, $k \in \mathbb{N}^\star$, $\vec{m}_1, \ldots, \vec{m}_\ell \in \mathbb{G}^k$, and $S \subseteq [k]$. Let $f\colon (\mathbb{G}^k)^\ell \to \mathbb{G}^k$ be a linear function. Define $y \coloneqq (f(\vec{m}_1, \ldots, \vec{m}_\ell))[S]$, and $y' \in \mathbb{G}^k$ as $y'[x] \coloneqq (y[x]$ if $x \in S$, and 0 otherwise). Let $\mathsf{cst}_{y'}$ denote the constant function on $(\mathbb{G}^k)^\ell$ equal to y'.

Observe that, if we define $I_S \coloneqq \bigcup_{i\in S}[(i-1)|\mathbb{G}|, i|\mathbb{G}|]$:

$$
\begin{aligned}
&\left\{(\mathsf{pk},\mathsf{sk},\tilde{\mathsf{ct}}_0,\tilde{\mathsf{ct}}_1[I_S]) \;:\; \begin{array}{l}(\mathsf{pk},\mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda,k)\\ \mathsf{ct}_i \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk},\vec{m}_i) \text{ for } i\in[\ell]\\ \tilde{\mathsf{ct}} = (\tilde{\mathsf{ct}}_0,\tilde{\mathsf{ct}}_1) \xleftarrow{\$} \mathsf{Eval\&Shrink}(\mathsf{pk},f,(\mathsf{ct}_1,\ldots,\mathsf{ct}_\ell))\end{array}\right\}\\
&\overset{c}{\approx}\left\{(\mathsf{pk},\mathsf{sk},\tilde{\mathsf{ct}}_0,\tilde{\mathsf{ct}}_1[I_S]) \;:\; \begin{array}{l}(\mathsf{pk},\mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda,k)\\ \tilde{\mathsf{ct}} = (\tilde{\mathsf{ct}}_0,\tilde{\mathsf{ct}}_1) \xleftarrow{\$} \mathsf{Sim}^{\mathsf{fn\text{-}priv}}_{\mathcal{LHE}}(\mathsf{pk},f(\vec{m}_1,\ldots,\vec{m}_\ell))\end{array}\right\}\\
&\overset{c}{\approx}\left\{(\mathsf{pk},\mathsf{sk},\tilde{\mathsf{ct}}_0,\tilde{\mathsf{ct}}_1[I_S]) \;:\; \begin{array}{l}(\mathsf{pk},\mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda,k)\\ \mathsf{ct}_i \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk},f(\vec{m}_i)) \text{ for } i\in[\ell]\\ \tilde{\mathsf{ct}} = (\tilde{\mathsf{ct}}_0,\tilde{\mathsf{ct}}_1) \xleftarrow{\$} \mathsf{Eval\&Shrink}(\mathsf{pk},g,(\mathsf{ct}_1,\ldots,\mathsf{ct}_\ell))\end{array}\right\}\\
&\overset{c}{\approx}\left\{(\mathsf{pk},\mathsf{sk},\mathsf{sim}_0,\mathsf{sim}_1) \;:\; \begin{array}{l}(\mathsf{pk},\mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda,k)\\ (\mathsf{sim}_0,\mathsf{sim}_1) \xleftarrow{\$} \mathsf{Sim}^{\mathsf{dec}}_{\mathcal{LHE}}(1^\lambda,\mathsf{pk},k,S,g,(f(\vec{m}_1,\ldots,\vec{m}_\ell))[S])\end{array}\right\}\\
&\equiv\left\{(\mathsf{pk},\mathsf{sk},\mathsf{sim}_0,\mathsf{sim}_1) \;:\; \begin{array}{l}(\mathsf{pk},\mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda,k)\\ (\mathsf{sim}_0,\mathsf{sim}_1) \xleftarrow{\$} \mathsf{Sim}^{\mathsf{priv\text{-}dec}}_{\mathcal{LHE}}(1^\lambda,\mathsf{pk},k,S,(f(\vec{m}_1,\ldots,\vec{m}_\ell))[S])\end{array}\right\}.
\end{aligned}
$$

The first two steps follow from eq. (6.3), the third from the definition of $\mathsf{Sim}^{\mathsf{dec}}_{\mathcal{LHE}}$, and the fourth by how we defined $\mathsf{Sim}^{\mathsf{priv\text{-}dec}}_{\mathcal{LHE}}$. □

We recall in fig. 6.9 the construction of packed LHE under QR from [DGI+19, BBDP22] and note that it is decomposable.

dec-$\mathcal{LHE}$, Adapted from [DGI+19, BBDP22]

KeyGen: On input the security parameter 1^λ and a batch size k:

1. Choose two safe primes $p = 2p'+1$ and $q = 2q'+1$ where p', q' are primes and compute $N = pq$. Choose a generator g of $\mathbb{QR}_N$.
2. Sample $s \xleftarrow{\$} \mathbb{Z}^k_{\phi(N)/2}$ and compute $\vec{h} \leftarrow g^s$.
3. Output $\mathsf{pk} \leftarrow (N, g, \vec{h})$ and $\mathsf{sk} \leftarrow s$.

Enc: On input the public key pk and a batch of k messages $\vec{m} = (m_1, \ldots, m_k)$:

1. Parse pk as $\mathsf{pk} = (N, g, \vec{h} = (h_1, \ldots, h_k))$
2. Sample $r \xleftarrow{\$} \mathbb{Z}_{(N-1)/2}$. Compute $c_1 \leftarrow g^r$ and $c_{2,i} \leftarrow (-1)^{m_i} h_i^r$ for $i \in [k]$.
3. Output $\mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2 = (c_{2,1}, \ldots, c_{2,k}))$.

Eval: On input a public key pk, an ℓ-input linear function f, and ℓ ciphertexts $(\mathsf{ct}_1, \ldots, \mathsf{ct}_\ell)$:

1. Parse pk as $\mathsf{pk} = (N, g, \vec{h} = (h_1, \ldots, h_k))$, f as $f(X_1, \ldots, X_\ell) = \sum_{j=1}^{\ell} a_j \cdot X_j + \vec{b}$ where $a_1, \ldots, a_\ell \in \{0,1\}$ and $\vec{b} \in \{0,1\}^k$; For $j \in [\ell]$, parse ct_j as $(c_{1,j}, \vec{c}_{2,j} = (c_{2,1,j}, \ldots, c_{2,k,j}))$.
2. Sample $t \xleftarrow{\$} \mathbb{Z}_{(N-1)/2}$ and compute $\tilde{c}_1 \leftarrow g^t \cdot \prod_{j=1}^{\ell} c_{1,j}^{a_j}$ and $\tilde{c}_{2,i} \leftarrow h_i^t \cdot (-1)^{b_i} \cdot \prod_{j=1}^{\ell} c_{2,i,j}^{a_j}$, then set $\tilde{\mathsf{ct}} = (\tilde{c}_1, \tilde{c}_2 = (\tilde{c}_{2,1}, \ldots, \tilde{c}_{2,k}))$.
3. Output $\tilde{\mathsf{ct}}$.

Shrink: On input a public key pk and a packed ciphertext ct:

1. Parse pk as $\mathsf{pk} = (N, g, \vec{h} = (h_1, \ldots, h_k))$ and ct as $\mathsf{ct} = (c_1, c_2 = (c_{2,1}, \ldots, c_{2,k}))$.
2. For $i \in [k]$, set $e_i \leftarrow 0$ if $c_{2,i} <_{\mathbb{J}_N} -c_{2,i}$ and $e_i \leftarrow 1$ otherwise.
3. Output $(c_1, (e_1, \ldots, e_k))$.

DecShrink: On input a secret key sk and a shrunken ciphertext ct:

1. Parse ct as $\mathsf{ct} = (c_1, (e_1, \ldots, e_k))$ where $\forall i \in [k], e_i \in \mathbb{G}$.
2. Parse sk as $\mathsf{sk} = (\mathsf{sk}_1, \ldots, \mathsf{sk}_k)$.
3. For $i \in [k]$, set $\tilde{m}_i \leftarrow \mathsf{LEq}(c_1^{\mathsf{sk}_i}, (-1) \cdot c_1^{\mathsf{sk}_i})$.
4. Output $(\tilde{m}_1, \ldots, \tilde{m}_k)$.

pDecShrink: On input a batch subset $S \subseteq [k]$, a secret key sk, a partial shrunken ciphertext $\tilde{c}$:

1. Parse $\tilde{c}$ as $\tilde{c} = (\tilde{c}_0, (e_i)_{i \in S})$ where $\forall i \in S, e_i \in \mathbb{G}$.
2. Parse sk as $\mathsf{sk} = (\mathsf{sk}_1, \ldots, \mathsf{sk}_k)$.
3. For $i \in S$, set $\tilde{m}_i \leftarrow \mathsf{LEq}(c_1^{\mathsf{sk}_i}, (-1) \cdot c_1^{\mathsf{sk}_i})$.
4. Output $(\tilde{m}_1, \ldots, \tilde{m}_k)$.

Figure 6.9: Decomposable Packed Linearly Homomorphic Encryption from QR.

Lemma 11. *Assuming the Quadratic Residuosity assumption, the construction of fig. 6.9 is a decomposable packed linearly homomorphic encryption scheme.*

Proof. The above scheme was shown to be a circuit-private LHE by [DGI+19,BBDP22], so it only remains to show it is decomposable. Decomposable security follows from the fact that (with the notations of fig. 6.9) a partial ciphertext for batch subset S is of the form $\mathsf{ct} = (c_1, (e_i)_{i \in S})$, which can be observed to information-theoretically contain no information about $(m_i)_{i \in [N] \setminus S}$. Decomposable correctness follows from inspection of the "locality" of $\mathsf{DecShrink}$. □

6.5.2 Two-Round co-PIR

We now recall the notion of *co-PIR* from [BBDP22, Section 6.1] (or *punctured OT* [BGI17]), which allows a receiver holding as input a set of indices S to interact with an input-free server in such a way that the sender obtains a pseudorandom string $\vec{y} \in \mathbb{Z}_q^m$ while the receiver gets $\vec{y}[[N] \setminus S]$ (all entries of $\vec{y}$ which are *not* in S).

Definition 26 (Two-Round co-PIR, [BBDP22])**.** *A two-round* co-PIR *scheme over $\mathbb{Z}_q$ (with poly-logarithmic communication complexity) is parameterised by an integer m where $m = \mathsf{poly}(\lambda)$, and is composed by a tuple of algorithms $\mathsf{copir} = (\mathsf{copir.Query}, \mathsf{copir.Send}, \mathsf{copir.Receive})$ with the following syntax and properties:*

- $\mathsf{Query}(1^\lambda, S)$: *On input the security parameter 1^λ and a set of indices $S \subseteq [m]$,* Query *outputs a receiver message copir_R and a private state st.*

- $\mathsf{Send}(\mathsf{copir}_R)$: *On input a receiver message* copir_R, Send *outputs a sender message* copir_S *and a string* $\boldsymbol{y} \in \mathbb{Z}_q^m$.

- $\mathsf{Dec}(\mathsf{copir}_S, \mathsf{st})$: *On input a sender message* copir_S *and a state* st, Dec *outputs a string* $\tilde{\boldsymbol{y}} \in \mathbb{Z}_q^m$.

- Correctness. *A co-pir scheme* copir *is said to be* correct *if for any* $m = \mathsf{poly}(\lambda)$ *and* $S \subseteq [m]$,

$$\Pr\left[\boldsymbol{y}[\overline{S}] = \tilde{\boldsymbol{y}}[\overline{S}] \colon \begin{array}{rl} (\mathsf{copir}_R, \mathsf{st}) & \xleftarrow{\$} \mathsf{Query}(1^\lambda, S) \\ (\mathsf{copir}_S, \boldsymbol{y}) & \xleftarrow{\$} \mathsf{Send}(\mathsf{copir}_R) \\ \tilde{\boldsymbol{y}} & \xleftarrow{\$} \mathsf{Receive}(\mathsf{copir}_S, \mathsf{st}) \end{array}\right] = 1 \ ,$$

where $\overline{S} = [m] \setminus S$.

- Receiver Security. *For all* $m = \mathsf{poly}(\lambda)$, *any subsets* $S_1, S_2 \subseteq [m]$, *any* p.p.t. *adversary* $\mathcal{A}$,

$$\Big|\Pr\left[\mathcal{A}(k, \mathsf{copir}_R) = 1 \colon (\mathsf{copir}_R, \mathsf{st}) \xleftarrow{\$} \mathsf{Query}(1^\lambda, S_1)\right] - \Pr\left[\mathcal{A}(k, \mathsf{copir}_R) = 1 \colon (\mathsf{copir}_R, \mathsf{st}) \xleftarrow{\$} \mathsf{Query}(1^\lambda, S_2)\right]\Big| \leq \mathsf{negl}(\lambda) \ .$$

- Sender Security. *For all* $m = \mathsf{poly}(\lambda)$, *any subset* $S \subseteq [m]$, *any* p.p.t. *adversary* $\mathcal{A}$,

$$\left|\Pr\left[\mathcal{A}(k, \mathsf{st}, \mathsf{copir}_S, \boldsymbol{y}_S) = 1 \colon \begin{array}{rl} (\mathsf{copir}_R, \mathsf{st}) & \xleftarrow{\$} \mathsf{Query}(1^\lambda, S) \\ (\mathsf{copir}_S, \boldsymbol{y}) & \xleftarrow{\$} \mathsf{Send}(\mathsf{copir}_R, \boldsymbol{x}) \end{array}\right] - \Pr\left[\mathcal{A}(k, \mathsf{st}, \mathsf{copir}_S, \boldsymbol{y}'_S) = 1 \colon \begin{array}{rl} (\mathsf{copir}_R, \mathsf{st}) & \xleftarrow{\$} \mathsf{Query}(1^\lambda, S) \\ (\mathsf{copir}_S, \boldsymbol{y}) & \xleftarrow{\$} \mathsf{Send}(\mathsf{copir}_R, \boldsymbol{x}) \\ \boldsymbol{y}'_S & \xleftarrow{\$} \mathbb{Z}_q^{|S|} \end{array}\right]\right| \leq \mathsf{negl}(\lambda) \ .$$

- Compactness ("Polylogarithmic Communication"). *For all* $m = \mathsf{poly}(\lambda)$, *any subset* $S \subseteq [m]$, *any* $(\mathsf{copir}_R, \mathsf{st}) \in \mathsf{Supp}(\mathsf{Query}(1^\lambda, S))$, *and any* $(\mathsf{copir}_S, \boldsymbol{y}) \in \mathsf{Supp}(\mathsf{Send}(\mathsf{copir}_R))$, *it holds that* $|\mathsf{copir}_R|, |\mathsf{copir}_S| = |S| \cdot \mathsf{polylog}(m) \cdot \mathsf{poly}(\lambda)$.

Lemma 12 (Instantiation of co-PIR [BBDP22])**.** *Assuming the Quadratic Residuosity assumption, there exists two-round polylogarithmic co-PIR over* $\{0,1\}$.

6.5.3 Decomposable OT from Decomposable LHE

For the sake of more unified notations with the construction of rate-1 OT from [BBDP22, Section 7], we depart in this section from definition 29, and take two-round single-server private information retrieval with (polylogarithmic communication) to be a tuple of algorithms $\mathsf{PIR} = (\mathsf{PIR.Query}, \mathsf{PIR.Send}, \mathsf{PIR.Receive})$.

dec-OT

Parameters: Batch number $k = \ell \cdot t$; Exact LPN error τ; A constant $\epsilon \in (0,1)$ tied to the hardness of one of the underlying LPN assumptions.
Requires:

- $\mathcal{LHE} = (\mathcal{LHE}.\mathsf{KeyGen}, \mathcal{LHE}.\mathsf{Enc}, \mathcal{LHE}.\mathsf{Eval}, \mathcal{LHE}.\mathsf{Shrink}, \mathcal{LHE}.\mathsf{DecShrink})$ is a decomposable packed linearly homomorphic encryption scheme (with partial decryption algorithm $\mathsf{pDecShrink}$) with plaintext space $\{0,1\}^\ell$ and equipped with a post-homomorphism shrinking procedure $\mathcal{LHE}.\mathsf{Shrink}$ which converts ciphertexts into a rate 1 representation.
- $\mathsf{copir} = (\mathsf{copir.Query}, \mathsf{copir.Send}, \mathsf{copir.Receive})$ is a two-round polylogarithmic co-PIR scheme over $\{0,1\}$ and parameterised by a database size of t.
- $\mathsf{PIR} = (\mathsf{PIR.Query}, \mathsf{PIR.Send}, \mathsf{PIR.Receive})$ is a two-round polylogarithmic PIR scheme over $\{0,1\}$.

dec-OTR: On input the security parameter 1^λ and a vector of selection bits $\vec{b} = (b_1, \ldots, b_k) \in \{0,1\}^k$:

1. Parse $\vec{b}$ as $\vec{b} = (\vec{b}_1, \ldots, \vec{b}_\ell)$ where each $\vec{b}_i \in \{0,1\}^t$ is a block of size t .
2. Choose $\boldsymbol{A} \xleftarrow{\$} \{0,1\}^{n \times t}$ uniformly at random, and sample $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathcal{LHE}.\mathsf{KeyGen}(1^\lambda, \ell)$.
3. For $i = 1, \ldots, \ell$:
   (a) Sample $\vec{s}_i \xleftarrow{\$} \{0,1\}^n$, and $\vec{e}_i \xleftarrow{\$} \mathsf{HW}_\tau(\{0,1\}^t)$ (Uniformly random τ-sparse length-t vector)
   (b) Compute $\vec{c}_i \leftarrow \vec{s}_i \cdot \boldsymbol{A} + \vec{e}_i + \vec{b}_i$
   (c) Set $\boldsymbol{S}_i \leftarrow \mathsf{SingleRowMatrix}(\ell, n, i, \vec{s}_i)$
   (d) Compute a matrix-ciphertext $\mathbf{ct}_i \xleftarrow{\$} \mathcal{LHE}.\mathsf{Enc}(\mathsf{pk}, \boldsymbol{S}_i)$
   (e) Set $J_i = \mathsf{supp}(\vec{e}_i)$
   (f) Compute $(\mathsf{copir}_{R,i}, \mathsf{st}_i) \leftarrow \mathsf{coPIR.Query}(J_i)$
4. Set $\mathsf{otr} \leftarrow (\mathsf{pk}, \boldsymbol{A}, \{\mathsf{ct}_i, \vec{c}_i, \mathsf{copir}_{R,i}\}_{i\in[\ell]}, \{q_{i,j}\}_{i\in[\ell],j\in[t]})$
5. Set $\mathsf{st} \leftarrow (\mathsf{sk}, \{\mathsf{st}_i, J_i\}_{i\in[\ell]}, \{\hat{\mathsf{st}}_{i,j}\}_{i\in[\ell],j\in[t]})$
6. Output $(\mathsf{otr}, \mathsf{st})$

dec-OTS: On input the security parameter 1^λ, a receiver message otr, a database $\vec{m} \in \{0,1\}^{2k}$:

1. Parse otr as $\mathsf{otr} = (\mathsf{pk}, \boldsymbol{A}, \{\mathsf{ct}_i, \vec{c}_i, \mathsf{copir}_{R,i}\}_{i\in[\ell]}, \{q_{i,j}\}_{i\in[\ell],j\in[t]})$
2. Parse $\vec{m}$ as $\vec{m} = ((m_{0,i,j}, m_{1,i,j}))_{i\in[\ell],j\in[t]}$ and set $\vec{m}_{b,i} \leftarrow (m_{b,i,1}, \ldots, m_{b,i,t}) \in \{0,1\}^t$.
3. For $i = 1, \ldots, \ell$:
   (a) $(\vec{y}_i, \mathsf{copir}_S) \xleftarrow{\$} \mathsf{coPIR.Send}(\mathsf{copir}_{R,i})$ where $\vec{y}_i \leftarrow (y_{i,1}, \ldots, y_{i,t})$

(b) Set $\vec{z}_i \leftarrow \vec{m}_{0,i} + \vec{y}_i$

4. Set $\boldsymbol{Z} \leftarrow \mathsf{RowMatrix}(\ell, t, \vec{z}_1, \ldots, \vec{z}_\ell)$
5. For $i = 1, \ldots, \ell$:
   (a) Set $\boldsymbol{C}_i \leftarrow \mathsf{SingleRowMatrix}(\ell, t, i, \vec{c}_i)$
   (b) Set $\boldsymbol{D}_i \leftarrow \mathsf{Diag}(t, \vec{m}_{1,i} - \vec{m}_{0,i})$
6. Define the $\mathbb{Z}_2$-linear function

$$\begin{array}{rccc} f\colon & (\{0,1\}^{\ell\times n})^\ell & \to & \{0,1\}^{\ell\times t} \\ & (\boldsymbol{X}_1, \ldots, \boldsymbol{X}_\ell) & \mapsto & \left(\sum_{i=1}^{\ell} (-\boldsymbol{X}_i\boldsymbol{A} + \boldsymbol{C}_i)\cdot \boldsymbol{D}_i\right) + \boldsymbol{Z} \end{array}$$

7. Compute $\hat{\mathsf{ct}} \xleftarrow{\$} \mathcal{LHE}.\mathsf{Eval\&Shrink}(\mathsf{pk}, f, \mathsf{ct}_1, \ldots, \mathsf{ct}_\ell)$.
8. For $i = 1, \ldots, \ell$:
   (a) Set $\mathtt{DB}_i \leftarrow (y_{i,1} + (m_{1,i,1} - m_{0,i,1}), \ldots, y_{i,t} + (m_{1,i,t} - m_{0,i,t}))$
   (b) For $j = 1, \ldots, t$:
   Compute $\vec{r}_{i,j} \leftarrow \mathsf{PIR.Send}(\mathtt{DB}_i, \vec{q}_{i,j})$
9. Set $\mathsf{ots}^\star \leftarrow (\{\mathsf{copir}_{S,i}\}_{i\in[\ell]}, \{r_{i,j}\}_{i\in[\ell],j\in[t]})$
10. Set $\mathsf{ots}^{\mathsf{dec}} \leftarrow \hat{\mathsf{ct}}$
11. Output $(\mathsf{ots}^\star, \mathsf{ots}^{\mathsf{dec}})$

dec-OTD: On input a batch subset $K \subseteq [k]$, a partial sender message ots' and an state st:

1. Parse ots as $\mathsf{ots} = (\mathsf{ots}^\star, \mathsf{ots}'_{\mathsf{dec}}) = ((\{\mathsf{copir}_{S,i}\}_{i\in[\ell]}, \{r_{i,j}\}_{i\in[\ell],j\in[t]}), \hat{\mathsf{ct}}')$
2. Parse st as $\mathsf{st} = (\mathsf{sk}, \{\mathsf{st}_i, J_i\}_{i\in[\ell]}, \{\hat{\mathsf{st}}_{i,j}\}_{i\in[\ell],j\in[t]})$
3. For $i = 1, \ldots, \ell$:
   (a) Compute $\overline{\boldsymbol{y}}_i \leftarrow (\overline{y}_{i,1}, \ldots, \overline{y}_{i,t}) \leftarrow \mathsf{coPIR.Retrieve}(\mathsf{copir}_{S,i}, \mathsf{st}_i)$
   (b) For $j = 1, \ldots, t$:
   Compute $\overline{\boldsymbol{z}}_{i,j} \leftarrow \mathsf{PIR.Retrieve}(\mathsf{copir}_{S,i}, \hat{\mathsf{st}}_{i,j})$
   (c) Set $\vec{z}_i \leftarrow (z_{i,1}, \ldots, z_{i,t})$ where

$$z_{i,l} = \begin{cases} \overline{z}_{i,j} & \text{if } l = J_i[j] \\ \overline{y}_{i,\ell} & \text{otherwise} \end{cases}.$$

4. Set $\boldsymbol{Z} \leftarrow \mathsf{RowMatrix}(\ell, t, \vec{z}_1, \ldots, \vec{z}_\ell)$
5. Set $S_K \coloneqq \{(j \bmod \ell, j \text{ quo } \ell)\colon j \in K\} \subseteq [\ell] \times [t]$

6. Compute $\hat{\boldsymbol{W}}' \leftarrow \mathcal{LHE}.\mathsf{pDecShrink}(K, \mathsf{sk}, \hat{\mathsf{ct}}')$, $\hat{\boldsymbol{W}}$ as

$$\hat{\boldsymbol{W}}[i,j] \coloneqq \begin{cases} \hat{\boldsymbol{W}}'[i,j] & \text{if } (i,j) \in S_K \\ 0 & \text{if } (i,j) \in [\ell] \times [t] \setminus S_K \end{cases}$$

and $\boldsymbol{W} \leftarrow \hat{\boldsymbol{W}} - \boldsymbol{Z}$.

7. Let $\vec{w}_1, \ldots, \vec{w}_\ell$ be the rows of $\boldsymbol{W}$.

8. Output $(\vec{w}_i[j])_{(i,j)\in S_K} \in \{0,1\}^{|K|}$.

Figure 6.10: Decomposable Batch OT from Decomposable LHE, PIR, and co-PIR

Theorem 12 (Decomposable Batch OT from Decomposable LHE, PIR, and co-PIR)**.** *Under the binary LPN assumption* $\mathsf{LPN}(\mathsf{dim}, \mathsf{num}, \rho)$ *with dimension* $\mathsf{dim} = \mathsf{poly}(\lambda)$*, number of samples* $\mathsf{num} = \mathsf{dim}^c$ *(for any constant* $c > 1$*), and noise rate* $\rho = \mathsf{num}^{\varepsilon-1}$ *(for some constant* $\varepsilon < 1$*), if the following conditions are met:*

- *The batch size is of the form* $k = \ell \cdot t$*, where* $\ell, t \in \mathbb{N}$*.*
- $\mathcal{LHE}$ *is a rate-*1 *decomposable packed linearly homomorphic encryption scheme (definition 25) with plaintext space* $\{0,1\}^\ell$*, whose post-homomorphism shrunken ciphertexts have size* $k + \alpha(k)$*, where* $\alpha = o(1)$ *is some sublinear function;*
- coPIR *is a two-round co-PIR scheme over* $\{0,1\}$ *with poly-logarithmic communication complexity and parameterised by database size* t*,*
- PIR *is a two-round PIR scheme with poly-logarithmic communication complexity and sender privacy for database size* t*,*

then construction $\mathsf{dec\text{-}OT} = (\mathsf{dec\text{-}OTR}, \mathsf{dec\text{-}OTS}, \mathsf{dec\text{-}OTD})$ *from fig. 6.10 is a two-round decomposable batch OT with* $\alpha + \mathsf{polylog}$ *overhead.*

Proof.

- *Decomposable Correctness:* Observe that $\mathsf{dec\text{-}OTR}$ and $\mathsf{dec\text{-}OTS}$ are defined exactly as OTR and OTS in [BBDP22, Section 7], and furthermore that $\mathsf{dec\text{-}OTD}([k], \cdot, \cdot)$ is in fact the same algorithm (with the observation that, by decomposable correctness of $\mathcal{LHE}$, $\mathsf{DecShrink}(\cdot,\cdot) \equiv \mathsf{pDecShrink}([k], \cdot, \cdot)$) as $\mathsf{OTD}(\cdot,\cdot)$. Correctness therefore follows from [BBDP22, Theorem 1].

- *Receiver Security (Against Semi-Honest Sender):* Observe that $\mathsf{dec\text{-}OTR}$ is the same as the OTR from [BBDP22, Section 7], and therefore sender security follows from [BBDP22, Theorem 2].

- *Decomposable Sender Security (Against Semi-Honest Receiver):* Observe that $\mathsf{dec\text{-}OTS}$ is the same as the OTS from [BBDP22, Section 7], and that $\mathsf{dec\text{-}OTD}$ is only slightly modified from OTD. As such, the proof will closely follow that of

[BBDP22, Theorem 3]. Let $\mathsf{Sim}^{\mathsf{dec}}_{\mathcal{LHE}}$ be a simulator as defined in the decomposable security of $\mathcal{LHE}$, and consider the following simulator $\mathsf{Sim}_{\mathsf{dec\text{-}OT}}$ which, on input $(1^\lambda, K, (m_i)_{i\in K}, \vec{b}, \mathsf{otr}, \mathsf{st})$, acts as follows:

1. Parse otr as $\mathsf{otr} = (\mathsf{pk}, \boldsymbol{A}, \{\mathsf{ct}_i, \vec{c}_i, \mathsf{copir}_{R,i}\}_{i\in[\ell]}, \{q_{i,j}\}_{i\in[\ell],j\in[t]})$
2. Parse st as $\mathsf{st} = (\mathsf{sk}, \{\mathsf{st}_i, J_i\}_{i\in[\ell]}, \{\hat{\mathsf{st}}_{i,j}\}_{i\in[\ell],j\in[t]})$
3. For $i = 0, \ldots, \ell t$:
   - If $i \in K$: Set $m_{b_i,i} \leftarrow m_i$ and $m_{1-b_i,i} \leftarrow 0$
   - If $i \notin K$: Set $m_{0,i} \leftarrow 0$ and $m_{1_i,i} \leftarrow 0$
4. Set $\vec{m}_0 \leftarrow (m_{0,1}, \ldots, m_{0,\ell t})$ and $\vec{m}_1 \leftarrow (m_{1,1}, \ldots, m_{1,\ell t})$.
5. Compute $(\mathsf{copir}_{S,i}, \vec{y}_i = (y_{i,1}, \ldots, y_{i,m})) \xleftarrow{\$} \mathsf{coPIR.Send}(\mathsf{copir}_{R,i})$.
6. For $i \in [\ell]$ and $j \in [t]$: Set $y'_{i,J_i[j]} \leftarrow y_{i,J_i[j]} + (m_{1,i,J_i[j]} - m_{0,i,J_i[j]})$, set $\overline{\mathsf{DB}}_{i,j} = (0, \ldots, 0, y'_{i,J_i[j]}, 0, \ldots, 0)$, and compute $\vec{r}_{i,j} \xleftarrow{\$} \mathsf{PIR.Send}(\mathsf{DB}_{i,j}, q_{i,j})$.
7. Compute $\tilde{\mathsf{ct}} \xleftarrow{\$} \mathcal{LHE}.\mathsf{Sim}^{\mathsf{priv\text{-}dec}}_{\mathcal{LHE}}(1^\lambda, \mathsf{pk}, k, K, (m_s + z'_s)_{s\in K})$, where
$$z'_{j'} \coloneqq \begin{cases} y'_{i\cdot\ell+j} & \text{if } j' = J_i[j] \text{ (where } (i,j) \in [\ell]\times[t]) \\ y_{i\cdot\ell+j'} & \text{otherwise} \end{cases}.$$
8. Output $\mathsf{ots} = (\tilde{\mathsf{ct}}, \{\mathsf{copir}_{S,i}\}_{i\in[\ell]}, \{\vec{r}_{i,j}\}_{i\in[\ell],j\in[t]})$.

The sequence of hybrids used to show indistinguishability between the real and the ideal worlds is the same as in the proof of [BBDP22, Theorem 3]:

1. Start with the real experiment.
2. For each $(i,j) \in [\ell]\times[t]$, by sender security of PIR we can set $\overline{\mathsf{DB}}_{i,j}$ to 0 everywhere except for $J_i[j]$.
3. For each $i = 1, \ldots, \ell$, by sender security of coPIR we can replace $y'_{i,J_i[j]} \leftarrow y_{i,J_i[j]} + (m_{1,i,J_i[j]} - m_{0,i,J_i[j]})$ (in such a way that $\overline{\mathsf{DB}}_{i,j} = (0, \ldots, 0, y'_{i,J_i[j]}, 0, \ldots, 0)$).
4. We can replace $\hat{\mathsf{ct}}$ with $\tilde{\mathsf{ct}} \xleftarrow{\$} \mathsf{Sim}^{\mathsf{priv\text{-}dec}}_{\mathcal{LHE}}(1^\lambda, \mathsf{pk}, k, K, (m_i)_{i\in K})$ by applying lemma 10.

□

By combining lemmas 11 and 12 and theorem 12 we obtain corollary 7.

Corollary 7. *Assume the* QR *assumption and the binary LPN assumption* $\mathsf{LPN}(\mathsf{dim}, \mathsf{num}, \rho)$ *with dimension* $\mathsf{dim} = \mathsf{poly}(\lambda)$*, number of samples* $\mathsf{num} = \mathsf{dim}^c$ *(for any constant* $c > 1$*), and noise rate* $\rho = \mathsf{num}^{\varepsilon-1}$ *(for some constant* $\varepsilon < 1$*). Then for any* $\ell = \ell(\lambda)$*, there exists a decomposable two-round batch oblivious transfer for batch size* $k = \ell \cdot \mathsf{num}$ *where*

- *The receiver message* otr *has size* $(\ell^2 \cdot \mathsf{dim} + \ell \cdot \mathsf{num}^\varepsilon) \cdot \mathsf{poly}(\lambda) + k$
- *The sender message* $\mathsf{ots} = (\mathsf{ots}^\star, \mathsf{ots}^{\mathsf{dec}})$ *has size* $|\mathsf{ots}^\star| = (\mathsf{num} + \ell \cdot \mathsf{num}^\varepsilon) \cdot \mathsf{poly}(\lambda)$ *and* $|\mathsf{ots}^{\mathsf{dec}}| = k$*.*

In particular, for appropriate parameters (sufficiently large ℓ*, and* num *sufficiently larger than* ℓ*),* $|\mathsf{otr}| = k + o(k)$*, and* $|\mathsf{ots}^\star| = o(k)$*.*

Chapter 7

Breaking the Multi-Party Barrier for Sublinear-Secure Computation, without FHE

This chapter describes results which have been communicated previously in [BCM23].

Based on joint work with Elette Boyle and Geoffroy Couteau.

In this chapter we show how our two approaches from chapters 4 and 6 for breaking the circuit-size barrier, each one seemingly stuck at the two-barrier, can be combined to yield sublinear-communication *multiparty* computation.

Our high-level approach centers around Function Secret Sharing (FSS) [BGI15], a form of secret sharing where the secret object and shares comprise succinct representations of *functions*. More concretely, FSS for function class $\mathcal{F}$ allows a client to split a secret function $f \in \mathcal{F}$ into function shares $f_1, \ldots, f_N$ such that any strict subset of f_i's hide f, and for every input x in the domain of f it holds that $\sum_{i=1}^{N} f_i(x) = f(x)$. (This can be seen as the syntactic dual of HSS, where the role of input and function are reversed; we refer the reader to e.g. [BGI+18] for discussion.[1]) N-party FSS/HSS for sufficiently rich function classes is known to support low-communication N-party secure computation, but lack of multi-party FSS constructions effectively leaves us stuck at $N = 2$.
The core conceptual contribution of this work is the following simple framework, which enables us to achieve $(N + 1)$-party secure computation by using a form of FSS for only N parties.

Proposition 1 ($(N + 1)$-PC from N-FSS framework, informal)**.** *For any ensemble of polynomial-size circuits $\mathcal{C} = \{\mathcal{C}_\lambda\}$, consider an N-party FSS scheme for the class of "partial evaluation" functions $\{C_\lambda(\cdot, x_1, \ldots, x_N)\}_{\lambda, x_1, \ldots, x_N}$, and define the following sub-computation functionalities:*

- $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}$*: N-party secure FSS share distribution, where each party P_i holds input x_i (and λ), and learns the ith FSS key f_i for the function $C_\lambda(\cdot, x_1, \ldots, x_N)$.*
- $\mathcal{F}_{\mathsf{OE}}^{\mathsf{FSS}}$*: Two-party oblivious FSS evaluation, where party P_i holds an FSS key f_i, party P_0 holds input x_0, and P_0 learns the ith output $f_i(x_0)$.*

Then there exists a $(N+1)$-party protocol for securely computing $\mathcal{C}$ making one call to $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}$ and N calls to $\mathcal{F}_{\mathsf{OE}}^{\mathsf{FSS}}$.

[1] Indeed, we will refer to both notions, using each when more conceptually convenient.

Once expressed in this form, the resulting $(N+1)$-party protocol becomes an exercise: Roughly, it begins by having parties $1, \ldots, N$ jointly execute $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}$ on their inputs $x_1, \ldots, x_N$ to each receive a function share f_i of the secret function $f(x_0) := C_\lambda(x_0, x_1, \ldots, x_N)$, and then each run a pairwise execution of $\mathcal{F}_{\mathsf{OE}}^{\mathsf{FSS}}$ together with the remaining party P_0 in order to obliviously communicate the ith output share $f_i(x)$. Given these shares, P_0 can compute the final output as $\sum_{i=1}^{N} f_i(x_0)$. (See the Technical Overview for more detailed discussion.)
The communication of the resulting protocol will be dominated by the executions of $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}, \mathcal{F}_{\mathsf{OE}}^{\mathsf{FSS}}$. Of course, the technical challenge thus becomes if and how one can construct corresponding FSS schemes which admit secure share distribution and oblivious evaluation with *low communication*.

Instantiating the framework. We demonstrate how to instantiate the above framework building from known constructions of Homomorphic Secret Sharing (HSS) combined with a version of low-communication PIR.
We first identify a structural property of an FSS scheme which, if satisfied, then yields a low-communication procedure for oblivious share evaluation, through use of a certain notion of "correlated" (batch) Symmetric Private Information Retrieval (SPIR). Loosely, correlated SPIR corresponds to a primitive where a client wishes to make *correlated* queries into m distinct size-S databases held by a single server. Without correlation between queries, the best-known PIR constructions would require $m \cdot \mathsf{polylog}(S)$ communication. However, it was shown in [BCM22] that if the m index queries (each $\log S$ bits) are given by various subsets of a fixed bit string of length $n \ll m \log S$ held by the client, then (using the rate-1 batch OT constructions from [BBDP22]) this batch SPIR can be performed with significantly lower communication.
We then demonstrate that FSS schemes with the necessary structural property can be realized from existing constructions of HSS. Loosely speaking, the FSS evaluation procedure will be expressible as a polynomial (which depends on $x_1, \ldots, x_N$) evaluated on the final input x_0, and the HSS will enable the N parties to compute additive secret shares of the coefficients of this corresponding polynomial.
We further extend the approach to support an underlying HSS scheme satisfying only a weaker notion of correctness, with *inverse-polynomial* (Las Vegas) *error*. In such scheme, homomorphic evaluation may fail with noticeable probability (over the randomness of share generation), in a manner identifiable to one or more parties. This is the notion satisfied by the 2-party HSS constructions from Decisional Diffie-Hellman [BGI16a], or Learning With Errors with only a polynomial-size modulus [DHRW16, BKS19]. This error must be removed in our construction while incurring minimal additional interaction. We demonstrate how to do so, using (standard) Private Information Retrieval [CGKS95] and punctured pseudorandom functions [BW13, KPTZ13, BGI14]. Note that the former is implied by correlated SPIR, and the latter implied by any one-way function, so that these tools do not impose additional assumptions in the statement below.

Theorem 13 (Sublinear MPC, informal)**.** *For any ensemble of polynomial-size circuits $\mathcal{C} = \{\mathcal{C}_\lambda\}$ of size s, depth $\log\log s$, and with n inputs and m outputs, if there exists the following:*

- *Correlated Symmetric Batch PIR, for m size-s databases where queries come from n bits, with communication $O(n+m+\mathsf{poly}(\lambda)+\mathsf{comm}(s))$ for some function comm.*
- *(Las Vegas) N-party Homomorphic Secret Sharing with compact shares (size $O(n)$ for input size n), for the class of $\log\log$-depth boolean circuits.*

Then there exists a secure $(N+1)$-party computation protocol for $\mathcal{C}$ with communication $O(n + m + \mathsf{poly}(\lambda) + N \cdot \mathsf{comm}(s))$. In particular, sublinearity is achieved when $N \cdot \mathsf{comm}(s) \in o(s)$.

Remark 4 (Compiling Sublinear MPC from Passive to Active Security)**.** *In this work, we focus on security against semi-honest adversaries. However, all our results extend immediately to the malicious setting, using known techniques. Indeed, to get malicious security while preserving sublinearity, one can just use the seminal GMW compiler [GMW87a] with zero-knowledge arguments, instantiating the ZKA with (interactive) succinct arguments [NN01]. Using Kilian's PCP-based 4-move argument [Kil92], which has polylogarithmic communication, this can be done using any collision-resistant hash function. The latter are implied by all assumptions under which we base sublinear MPC, hence our results generalise directly to the malicious setting. This observation was made in previous works on sublinear-communication secure computation (e.g. [BGI16a, CM21, BCM22]).*

Remark 5 (Beyond Boolean circuits)**.** *The above approach can be extended to arithmetic circuits over general fields $\mathbb{F}$, by replacing the correlated SPIR with an analogous form of (low-communication)* correlated oblivious polynomial evaluation *(OPE). We discuss and prove this more general result in the main body, but focus here on the Boolean setting, as required instantiations of such correlated-OPE beyond constant-size fields are not yet currently known.*

Resulting constructions. Finally, we turn to the literature to identify constructions of the required sub-tools, yielding resulting sublinear secure computation results from various mathematical structures and computational assumptions.

Corollary 8 (Instantiating the framework, informal)**.** *There exists secure 3-party computation for evaluating Boolean circuits of size s and depth $\log \log s$ with n inputs and m outputs, with communication complexity $O(n + m + \sqrt{s} \cdot \mathsf{poly}(\lambda) \cdot (n + m)^{2/3})$ based on the Learning Parity with Noise (LPN) assumption for any inverse-polynomial error rate, together with any of the following additional computational assumptions:*

- *Decisional Diffie-Hellman (DDH)*
- *Learning with Errors with polynomial-size modulus (poly-modulus LWE)*
- *Quadratic Residuosity (QR) + superpolynomial LPN*[2]

This can be extended under the same assumptions to secure 3-party computation of general "layered" (in fact, only locally synchronous[3]*) circuits of depth d and size s with communication $O(s/\log\log s + d^{1/3} \cdot s^{2(1+\epsilon)/3} \cdot \mathsf{poly}(\lambda))$, for arbitrary small constant ϵ. The latter is sublinear in s whenever $d = o(s^{1-\epsilon}/\mathsf{poly}(\lambda))$, i.e., the circuit is not too "tall and skinny."*
If we further assume the existence of a constant-locality PRG with some polynomial stretch and the super-polynomial security of the Decisional Composite Residuosity (DCR) assumption, then the above extends to the 5-party setting, both for loglog-depth boolean circuits and for layered boolean circuits.

[2]Superpolynomial hardness of LPN with a small inverse-superpolynomial error rate, but few samples, as assumed in [CM21].

[3]Recall from chapter 1 that a circuit is *layered* [GJ11] if all gates and inputs are arranged into layers, such that any wire only connects one layer to the next, but each input may occur multiple times at different layers. A layered circuit is *locally synchronous* [Bel84] if each input occurs exactly once (but at an arbitrary layer). A locally synchronous circuit is *synchronous* [Har77] if all inputs are in the first layer.

More concretely, the required notion of correlated SPIR was achieved in [BCM22], building on [BBDP22], from a selection of different assumptions. The required HSS follows for $N = 2$ from DDH from [BGI16a], LWE with polynomial-size modulus from [DHRW16, BKS19], DCR from [OSY21, RS21], and from superpolynomial LPN from [CM21]. It holds for $N = 4$ from DCR from [**?**] (with some extra work, complexity leveraging, and restrictions; see technical section). Note that combining the works of [BBDP22, OSY21] seems to implicitly yield rate-1 batch OT from DCR, and in turn correlated SPIR [BCM22]: if true, the assumptions for sublinear-communication five-party MPC can be simplifed to constant-locality PRG, LPN, and superpolynomial DCR (without the need for DDH, LWE, or QR). Since this claim was never made formally, we do not use it.
A beneficial consequence of our framework is that future developments within these areas can directly be plugged in to yield corresponding new constructions and feasibilities.

7.1 Overview of this Chapter's Results

General framework. Recall the secure computation framework via homomorphic secret sharing (HSS). Given access to an N-party HSS scheme supporting homomorphic evaluation of the desired circuit C, the parties begin by jointly HSS-sharing their inputs via a small secure computation. Each party can then homomorphically evaluate the circuit C on its respective HSS share without interaction, resulting in a short output share that it exchanges with all other parties. The parties can then each reconstruct the desired output by combining the evaluated shares (for standard HSS, this operation is simply addition). The resulting MPC communication cost scales only with the complexity of HSS share generation plus exchange of (short) output shares, but remains otherwise independent of the complexity of C.
In theory, this approach provides sublinear secure computation protocols for any number of parties N. In practice, however, we simply do not have HSS constructions for rich function classes beyond $N = 2$ with security against collusion of two or more corrupted parties, crucial for providing the corresponding MPC security. This remains a standing open question that has received notable attention, and unfortunately seems to be a challenging task.
A natural question is whether the above framework can somehow be modified to extend beyond the number of parties N supported by the HSS, for example to $N' = N + 1$. The issue with the above approach is that parties cannot afford to secret share their input to any N-subset in which they do not participate, as all parties within this subset may be corrupt, in which case combining all HSS shares reveals the shared secrets.
Instead, suppose that only the N parties share their inputs amongst each other. In this case, there is no problem with all N shareholding parties being corrupt, as this reveals only their own set of inputs. But, we now have a challenge: how to involve the final party's input into the computation?
In the HSS framework, parties each homomorphically evaluated the public C on shares. Suppose, on the other hand, the HSS supports homomorphic evaluation of the *class* of functions $C_{x_0} := C(x_0, \cdot, \ldots, \cdot)$. Or, more naturally, consider a dual view: Where the N parties collectively generate shares of a secret *function* $C(\cdot, x_1, \ldots, x_N)$ with their inputs hardcoded, which accepts a single input x_0 and outputs $C(x_0, \ldots, x_N)$. That is, using *function* secret sharing (FSS).
Of course, normally in FSS we think of the input on which the function is to be evaluated (in this case, x_0) as a public value, which each shareholder will know. Here,

this clearly cannot be the case. Instead, we consider a modified approach, where each of the N FSS shareholders will perform a pairwise *oblivious evaluation* procedure, with the final $(N+1)$st party P_0. That is, the ith shareholder holds the ith function key FSS k_i, which defines a share evaluation function "f_i"= $\mathsf{FSS.Eval}(i, k_i, \cdot)$. As a result of the oblivious evaluation, party P_0 will learn the evaluation $y_i = \mathsf{FSS.Eval}(i, k_i, x_0)$ of this function on its secret input x_0, and neither party will learn anything beyond this; in particular, P_0 does not learn k_i, and P_i does not learn x_0. At the conclusion of this phase, party P_0 learns exactly the set of N output shares, and can reconstruct the final output $C(x_0, \cdots, x_N) = y_1 + \cdots + y_N$ and send to all parties.
The corresponding high-level protocol template is depicted in Figure 7.1. Here, $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}$ represents an ideal N-party functionality for N-FSS share generation (defined formally in Figure 7.2 of Section 7.2), where each party provides its input x_i and receives its FSS share k_i. $\mathcal{F}_{\mathsf{OE}}^{\mathsf{FSS}}$ represents an ideal two-party functionality for oblivious FSS share evaluation (defined formally in Figure 7.3 of Section 7.2), where P_i and P_0 respectively provide inputs k_i and x_0, and P_0 learns the evaluation $\mathsf{FSS.Eval}(i, k_i, x_0)$.

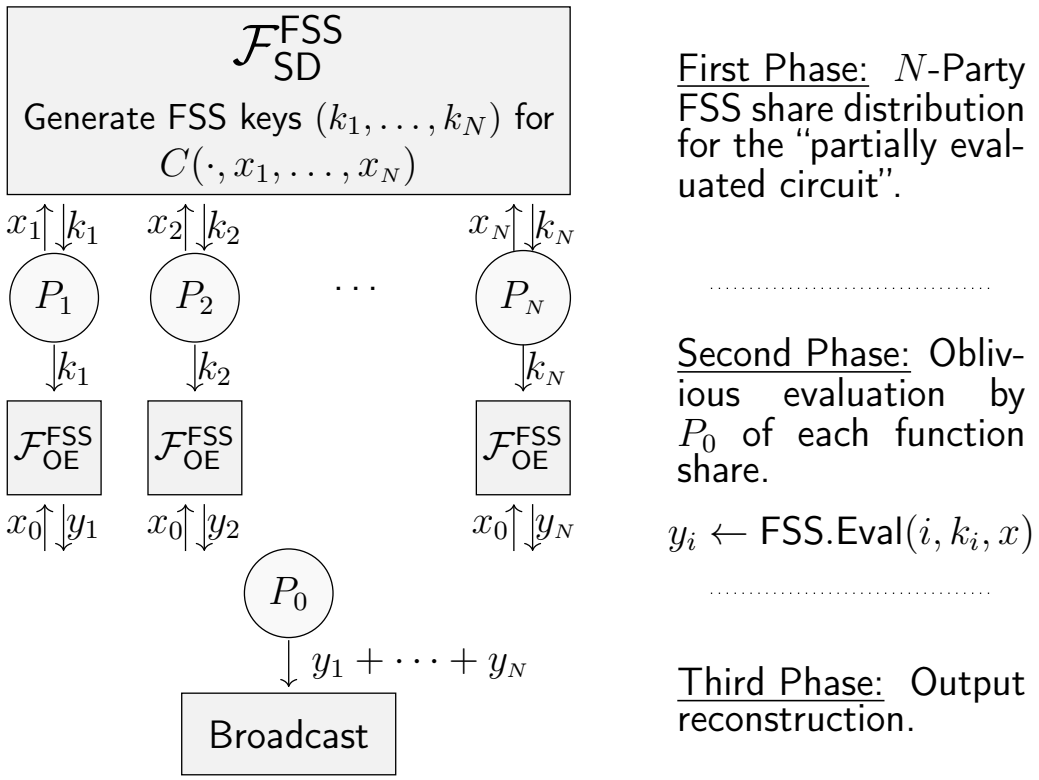


Figure 7.1: Template for $(N+1)$-party sublinear secure computation of C from N-party additive FSS.

Consider the (passive) security of the proposed scheme against up to N corruptions. If the corrupted parties are (any subset of) those holding FSS shares, then since the parties execute a secure computation for share generation, their view is restricted to a subset of FSS key shares $(k_i)_{i \in T}$, which hides any honest parties' inputs $(x_i)_{i \in [N] \setminus T}$ by the security of the FSS. (Note if all N shareholding parties are corrupt, then this statement holds vacuously, as no honest parties' inputs were involved.) If the corrupted parties include P_0 together with a (necessarily *strict*) subset of FSS shareholders, then their collective view consists of a strict subset of FSS keys $(k_i)_{i \in T}$ together with evaluated output shares $(y_i)_{i \in [N]}$. However, the security of the FSS combined with the additive reconstruction of output shares implies this reveals nothing beyond the func-

tion output.[4]
Now, in order for this framework to provide low communication, it must be the case that we have an FSS scheme for the relevant partial-evaluation function class $\{f_{\alpha_1,\ldots,\alpha_N} = C(\cdot, \alpha_1, \ldots, \alpha_N)\}$, for which the following two steps can be performed succinctly:

- Secure N-party FSS share generation, and
- Oblivious evaluation by P_0 of each function share.

We next address approaches for how each of these pieces can be achieved.

Oblivious evaluation for "Loglog-depth" FSS via PIR. Consider first the pairwise oblivious FSS evaluation procedure, where P_0 holds x_0, party P_i holds FSS key k_i, and P_0 should learn $\mathsf{FSS.Eval}(i, k_i, x_0)$.
Since this is reduced to a 2-party functionality, a natural first place to look would be for FSS schemes where $\mathsf{FSS.Eval}(i, k_i, \cdot)$ is within a function class already admitting low-communication *2-party* secure computation. Unfortunately, this is more challenging than it sounds. While sublinear-communication 2PC exists for general layered circuits from a variety assumptions, recall that the sublinearity will be here in the complexity not of C, but of $\mathsf{FSS.Eval}$, almost certainly a more complex computation.
Indeed, the idea of increasing the number of parties by homomorphically evaluating an $\mathsf{HSS.Eval}$ has previously been considered in the related setting of HSS, and hit similar limitations. For example, relatively strong HSS schemes based on DDH or DCR support homomorphic evaluation (and thus secure computation with very low communication) of NC^1; but, the corresponding operations required to actually *compute* $\mathsf{HSS.Eval}$ itself lies outside of NC^1. In [BGI17], this was addressed by instead securely computing a (low-depth) *randomized encoding* of the evaluation operation, effectively squashing the depth of the computation to be securely performed. This enabled them to achieve low round complexity, but resulted in large communication (scaling with the size of the entire $\mathsf{HSS.Eval}$ circuit). Recently, it was shown by Chillotti et al. [**?**] that for the specific DCR-based HSS construction of [OSY21,RS21], $\mathsf{HSS.Eval}$ for homomorphically evaluating a constant-degree computation can be computed within NC^1. However, this only gives low-communication secure computation for constant-degree functions, which will not suffice for overall sublinearity.
Instead, we take a different approach, going beyond black-box use of existing sublinear 2PC results. While the full $\mathsf{FSS.Eval}(i, k_i, x_0)$ computation itself may be complex, suppose it is the case that it can be decomposed into two parts: (1) some form of precomputation, depending only on i and k_i, followed by (2) computation on x_0, which is of low complexity. More concretely, consider the output of part (1) to be a new circuit C_{Eval} whose input is x_0 and output is $\mathsf{FSS.Eval}(i, k_i, x_0)$, and suppose it is the case that C_{Eval} has low $\log\log(s)$ depth (where s is the size of the original circuit C the parties wish to compute in the MPC). Note that while C_{Eval} has low depth, its identity depends on the secret k_i (of P_i), so that black-box secure computation of C_{Eval} does not apply.
On the other hand, opening the box of one such recent secure computation protocol, we identify that an intermediate tool developed actually has stronger implications. The tool is *correlated batch symmetric PIR*, for short correlated SPIR [BCM22], which as

[4]Note that in fact we do not need FSS with *additive* reconstruction, but rather any form of reconstruction will suffice, as long as the output shares provide this property of revealing nothing beyond the function output. We formalize this property, and prove it holds for additive reconstruction, in lemma 13.

discussed above, enables low-communication of several batched instances of (single-server) SPIR whose queries are correlated. In this case, the m "databases" will be defined implicitly by the m output bits of the circuit C_{Eval}. Because C_{Eval} is $\log\log s$ depth as a function of its input x_0 (and circuits are taken to be fan-in 2), each computed output bit depends on at most $\log s$ bits of x_0, and as such can be represented as a size-s database indexed by the corresponding $\log s$ input bits. Oblivious evaluation of C_{Eval} on x_0 can then be achieved by P_0 making m batch queries into these databases, where the collective query bits are all derived from various bits of the single string x_0.
As a brief aside: Extending to larger arithmetic spaces, the role of correlated SPIR here can be replaced by an analogous version of correlated Oblivious Polynomial Evaluation (OPE). Here, a $\log\log s$ depth arithmetic circuit C_{Eval} can be expressed as a secret multivariate polynomial in x_0 of size $\mathsf{poly}(s)$, where each monomial depends on at most $\log s$ elements of the arithmetic vector x_0. Unfortunately, we are not presently aware of tools for achieving low-communication correlated OPE beyond constant-size fields. However, we include this in the technical exposition, in case such techniques are later developed. We note that the final steps in our instantiation (described in the following) do hold over larger arithmetic spaces under certain computational assumptions.

"Loglog-depth" FSS from HSS. Consider an ensemble $\mathcal{C} = \{\mathcal{C}_\lambda\}$ of Boolean circuits of size s and depth $\log\log s$. The remaining goal is to obtain FSS for the corresponding class of partial-evaluation functions $\{C_\lambda(\cdot, x_1, \ldots, x_N)\}$ for which the FSS evaluation C_{Eval} is $(\log\log s)$-depth, as discussed above.
From the structure of C_λ, the evaluation of $C_\lambda(x_0, \ldots, x_N)$ on *all* inputs can be expressed as a $\mathsf{poly}(s)$-size multivariate polynomial in the bits $x_i[j]$ of the x_i, where each monomial is of degree at most $\log s$. When viewed as a function of just x_0, we thus have $\mathsf{poly}(s)$-many monomials in the bits of x_0 whose coefficients p_j are each formed by the product of at most $\log s$ bits from the inputs $x_1, \ldots, x_N$. That is, $\sum_j p_j \prod_{\ell \in S_j} x_0[\ell]$, where each $|S_j| \leq \log s$ is a publicly known set.
If the N parties can somehow produce *additive secret shares* $\{p_j^{(i)}\}_{i\in[N]}$ of each one of these coefficients p_j, then this would constitute the desired FSS evaluation: Indeed, the ith share evaluation $\mathsf{FSS.Eval}(i, k_i, x_0)$ would be computable as $y^{(i)} = \sum_j p_j^{(i)} \prod_{\ell \in S_j} x_0[\ell]$, satisfying $\sum_{i=1}^N y^{(i)} = \sum_j p_j \prod_{\ell \in S_j} x_0[\ell] = C_\lambda(x_0, \ldots, x_N)$. Further, each $\mathsf{FSS.Eval}(i, k_i, \cdot)$ is expressible as a $(\log\log s)$-depth circuit in x_0—as required from the previous discussion.
The question is how to *succinctly* reach a state where the N parties hold these coefficient secret shares. Of course, direct secure computation is not an option, as even the output size is large, $\mathsf{poly}(s)$. However, this is not a general computation. Suppose we have access to an *HSS* scheme supporting homomorphic evaluation of $\log\log s$ depth operations. Such constructions are known to exist from a variety of assumptions (as discussed after Corollary 8). Then, if the parties HSS share their respective inputs $x_1, \ldots, x_N$, they can *locally* evaluate additive shares of the corresponding $(\log s)$-products p_j.
The corresponding $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}$ operation will thus correspond to the $\mathsf{HSS.Share}$ procedure of the HSS scheme on the parties' collective inputs. If the HSS scheme has a compact sharing procedure, then this will be computable with sufficiently low communication. Note that vanilla usage of some HSS schemes will not provide the required compactness (e.g., including structured ciphertexts of the input bits); however, using standard hybrid encryption tricks this can be facilitated.

"Loglog-depth" FSS from Las Vegas HSS. An additional challenge arises, however, when the underlying HSS scheme we attempt to use provides correctness only up to *inverse-polynomial error*. This is the case, for example, in known 2-party HSS schemes for NC^1 from DDH [BGI16a] or from LWE with polynomial-size modulus [DHRW16, BKS19]. In these schemes, the inverse-polynomial error rate δ can be chosen as small as desired, but shows up detrimentally as $1/\delta$ in other scheme parameters (runtime for the DDH scheme; modulus size for LWE).

This means with noticeable probability, the shares of at least one of the coefficients p_j from above will be computed incorrectly. Even worse, as typical in these settings, the parties cannot learn or reveal where errors truly occurred, as this information is dependent on the values of the secret inputs. This remains a problem even if the HSS scheme is "Las Vegas," in the sense that for every error at least one of the parties will identity that a potential-error event has occurred (i.e., will evaluate output share as $\perp$). Even then, the flagging party must not learn whether an error truly took place, and the other party must not learn that a potential error was flagged.

We present a method for modifying the HSS-based FSS sharing procedure from above, to remove the error in the required homomorphic evaluations, while hiding from the necessary parties where these patches took place. We focus on the 2-party case, and further assume the HSS has a succinct protocol (communication linear in the input size, up to an additive $\mathsf{poly}(\lambda)$ term) for distributing the shares of the HSS, where homomorphic evaluation can take place across different sets of shared values. This is the case for known Las Vegas HSS schemes.

This procedure can be viewed as a modification to either the Share or Eval portion of the FSS. By viewing it as part of $\mathsf{FSS.Share}$, we automatically fit into the framework of the previous sections. Namely, this can be viewed as a new $\mathsf{FSS.Share}$ (or $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}$) procedure with relatively large computational complexity (comparable to the truth table of the shared function), but which we show *admits a low-communication secure computation procedure*. We describe the sharing procedure directly via the achieving protocol; the corresponding $\mathsf{FSS.Share}$ procedure can be inferred.

First, note that by taking the inverse-polynomial error rate δ to be sufficiently small, we can guarantee with high probability that the total number of potential-error flags $\perp$ obtained by any party is at most the security parameter, λ. The sharing protocol begins by HSS sharing the inputs $(s_0, s_1) \leftarrow \mathsf{HSS.Share}(x_1, x_2)$ as usual. Then, each party homomorphically evaluates all required values corresponding to shares of each of the coefficients p_j. For each party P_i ($i \in \{1, 2\}$), denote these values in an array T_i, which contains at most λ positions in which $T_i[j] = \perp$. For each such position j^*, $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}$ sets $T_i[j^*] = 0$, and must now "patch" the missing value. Consider this procedure for party P_1 (P_2 will be reversed).

In order to compute the correct output (i.e., coefficient p_j) in this position, the parties run a small-scale secure protocol that HSS shares the *index* position j^* of each $\perp$ symbol of P_1. This enables them to homomorphically re-evaluate shares of the corresponding coefficient term p_{j^*}, in a way that hides the index j^* from P_2 (note that this computation, with index selection, remains within NC^1). In fact, by re-evaluating this computation λ-many times, then with overwhelming probability, at least one is error-free. By running a small-scale secure computation on these shares, we can assume that the parties hold additive shares of the correct value p_{j^*}.

It would seem the remaining step is for P_1 to somehow learn the correct value p_{j^*} offset by P_2's share $T_2[j^*]$, while keeping j^* hidden from P_2. However, the situation is somewhat more sticky. The problem is that in the original HSS evaluation, P_1 learns not only $\perp$, but also a candidate output share. By receiving the *correct* output share

$(p_{j^*} - T_2[j^*])$, party P_1 would learn whether or not an error actually occurred, leaking sensitive information. This means that inherently, P_2 must also modify its share in position j^* as part of the correction procedure. But, this must be done in a way that both hides the identity of j^*, and also does *not* affect the secret sharing across the two parties in other positions.

This will be done in two pieces: (1) P_1 will learn $(T_2[j^*] - r)$, for some secret mask r chosen by P_2; and (2) they will both perform some operation on their local T_i array that offsets the value shared in position j^* by exactly $(p_{j^*} - T_2[j^*])$ while preserving the values shared in all other $j' \neq j^*$.

The first of these tasks can be performed by executing a standard single-server polylogarithmic symmetric PIR protocol, where P_1 acts as client with query index j^*, and P_2 acts as server with the *r-shifted* database $T_2'[j] = T_2[j] - r$, for random r of its choice.

The second task will be performed by a low-communication private increment procedure using *distributed point functions* (DPF): namely, FSS for the class of point functions (equivalently, compressed secret shares of a secret unit vector). Actually, since party P_1 knows the identity of j^*, a weaker tool of punctured PRFs suffice; however, we continue with DPF terminology for notational convenience (both are implied by one-way functions). More concretely, the parties will run a small-scale secure computation protocol on inputs $j^*, (T_2[j^*] - r)$ (held by P_1), the additive shares of p_{j^*}, and r (held by P_2), which outputs short DPF key shares k_1, k_2 to the respective parties, with the property that $\mathsf{DPF.Eval}(1, k_1, j) + \mathsf{DPF.Eval}(2, k_2, j) = 0$ for every $j \neq j^*$, and $= (p_{j^*} - T_2[j^*])$ for $j = j^*$. Each party thus modifies its T_i array by offsetting each position j with the jth DPF evaluation, yielding precisely the required effect.

This procedure is performed for every flag position j^*, and for each party P_1, P_2. (Note that the parties should always perform the above steps λ times, sometimes on dummy values, in order to hide the true number of flagged positions.) The final resulting scheme provides standard FSS correctness guarantees, *removing* the inverse-polynomial error, and thus can be plugged into the approach from above. As mentioned, the new resulting $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}$ functionality is now a complex procedure, with runtime scaling as the entire truth table size of the shared function. But, the above-described protocol provides a means for securely emulating $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}$ with *low communication*: scaling just as λ-many small-scale secure computations and PIR executions.

7.2 General Template for $(N+1)$-Party Sublinear Secure Computation from N-Party FSS

In this section we present a generic template for building $(N+1)$-party sublinear secure computation from an N-party additive function secret sharing scheme (for a well-chosen function class) with two specific properties. We require of the FSS scheme that there exist low-communication protocols to realise the following tasks:

- *N-Party Share Distribution:* N servers generate FSS shares of some function of their inputs; the ideal functionality $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}$ is provided in fig. 7.2.
- *Two-Party Oblivious Share Evaluation:* A client obliviously evaluates an FSS share held by a server; the ideal functionality $\mathcal{F}_{\mathsf{OE}}^{\mathsf{FSS}}$ is provided in fig. 7.3.

Theorem 14 proves that the protocol provided in fig. 7.5 is an $(N+1)$-party secure computation scheme in the $(\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}, \mathcal{F}_{\mathsf{OE}}^{\mathsf{FSS}})$-hybrid model. This template achieves sublinear secure computation provided $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}$ and $\mathcal{F}_{\mathsf{OE}}^{\mathsf{FSS}}$ can be realised with low enough communication. A high level overview of the protocol is provided in fig. 7.1.

7.2.1 Requirements of the FSS Scheme

We start by isolating in lemma 13 the properties we require of the FSS scheme to fit our template for sublinear secure computation, and show that they are satisfied by any additive FSS scheme. At a high level, we require that given a strict subset of the FSS keys, together with the evaluated output shares of all keys on some known input x, it should be computationally hard to recover any information about the secret shared function f beyond its evaluation $f(x)$. For the formalisation of this property, as well as the proof that additivity suffices, we refer to lemma 13.

Lemma 13. *Let* $\mathsf{FSS} = (\mathsf{FSS.Gen}, \mathsf{FSS.Eval})$ *be an* N*-party additive function secret sharing scheme for some function family* $\mathcal{F}$*, and let* $\mathsf{Sim}^{\mathsf{FSS}}$ *be a simulator defined as in definition 3. Then, the following holds:*

> ***Reconstruction-Only (given some keys, all shares, the input, and auxiliary information):*** *Let* $\mathsf{info}\colon \{0,1\}^\star \to \{0,1\}^\star$ *be any function which, on input the description* $\tilde{f}$ *of some function* $f \in \mathcal{F}$*, outputs some partial information* $\mathsf{info}(\tilde{f})$ *about* f*. For every set of corrupted parties* $\mathcal{D} \subsetneq [N]$*, there exists a probabilistic polynomial-time algorithm* $\mathsf{Sim}^{\mathsf{FSS}}_{\mathsf{rec,info}}$ *(a* simulator*), such that for every sequence* $\tilde{f}_1, \tilde{f}_2, \ldots$ *of polynomial-size function description of functions* $f_1, f_2, \cdots \in \mathcal{F}$*, for every sequence of inputs* $(x_\lambda)_{\lambda \in \mathbb{N}^\star} \in D_{f_1} \times D_{f_2} \times \ldots$*, the outputs of the following experiments* $\mathsf{Real}_{\mathsf{rec,info}}$ *and* $\mathsf{Ideal}_{\mathsf{rec,info}}$ *are computationally indistinguishable:*
>
> - $\mathsf{Real}_{\mathsf{rec,info}}(1^\lambda)$:
>   - $(k_1, \ldots, k_N) \xleftarrow{\$} \mathsf{Gen}(1^\lambda, \tilde{f}_\lambda)$;
>   - $y_i \leftarrow \mathsf{Eval}(i, k_i, x_\lambda)$ *for* $i \notin \mathcal{D}$;
>   - *Output* $(x_\lambda, (k_i)_{i\in\mathcal{D}}, (y_i)_{i\notin\mathcal{D}}, \mathsf{info}(\tilde{f}_\lambda))$.
> - $\mathsf{Ideal}_{\mathsf{rec,info}}(1^\lambda)$: *Output* $\mathsf{Sim}_{\mathsf{rec,info}}(1^\lambda, 1^n, 1^m, x_\lambda, f_\lambda(x_\lambda), \mathsf{info}(\tilde{f}_\lambda))$.

Proof. We first consider the intermediary step of "Function Secrecy (given some keys, and auxiliary information)".

1. **Function Secrecy (given some keys, and auxiliary information):** Let $\mathsf{info}\colon \{0,1\}^\star \to \{0,1\}^\star$ be any function which, on input the description $\tilde{f}$ of some function $f \in \mathcal{F}$, outputs some partial information $\mathsf{info}(\tilde{f})$ about f. For every set of corrupted parties $\mathcal{D} \subsetneq [N]$, there exists a probabilistic polynomial-time algorithm $\mathsf{Sim}^{\mathsf{FSS}}_{\mathsf{info}}$ (a *simulator*), such that for every sequence $\tilde{f}_1, \tilde{f}_2, \ldots$ of polynomial-size function description of functions $f_1, f_2, \cdots \in \mathcal{F}$ (we denote $n(\lambda)$ and $m(\lambda)$ respectively the input and output length of f_λ for $\lambda \in \mathbb{N}^\star$), the outputs of the following experiments $\mathsf{Real}_{\mathsf{info}}$ and $\mathsf{Ideal}_{\mathsf{info}}$ are computationally indistinguishable:

   - $\mathsf{Real}_{\mathsf{info}}(1^\lambda)$: $(k_1, \ldots, k_N) \xleftarrow{\$} \mathsf{Gen}(1^\lambda, \tilde{f}_\lambda)$; Output $((k_i)_{i\in\mathcal{D}}, \mathsf{info}(\tilde{f}_\lambda))$.
   - $\mathsf{Ideal}_{\mathsf{info}}(1^\lambda)$: Output $\mathsf{Sim}^{\mathsf{FSS}}_{\mathsf{info}}(1^\lambda, 1^N, 1^{n(\lambda)}, 1^{m(\lambda)}, \mathsf{info}(\tilde{f}_\lambda))$.

   And furthermore, such a simulator can be built as:

   $\mathsf{Sim}^{\mathsf{FSS}}_{\mathsf{info}}(1^\lambda, 1^N, 1^{n(\lambda)}, 1^{m(\lambda)}, \mathsf{info}(\tilde{f}_\lambda))$: Output $(\mathsf{Sim}^{\mathsf{FSS}}(1^\lambda, 1^N, 1^{n(\lambda)}, 1^{m(\lambda)}), \mathsf{info}(\tilde{f}_\lambda))$.

   The above is true because the notion of FSS security is hereditary in that any FSS scheme for some function class $\mathcal{F}$ is also an FSS scheme for any subclass $\mathcal{F}' \subseteq \mathcal{F}$, and furthermore the same simulator can be used. In particular this is true for $\mathcal{F}'$ obtained by quotienting $\mathcal{F}$ by the auxiliary information given to an adversary.

2. **Reconstruction-Only (given some keys, all shares, and the input):** We now prove the lemma. Let $\mathsf{info}\colon \{0,1\}^\star \to \{0,1\}^\star$ be any function which, on input the description $\tilde{f}$ of some function $f \in \mathcal{F}$, outputs some partial information $\mathsf{info}(\tilde{f})$ about f. Let info' be the function which, on input $\tilde{f}_\lambda$, outputs $\mathsf{info}'(\tilde{f}_\lambda) \coloneqq (x_\lambda, f_\lambda(x_\lambda))$. Let $\mathcal{D} \subsetneq [N]$, and let $\mathcal{D}^\star$ be any size-$(N-1)$ superset of $\mathcal{D}$ in $[N]$ (i.e. $\mathcal{D} \subseteq \mathcal{D}^\star \subsetneq [N]$, $|\mathcal{D}^\star| = N-1$). Denote u the unique element of $[N] \setminus \mathcal{D}^\star$. Let $(f_\lambda)_{\lambda\in\mathbb{N}^\star} \in \mathcal{F}^{\mathbb{N}^\star}$ and $(x_\lambda)_{\lambda\in\mathbb{N}^\star} \in (D_{f_1} \times D_{f_2} \times \dots)$. By function secrecy (given some keys, and auxiliary information), there exists a simulator $\mathsf{Sim}^{\mathsf{FSS}}_{(\mathsf{info}',\mathsf{info})}$ such that the following distributions Δ_1 and Δ_2 are computationally indistinguishable:

$$\begin{aligned}
\Delta_1 &\coloneqq \left\{((k_i)_{i\in\mathcal{D}^\star}, \mathsf{info}'(\tilde{f}_\lambda), \mathsf{info}(\tilde{f}_\lambda)) : (k_1,\dots,k_N) \xleftarrow{\$} \mathsf{Gen}(1^\lambda, \tilde{f}_\lambda)\right\} \\
\Delta_2 &\coloneqq \left\{\mathsf{Sim}^{\mathsf{FSS}}_{(\mathsf{info}',\mathsf{info})}(1^\lambda, 1^N, 1^{n(\lambda)}, 1^{m(\lambda)}, \mathsf{info}'(\tilde{f}_\lambda), \mathsf{info}(\tilde{f}_\lambda))\right\} .
\end{aligned}$$

Now consider the distributions Δ_3, Δ'_3, Δ_4, and Δ'_4 defined as:

$$\begin{aligned}
\Delta_3 &\coloneqq \left\{(x_\lambda, (k_i)_{i\in\mathcal{D}}, (y_i)_{i\in\mathcal{D}^\star\setminus\mathcal{D}}, \mathsf{info}(\tilde{f}_\lambda)) : \begin{array}{l}(k_1,\dots,k_N) \xleftarrow{\$} \mathsf{Gen}(1^\lambda, \tilde{f}_\lambda) \\ y_i \leftarrow \mathsf{Eval}(i, k_i, x_\lambda), i \in \mathcal{D}^\star \setminus \mathcal{D}\end{array}\right\} \\
\Delta'_3 &\coloneqq \left\{(x_\lambda, (k_i)_{i\in\mathcal{D}}, (y_i)_{i\notin\mathcal{D}}, \mathsf{info}(\tilde{f}_\lambda)) : \begin{array}{l}(k_1,\dots,k_N) \xleftarrow{\$} \mathsf{Gen}(1^\lambda, \tilde{f}_\lambda) \\ y_i \leftarrow \mathsf{Eval}(i, k_i, x_\lambda), \text{ for } i \notin \mathcal{D}\end{array}\right\}
\end{aligned}$$

$$\begin{aligned}
\Delta_4 &\coloneqq \left\{(\alpha, (k_i)_{i\in\mathcal{D}}, (y_i)_{i\in\mathcal{D}^\star\setminus\mathcal{D}}, \gamma) : \begin{array}{l}((k_i)_{i\in\mathcal{D}^\star}, (\alpha,\beta), \gamma) \xleftarrow{\$} \mathsf{Sim}^{\mathsf{FSS}}_{(\mathsf{info}',\mathsf{info})} \\ y_i \leftarrow \mathsf{Eval}(i, k_i, \alpha), \text{ for } i \in \mathcal{D}^\star \setminus \mathcal{D}\end{array}\right\} \\
\Delta'_4 &\coloneqq \left\{(\alpha, (k_i)_{i\in\mathcal{D}}, (y_i)_{i\notin\mathcal{D}}, \gamma) : \begin{array}{l}((k_i)_{i\in\mathcal{D}^\star}, (\alpha,\beta), \gamma) \xleftarrow{\$} \mathsf{Sim}^{\mathsf{FSS}}_{(\mathsf{info}',\mathsf{info})} \\ y_i \leftarrow \mathsf{Eval}(i, k_i, \alpha), \text{ for } i \in \mathcal{D}^\star \\ y_u \leftarrow \beta - \sum_{i\in\mathcal{D}^\star} y_i\end{array}\right\} .
\end{aligned}$$

Note that the only difference between Δ_3 and Δ'_3 (*resp.* Δ_4 and Δ'_4) is whether or not y_u is part of the output of the distribution (where $\{u\} = [N] \setminus \mathcal{D}^\star$). Because $\Delta_1 \overset{\text{c}}{\approx} \Delta_2$, the following algorithm cannot distinguish between Δ_1 and Δ_2: On input $((k_i)_{i\in\mathcal{D}^\star}, \mathsf{info}(\tilde{f}_\lambda) = (\alpha,\beta))$, set $y_i \leftarrow \mathsf{Eval}(i, k_i, \alpha)$ for $i \in \mathcal{D}^\star \setminus \mathcal{D}$, then output $(\alpha, (k_i)_{i\in\mathcal{D}}, (y_i)_{i\in\mathcal{D}^\star\setminus\mathcal{D}})$. This implies, by perfect correctness of the additive FSS scheme, that $\Delta_3 \overset{\text{c}}{\approx} \Delta_4$, which in turn implies that the following algorithm cannot distinguish between Δ_3 and Δ_4: On input$(\alpha, (k_i)_{i\in\mathcal{D}}, (y_i)_{i\in\mathcal{D}^\star\setminus\mathcal{D}})$, set $y_u \leftarrow \beta - \sum_{i\in\mathcal{D}^\star} y_i$, and output $(\alpha, (k_i)_{i\in\mathcal{D}}, (y_i)_{i\notin\mathcal{D}})$. Therefore, $\Delta'_3 \overset{\text{c}}{\approx} \Delta'_4$.

□

7.2.2 The Secure Computation Protocol

We define the ideal functionalities $\mathcal{F}^{\mathsf{FSS}}_{\mathsf{SD}}$ (fig. 7.2) for N-party FSS share distribution, and $\mathcal{F}^{\mathsf{FSS}}_{\mathsf{OE}}$ (fig. 7.3) for 2-party oblivious evaluation of FSS shares. We then introduce in fig. 7.5 the generic template for secure computation from additive FSS in the $(\mathcal{F}^{\mathsf{FSS}}_{\mathsf{SD}}, \mathcal{F}^{\mathsf{FSS}}_{\mathsf{OE}})$-hybrid model.

Functionality FSS Share Distribution $\mathcal{F}^{\mathsf{FSS}}_{\mathsf{SD}}$

Parameters: The ideal functionality $\mathcal{F}^{\mathsf{FSS}}_{\mathsf{SD}}$ is parameterised by a number of parties N, a function class $\mathcal{C} = \{f_{\alpha_1,\dots,\alpha_N}\}_{(\alpha_1,\dots,\alpha_N)\in\mathbb{F}^{\ell_1}\times\dots\times\mathbb{F}^{\ell_N}}$, and an additive FSS scheme $\mathsf{FSS} = (\mathsf{FSS.Gen}, \mathsf{FSS.Eval})$ for $\mathcal{C}$.

$\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}$ interacts with the N parties $P_1, \ldots, P_N$ in the following manner.

Input: Wait to receive $(\texttt{input}, i, x_i)$ where $x_i \in \{0,1\}^{\ell_i}$ from each party P_i (for $1 \leq i \leq N$).

Output: Run $(k_1, \ldots, k_N) \xleftarrow{\$} \mathsf{FSS.Gen}(1^\lambda, \tilde{f}_{x_1,\ldots,x_N})$, where $\tilde{f}_{x_1,\ldots,x_N}$ is a description of $f_{x_1,\ldots,x_N}$; Output k_i to each party P_i (for $1 \leq i \leq N$).

Figure 7.2: Ideal functionality $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}$ for the generation of FSS keys of a distributed function.

Functionality Oblivious Evaluation of FSS Shares $\mathcal{F}_{\mathsf{OE}}^{\mathsf{FSS}}$

Parameters: The ideal functionality $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}$ is parameterised by a number of parties N, and an additive FSS scheme $\mathsf{FSS} = (\mathsf{FSS.Gen}, \mathsf{FSS.Eval})$ for some function class $\mathcal{C}$.

$\mathcal{F}_{\mathsf{OE}}^{\mathsf{FSS}}$ interacts with two parties, Alice ("the client") and Bob ("the server"), in the following manner.

Input: Wait to receive $(\texttt{Client}, x)$ from Alice and $(\texttt{Server}, i, k_i)$ from Bob, and record (i, k_i, x).

Output: Run $y_i \leftarrow \mathsf{FSS.Eval}(i, k_i, x)$; Output y_i to Alice.

Figure 7.3: Ideal functionality $\mathcal{F}_{\mathsf{OE}}^{\mathsf{FSS}}$ for the two-party oblivious evaluation of FSS shares.

Functionality $\mathcal{F}_{\mathsf{SFE}}(C)$

The functionality is parameterised with a number N and an arithmetic circuit C with $n = \ell_0 + \ell_1 + \cdots + \ell_N$ inputs and m outputs over a finite field $\mathbb{F}$.

Input: Wait to receive $(\texttt{input}, i, x_i)$ from each party P_i $(0 \leq i \leq N)$, where $x_i \in \mathbb{F}^{\ell_i}$, and set $\vec{x} \leftarrow x_0 \| x_1 \| \ldots \| x_N$.

Output: Compute $\vec{y} \leftarrow C(\vec{x})$; Output $\vec{y}$ to all parties $P_0, P_1, \ldots, P_N$.

Figure 7.4: Ideal functionality $\mathcal{F}_{\mathsf{SFE}}(C)$ for securely evaluating an arithmetic circuit C among $N+1$ parties.

Protocol Π_C

Parties: $P_0, P_1, \ldots, P_N$

Parameters: The protocol is parameterised with a number of parties $(N+1)$, an arithmetic circuit $C\colon \mathbb{F}^n \to \mathbb{F}^m$ with $n = \ell_0 + \ell_1 + \cdots + \ell_N$, and an additive FSS scheme $\mathsf{FSS} = (\mathsf{FSS.Gen}, \mathsf{FSS.Eval})$ for the following function family of "partial evaluations of C":

$$\left\{ \begin{array}{rccl} g_{\alpha_1,\ldots,\alpha_N}\colon & \mathbb{F}^{\ell_0} & \to & \mathbb{F}^m \\ & x & \mapsto & C(x, \alpha_1, \ldots, \alpha_N) \end{array} : (\alpha_1, \ldots, \alpha_N) \in \mathbb{F}^{\ell_1} \times \cdots \times \mathbb{F}^{\ell_N} \right\} .$$

$(\mathsf{sid}_1, \ldots, \mathsf{sid}_N)$ are N distinct session ids.

Hybrid Model: The protocol is defined in the $(\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}, \mathcal{F}_{\mathsf{OE}}^{\mathsf{FSS}})$-hybrid model.

Input: Each party P_i holds input $x_i \in \mathbb{F}^{\ell_i}$.

The Protocol:

1. Each party P_i for $i \neq 0$ sends $(\texttt{input}, i, x_i)$ to $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}(C)$, and waits to receive k_i.
2. For each $i = 1, \ldots, N$:
   (a) Party P_0 sends $(\mathsf{sid}_i, \texttt{Client}, x_0)$ to $\mathcal{F}_{\mathsf{OE}}^{\mathsf{FSS}}(C)$ and P_i sends $(\mathsf{sid}_i, \texttt{Server}, i, k_i)$ to $\mathcal{F}_{\mathsf{OE}}^{\mathsf{FSS}}(C)$
   (b) Party P_0 waits to receive (sid_i, y_i) from $\mathcal{F}_{\mathsf{OE}}^{\mathsf{FSS}}(C)$.
3. Party P_0 sets $\vec{y} \leftarrow y_1 + \cdots y_N$, and sends $\vec{y}$ to all parties.
4. Every party outputs $\vec{y}$.

Figure 7.5: (Sublinear) secure computation protocol in the $(\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}, \mathcal{F}_{\mathsf{OE}}^{\mathsf{FSS}})$-hybrid.

Theorem 14 (Template for $(N+1)$-Party Sublinear MPC from N-Party FSS)**.** *Let $N \geq 2$. Let $C\colon \mathbb{F}^n \to \mathbb{F}^m$ be an arithmetic circuit with $n = \ell_0 + \ell_1 + \cdots + \ell_N$ inputs over a finite field $\mathbb{F}$, and let $\mathsf{FSS} = (\mathsf{FSS.Gen}, \mathsf{FSS.Eval})$ be an (additive) FSS scheme for the following function family of "partial evaluations of C":*

$$\left\{ \begin{array}{rccl} g_{\alpha_1,\ldots,\alpha_N}\colon & \mathbb{F}^{\ell_0} & \to & \mathbb{F}^m \\ & x & \mapsto & C(x, \alpha_1, \ldots, \alpha_N) \end{array} : (\alpha_1, \ldots, \alpha_N) \in \mathbb{F}^{\ell_1} \times \cdots \times \mathbb{F}^{\ell_N} \right\} .$$

The protocol Π_C provided in fig. 7.5 UC-securely implements the $(N+1)$-party functionality $\mathcal{F}_{\mathsf{SFE}}(C)$ in the $(\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}(C), \mathcal{F}_{\mathsf{OE}}^{\mathsf{FSS}}(C))$-hybrid model, against a static passive adversary corrupting at most N out of $(N+1)$ parties. The protocol uses $N \cdot m \cdot \log|\mathbb{F}|$ bits of communication, and additionally makes one call to $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}(C)$ and N calls to $\mathcal{F}_{\mathsf{OE}}^{\mathsf{FSS}}(C)$.

Proof. Let $\mathcal{A}$ be a semi-honest, static adversary that interacts with parties $P_0, P_1, \ldots, P_N$ running protocol Π_C in the $(\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}, \mathcal{F}_{\mathsf{OE}}^{\mathsf{FSS}})$-hybrid model. We construct in fig. 7.6 a simulator Sim such that no environment $\mathcal{Z}$ can distinguish with non-negligible probability whether it is interacting with $\mathcal{A}$ and Π_C in the $(\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}, \mathcal{F}_{\mathsf{OE}}^{\mathsf{FSS}})$-hybrid model, or with Sim and $\mathcal{F}_{\mathsf{SFE}}(C)$.

Simulator Sim

Let $\mathcal{D} \subsetneq [0, N]$ be the subset of statically corrupted parties, and let $\mathcal{H} \coloneqq [0, N] \setminus \mathcal{D}$ be the (non-empty) set of honest parties. Let $(x_i)_{i \in \mathcal{D}}$ be the set of inputs of the semi-honestly corrupt parties.

Requires: Simulator $\mathsf{Sim}^{\mathsf{FSS}}$ is defined as in definition 3. If P_0 is corrupted (i.e. $0 \in \mathcal{D}$), then let function info be defined on $\mathcal{C}$ as $\mathsf{info}(g_{\alpha_1,\ldots,\alpha_N}) \coloneqq (\alpha_i)_{i \in \mathcal{D} \setminus \{0\}}$, and let simulator $\mathsf{Sim}_{\mathsf{rec,info}}^{\mathsf{FSS}}$ be defined as in lemma 13, so that:

$$\left\{ (x_\lambda, (k_i)_{i \in \mathcal{D}}, (y_i)_{i \notin \mathcal{D}}, \mathsf{info}(\tilde{f}_\lambda)) \;:\; \begin{array}{l} (k_1, \ldots, k_N) \xleftarrow{\$} \mathsf{Gen}(1^\lambda, \tilde{f}_\lambda) \\ y_i \leftarrow \mathsf{Eval}(i, k_i, x_\lambda), \text{ for } i \notin \mathcal{D} \end{array} \right\}$$
$$\overset{\mathrm{c}}{\approx} \left\{ \mathsf{Sim}_{\mathsf{rec,info}}(1^\lambda, 1^n, 1^m, x_\lambda, f_\lambda(x_\lambda), \mathsf{info}(\tilde{f}_\lambda)) \right\} .$$

Simulation:

- **Simulating the communication with $\mathcal{Z}$:** Every input value that Sim receives from $\mathcal{Z}$ is written on $\mathcal{A}$'s input tape (as if coming from $\mathcal{A}$'s environment), and every output value written by $\mathcal{A}$ on its output tape is copied to Sim's own output tape (to be read by $\mathcal{Z}$).
- **Simulating the protocol's execution:**
  1. Sim activates $\mathcal{A}$ and receives the messages $(\mathtt{input}, i, x_i)_{i \in \mathcal{D} \setminus \{0\}}$ that $\mathcal{A}$ would send to $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}(C)$ (on behalf of the corrupted parties) in a real execution.
  2. If P_0 is honest, Sim sets
  $$(k_i)_{i \in \mathcal{D}} \xleftarrow{\$} \begin{cases} \mathsf{Sim}^{\mathsf{FSS}}(1^\lambda, 1^N, 1^{\ell_0}, 1^m) & \text{if } \mathcal{D} \neq [1, N] \\ \mathsf{FSS.Gen}(1^\lambda, g_{x_1,\ldots,x_N}) & \text{if } \mathcal{D} = [1, N] \end{cases} .$$
  If P_0 is corrupted, Sim instead sets
  $$(-, (k_i)_{i \in \mathcal{D} \setminus \{0\}}, (y_i)_{i \in [1,N] \setminus \mathcal{D}}, -) \xleftarrow{\$} \mathsf{Sim}_{\mathsf{rec,info}}^{\mathsf{FSS}}(1^\lambda, 1^N, 1^{\ell_0}, 1^m, x_0, y, (x_i)_{i \in \mathcal{D} \setminus \{0\}}) .$$
  In either case, Sim then writes $(k_i)_{i \in \mathcal{D} \setminus \{0\}}$ on $\mathcal{A}$'s input tape (as if $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}(C)$ sent k_i to P_i for each $i \in \mathcal{D} \setminus \{0\}$).
  3. If P_0 is corrupted:

     Sim waits to receive the messages $(\mathsf{sid}_i, \mathtt{Client}, x_0)_{i \in \mathcal{D} \setminus \{0\}}$ that $\mathcal{A}$ would send to $\mathcal{F}_{\mathsf{OE}}^{\mathsf{FSS}}(C)$ (on behalf of the P_0).
  4. Sim sends $(\mathtt{input}, i, x_i)_{i \in \mathcal{D}}$ to $\mathcal{F}_{\mathsf{SFE}}(C)$ and waits to receive y (on behalf of the corrupted parties).
  5. If P_0 is corrupted:

For $i \notin \mathcal{D}$, Sim writes (sid_i, y_i) (recall that y_i was defined in step 2. above) on $\mathcal{A}$'s input tape (as if P_i sent it to P_0).

6. If P_0 is not corrupted then, for $i \in \mathcal{D}$, Sim writes y on $\mathcal{A}$'s input tape (as if P_0 sent it to P_i).

Figure 7.6: Simulator Sim for an execution of protocol Π_C.

In order to prove that no environment $\mathcal{Z}$ can distinguish between the real and ideal worlds, we proceed by a case analysis over the set of corrupted parties $\mathcal{D}$.

- **If only P_0 is honest (i.e. $\mathcal{D} = [1, N]$):** In this case the simulation is perfect. Indeed, the simulator knows the corrupt parties' inputs $x_1, \ldots, x_N$, which allows it to perfectly emulate the ideal functionality $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}(C)$. The only other incoming messages received by the corrupted parties is the broadcast output, which is identical both in the real and ideal worlds by (perfect) correctness of the FSS scheme.

- **If P_0 and at least one other P_i are honest (i.e. $\mathcal{D} \subsetneq [1, N]$):** The only difference between the real and the ideal worlds is whether the corrupted parties receive real FSS keys or simulated ones. Indeed, by (perfect) correctness of the FSS the broadcast message received is identical in the real and the ideal world. By the first part of lemma 13 (function secrecy given some keys, and auxiliary information), the joint views of $\mathcal{Z}$ and $\mathcal{A}$ are indistinguishable in the real and the ideal worlds: even given the auxiliary information $(y, (x_i)_{i \in \mathcal{D}})$ about the function, it cannot distinguish the real corrupted keys $(k_i)_{i \in \mathcal{D}}$ from $(k_i)_{i \in \mathcal{D}} \xleftarrow{\$} \mathsf{Sim}^{\mathsf{FSS}}(1^\lambda, 1^N, 1^{\ell_0}, 1^m)$.

- **If P_0 is corrupt (i.e. $\mathcal{D} \not\subseteq [1, N]$):** Note that since the adversary is only allowed to corrupt up to all but one parties, if P_0 is corrupt then at least one P_i ($i \in [1, N]$) is honest. The only difference between the real and the ideal world is whether the corrupted keys $(k_i)_{i \in \mathcal{D} \setminus \{0\}}$ and the honest shares $(y_i)_{i \in [1,N] \setminus \mathcal{D}}$ are real or simulated by $\mathsf{Sim}_{\mathsf{rec,info}}^{\mathsf{FSS}}$. It follows from the second part of lemma 13 (reconstruction-only given some keys, all shares, and auxiliary information) that the joint views of $\mathcal{Z}$ and $\mathcal{A}$ are indistinguishable in the real and the ideal worlds.

□

7.3 Oblivious Evaluation of LogLog-Depth FSS from PIR

In the previous section we provided a generic template for $(N + 1)$-party sublinear secure computation from N-party additive function secret sharing for which $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}$ and $\mathcal{F}_{\mathsf{OE}}^{\mathsf{FSS}}$ can be securely realised with low communication. In this section we introduce the notion of *loglog-depth* for (additive) FSS schemes, and show that this property allows $\mathcal{F}_{\mathsf{OE}}^{\mathsf{FSS}}$ to be securely realised with low communication using *correlated symmetric PIR (*$\mathsf{corrSPIR}$*)*, a primitive introduced in [BCM22] (and which can be instantiated from standard assumptions using the rate-1 batch OT from [BBDP22]).

7.3.1 LogLog-Depth FSS

A depth-d, n-input, m-output arithmetic circuit with gates of fan-in at most two over a finite field $\mathbb{F}$ can be associated with the degree-($\leq 2^d$) n-variate m-output[5] polynomial with coefficients in $\mathbb{F}$ that it computes. In all generality, a degree-2^d n-variate polynomial can have up to $\mathsf{nb}_{n,2^d} = \sum_{k=0}^{2^d} \binom{k+n-1}{n-1}$ different monomials (which can be verified using a stars-and-bars counting argument). In this section we will only be interested in circuits whose representation as a polynomial is the sum of $\mathsf{poly}(\lambda)$ monomials (where λ is the security parameter). A sufficient condition is for it to have $n = \mathsf{poly}(\lambda)$ inputs and depth $d \leq \log\log(n)$; we refer to this property as a circuit being "of loglog-depth". Indeed, because we only consider circuits whose gates have fan-in at most two, if a circuit has depth d then it is 2^d-local (i.e. each of its m outputs is a function of only at most 2^d inputs). Therefore each of its outputs is computed by a polynomial with at most $\mathsf{nb}_{2^d,2^d} \leq 2^{2^d+2^d}$ monomials, which is $\mathsf{poly}(\lambda)$ if $d = \log\log n = \log\log\lambda + \mathcal{O}(1)$.

We extend in definition 27 the above notion of "loglog-depth" circuits to "loglog-depth" FSS schemes.

Definition 27 (LogLog-Depth FSS)**.** *Let $\mathcal{F}$ be a class of functions with n inputs and m outputs over a finite field $\mathbb{F}$. We say that an N-party FSS scheme $\mathsf{FSS} = (\mathsf{FSS.Gen}, \mathsf{FSS.Eval})$ for $\mathcal{F}$ whose evaluation algorithm $\mathsf{FSS.Eval}$ is explicitly described as an arithmetic circuit, has* loglog-depth *(alternatively, FSS is a loglog-depth function secret sharing scheme) if for every party index $i \in [N]$ and every key $k_i \in \mathsf{Supp}([\mathsf{FSS.Gen}]_i)$ the circuit $\mathsf{FSS.Eval}(i, k_i, \cdot)$ (which has hardcoded i and k_i) has depth* $\log\log(n)$.

Throughout this section we will be using "loglog-depth" circuits and FSS schemes, but it should be noted that all of our results go through if this is replaced everywhere with the more obtuse notion of "circuits (*resp.* FSS evaluation) whose polynomial representation has a polynomial number of coefficients".

When considering "loglog-depth", which in particular are "log-local" circuits, we will be interested in the log-sized subsets of the inputs on which each output depends. We say that an FSS scheme is $(S_1, \ldots, S_m)$-local if the j^{th} output of $\mathsf{FSS.Eval}$, which takes as input a party index i, a key k_i, and an input x, only depends on $(i, k_i, x[S_j])$. In other words, an FSS scheme is $(S_1, \ldots, S_m)$-local if its evaluation algorithm is $(S_1, \ldots, S_m)$-local in its last input. We emphasize that a loglog-depth circuit or FSS scheme is always log-local, but that the converse is not necessarily true if $\mathbb{F} \neq \mathbb{F}_2$.

7.3.2 Oblivious Evaluation of LogLog-Depth FSS from PIR

We first discuss the notion of PIR we need, then show how it can be leveraged to build oblivious evaluation of any loglog-depth FSS scheme.

7.3.2.1 Correlated PIR.

We recall the ideal functionality for *batch SPIR with correlated "mix and match" queries* ($\mathcal{F}_{\mathsf{corrSPIR}}$, fig. 7.7) from [BCM22], which we extend from the boolean to the arithmetic setting as *batch Oblivious (Multivariate) Polynomial Evaluation with*

[5] An m-output (multivariate) polynomial can be seen as a tuple of m (multivariate) polynomials.

correlated "mix and match" queries ($\mathcal{F}_{\mathsf{corrOPE}}$, fig. 7.8).

In the boolean world, this corresponds to a batched form of SPIR, querying into k size-N databases, where the queries are not independent. Rather, the queried indices can be reconstructed via a public function that "mixes and matches" individual bits of a single bitstring $\vec{\alpha} = (\alpha_1, \ldots, \alpha_w)$ of length $w < k \log N$, in a public manner. What this means is that each of the $(n = \log N)$-bit queries to a single database can be obtained by concatenating n of the bits α_i, possibly permuted. In the arithmetic world, this corresponds to batch multivariate OPE, where each database corresponds to a polynomial, and the evaluation inputs are various subvectors of some *joint input vector*, comprised of w field elements. More specifically, the input to a single d-variate polynomial (in the batch to be obliviously evaluated) is a size-d ordered subset of the joint inputs.
We will be interested in how many times a given bit of entropy (*resp.* input) α_i appears within the k queries (*resp.* input)–counted by the occurrence function t_i below–, as well as how many times it appears in specific index position $j' \in [n]$ within the k queries (*resp.* input)–denoted below by $t_{i,j'}$–. To the best of our knowledge, there are no protocols realising corrOPE over superpolynomial-size fields without FHE, and the only protocol realising corrSPIR without FHE requires introducing this notion of "balance between the queried bits".

Definition 28 ("Mix and Match" Functions, adapted from [BCM22])**.** *A "mix and match" function* $\mathsf{MixAndMatch}\colon \mathbb{F}^w \to (\mathbb{F}^{N_{\mathsf{var}}})^k$ *is one parameterised by* k *ordered subsets of* N_{var} *elements of* $[w]$, $S_j = (s_{j,1}, \ldots, s_{j,n}) \in [w]^{N_{\mathsf{var}}}$ *for* $j \in [k]$ *such that:*

$$\forall \vec{\alpha} = (\alpha_1, \ldots, \alpha_w) \in \mathbb{F}^w, \mathsf{MixAndMatch}(\alpha_1, \ldots, \alpha_w) \coloneqq (x_1, \ldots, x_k), \\ \text{with } x_j \coloneqq \alpha_{s_{j,1}} \cdots \alpha_{s_{j,n}} \in (\mathbb{F}^{N_{\mathsf{var}}}).$$

Such a function is associated with an occurrence function, which counts the occurrences of each input position in the outputs:

$$\begin{array}{rccl} t.\colon & [w] & \to & [k] \\ & i & \mapsto & t_i = \sum_{j=1}^{k} \mathbf{1}_{i \in S_j} \end{array}$$

Each t_i $(i \in [w])$ *can be decomposed as* $t_i = t_{i,1} + \cdots + t_{i,N_{\mathsf{var}}}$*, where* $t_{i,j'}$ *is equal to the number of values of* $j \in [k]$ *such that* $s_{j,j'} = i$*.*

- $\mathsf{MixAndMatch}$ *is said to be* T-balanced *if* $\forall i \in [w], \forall j' \in [N_{\mathsf{var}}], t_{i,j'} \leq T$*.*
- $\mathsf{MixAndMatch}$ *is said to be* T-balanceable *if it can be expressed as the function* $\mathsf{MixAndMatch} = (\mathsf{MixAndMatch}' \circ \mathsf{replicate})$*, where* $\mathsf{MixAndMatch}\colon \mathbb{F}^{w'} \to (\mathbb{F}^{N_{\mathsf{var}}})^k$ *is a* T*-balanced mix-and-match function and* replicate *is defined as:*

$$\begin{array}{rccl} \mathsf{replicate}\colon & \mathbb{F}^w & \to & \mathbb{F}^{w'} \\ & (b_1, \ldots, b_w) & \mapsto & (b_1^{\| \lceil t_1/T \rceil} \| \ldots \| b_w^{\| \lceil t_w/T \rceil}) \end{array} \quad \text{where } w' \coloneqq \sum_{i \in [w]} \lceil t_i/T \rceil.$$

The ideal functionality for batch SPIR with correlated "Mix and Match" queries from [BCM22] is recalled in fig. 7.7. It can be seen as the special case of batch OLE with correlated "Mix and Match" inputs over $\mathbb{F}_2$.

Functionality $\mathcal{F}_{\mathsf{corrSPIR}}$

The functionality $\mathcal{F}_{\mathsf{corrSPIR}}$ is parameterised by the number k of SPIRs in the batch, the size N of each database, and the number w of selection bits. Furthermore, it is parameterised by a public T-balanceable "mix and match" function (definition 28) $\mathsf{MixAndMatch}\colon \{0,1\}^w \to [N]^k$. $\mathcal{F}_{\mathsf{corrSPIR}}$ interacts with an ideal sender $\mathbf{S}$ and an ideal receiver $\mathbf{R}$ via the following queries.

1. On input $(\texttt{sender}, \vec{M} = (\vec{m}_i)_{i\in[k]})$ from $\mathbf{S}$ (where $\vec{m}_i = (m_{i,j})_{j\in[N]} \in \{0,1\}^N$), store $\vec{M}$.
2. On input $(\texttt{receiver}, (\alpha_j)_{j\in[w]})$ from $\mathbf{R}$ (where $(\alpha_j)_{j\in[w]} \in \{0,1\}^w$), check if a tuple of inputs $\vec{M}$ has already been recorded; if so, compute $(x_1, \ldots, x_k) \coloneqq \mathsf{MixAndMatch}(\alpha_1, \ldots, \alpha_w) \in [N]^k$, send $(m_{i,x_i})_{i\in[k]}$ to $\mathbf{R}$, and halt.

If the functionality receives an incorrectly formatted input, it aborts.

Figure 7.7: Ideal Functionality $\mathcal{F}_{\mathsf{corrSPIR}}$ for Batch SPIR with Correlated "Mix and Match" Queries

Functionality $\mathcal{F}_{\mathsf{corrOPE}}$

The functionality $\mathcal{F}_{\mathsf{corrOPE}}$ is parameterised by a finite field $\mathbb{F}$, the number k of oblivious (multivariate) polynomial evaluations (OPE) in the batch, the number N_{var} of variables of each polynomial, the degree d of each polynomial, and the size w of the joint inputs vector. Let $\mathsf{nb}_{N_{\mathsf{var}},d}\colon \sum_{d'=0}^{d} \binom{N_{\mathsf{var}}+d'}{N_{\mathsf{var}}}$ denote the maximum number of monomials of a degree-d N_{var}-variate polynomial. Furthermore, it is parameterised by a public polylog-balanced "mix and match" function (definition 28) $\mathsf{MixAndMatch} \coloneqq \mathbb{F}^w \to \mathbb{F}^k$. $\mathcal{F}_{\mathsf{corrOPE}}$ interacts with an ideal sender $\mathbf{S}$ and an ideal receiver $\mathbf{R}$ via the following queries.

1. On input $(\texttt{sender}, \vec{c}_i)$ from $\mathbf{S}$ (where $(\vec{c}_i)_{i\in[k]} \in \mathbb{F}^{k\cdot \mathsf{nb}_{N_{\mathsf{var}},d}}$), store $(\vec{c}_i)_{i\in[k]}$.
   // $\vec{c}_i$ is the vector of coefficients of the i^{th} polynomial (with zeroes).
2. On input $(\texttt{receiver}, (\alpha_j)_{j\in[w]})$ from $\mathbf{R}$ (where $(\alpha_j)_{j\in[w]} \in \mathbb{F}^w$), check if a tuple of inputs $(\vec{c}_i)_{i\in[k]}$ has already been recorded; if so, compute $(x_1, \ldots, x_k) \coloneqq \mathsf{MixAndMatch}(\alpha_1, \ldots, \alpha_w) \in (\mathbb{F}^{N_{\mathsf{var}}})^k$, compute $y_i \leftarrow \vec{c}_i \cdot (x_i^{\otimes d} \| \ldots \| x_i^{\otimes 1} \| 1) \in \mathbb{F}$ for $i \in [k]$, send $(y_i)_{i\in[k]}$ to $\mathbf{R}$, and halt.
   // $\otimes$ denotes the tensor product. Each x_i is the vector of the N_{var} inputs to the i^{th} polynomial in the batch, whose coefficients are given by $\vec{c}_i$. Therefore, y_i is the evaluation of the i^{th} polynomial on input x_i.

If the functionality receives an incorrectly formatted input, it aborts.

Figure 7.8: Ideal Functionality $\mathcal{F}_{\mathsf{corrOPE}}$ for Batch Oblivious Polynomial Evaluation with Correlated "Mix and Match" Inputs

7.3.2.2 Oblivious Evaluation of LogLog-Depth FSS from PIR.

Let $\mathsf{FSS} = (\mathsf{FSS.Gen}, \mathsf{FSS.Eval})$ be a loglog-depth, $(S_1, \ldots, S_m)$-local FSS scheme (definition 27). Because FSS has loglog-depth, the polynomial representation of $\mathsf{FSS.Eval}$ has $m \cdot \mathsf{poly}(n)$ coefficients. Furthermore, each of its local evaluation algorithms $\mathsf{FSS.Eval}_j$ depends only on the inputs indexed by S_j. Therefore obliviously evaluating $\mathsf{FSS.Eval}$ can be done by using batch OPE with correlated "mix and match" inputs: the m polynomials in the batch are the $\mathsf{FSS.Eval}_j(i, k_i, \cdot)$, where k_i is known only to the server P_i. This protocol is formalised in fig. 7.9.
Note that this notion of $\mathsf{corrOPE}$, as defined in fig. 7.8, requires the polynomials in the batch be represented as a vector of coefficients. For this reason we impose that FSS be loglog-depth, so this vector be polynomial-size.

Protocol Oblivious Evaluation of Partial Function Shares Π_{OE}

Parties: P_0 (the client) and P_i (the server).

Parameters:

- Let N be a number, and let $C = C_1 \| \ldots \| C_m$ be a loglog-depth circuit (definition 27) with $n = \ell_0 + \ell_1 + \cdots + \ell_N$ inputs and m outputs over $\mathbb{F}$ such that the following function family $\mathcal{C}$ is $(S_1, \ldots, S_m)$-local, where $S_1, \ldots, S_m$ is some family of $(\log / \log\log)$-sized subsets of $[n]$:

$$\mathcal{C} = \left\{ g_{\alpha_1,\ldots,\alpha_N} : \begin{array}{ccc} \mathbb{F}^{\ell_0} & \to & \mathbb{F}^m \\ x & \mapsto & C(x, \alpha_1, \ldots, \alpha_N) \end{array} : (\alpha_1, \ldots, \alpha_N) \in \mathbb{F}^{\ell_1} \times \cdots \times \mathbb{F}^{\ell_N} \right\} .$$

  We assume that each of the S_i is ordered in such a way that the function $\mathsf{MixAndMatch}$ associated with $(S_1, \ldots, S_m)$ is $\mathsf{polylog}$-balanced[a] (definition 28).

- $\mathsf{FSS} = (\mathsf{FSS.Gen}, \mathsf{FSS.Eval})$ is an $(S_1, \ldots, S_m)$-local (additive) FSS scheme for $\mathcal{C}$, whose local evaluation algorithms are $(\mathsf{FSS.Eval}_j)_{j \in [m]}$.

Hybrid Model: The protocol is defined in the $\mathcal{F}_{\mathsf{corrOPE}}$-hybrid model (the subsets characterising $\mathsf{MixAndMatch}$, and in turn $\mathsf{corrOPE}$, are $(S_1, \ldots, S_m)$).

Input: P_0 holds input $x_0 \in \{0,1\}^{\ell_0}$, and P_i holds k_i.

The Protocol:

- **First Round:**
  1. P_0 sends $(\texttt{receiver}, x_0)$ to $\mathcal{F}_{\mathsf{corrOPE}}$
  2. P_i sends $(\texttt{sender}, (\vec{c}_j)_{j \in [m]})$ to $\mathcal{F}_{\mathsf{corrOPE}}$ where $\vec{c}_j$ is the vector of coefficients of $\mathsf{FSS.Eval}_j(i, k_i, \cdot)$
     // For the case $\mathbb{F} = \mathbb{F}_2$ (i.e. when using $\mathcal{F}_{\mathsf{corrSPIR}}$), the databases can be more simply described as the truth tables of the $\mathsf{FSS.Eval}_j(i, k_i, \cdot)$ for $j \in [m]$, i.e. $(\mathsf{FSS.Eval}_j(i, k_i, x'))_{x' \in \{0,1\}^{|S_j|}}$.

- **Second Round:**
  3. P_0 waits to receive $(y_{i,1}, \ldots, y_{i,m})$ from $\mathcal{F}_{\mathsf{corrOPE}}$

4. P_0 outputs $(y_{i,1}, \ldots, y_{i,m})$

[a] By [?, Lemma 9], such orderings exist and furthermore can be found in expected constant time by random shuffling. Alternatively, since a random ordering of $(S_1, \ldots, S_m)$ works with high probability, the protocol could be modified so that P_0 samples a PRG key and sends it to P_1, and both use the resulting pseudorandomness to order $(S_1, \ldots, S_m)$. This additional step incurs only a small additive overhead in communication, and the resulting protocol is still sublinear.

Figure 7.9: Two-party protocol for obliviously evaluating shares of an additive loglog-depth FSS scheme.

Lemma 14 (Oblivious Share Evaluation for LogLog-Depth FSS Schemes)**.** *Let $N \geq 2$. Let $C\colon \mathbb{F}^n \to \mathbb{F}^m$ be a loglog-depth arithmetic circuit with $n = \ell_0 + \ell_1 + \cdots + \ell_N$ inputs over a finite field $\mathbb{F}$, and let $\mathcal{C}$ be the family of "partial evaluations of C":*

$$\left\{ g_{\alpha_1,\ldots,\alpha_N}\colon \begin{array}{rcl} \mathbb{F}^{\ell_0} & \to & \mathbb{F}^m \\ x & \mapsto & C(x, \alpha_1, \ldots, \alpha_N) \end{array} : (\alpha_1, \ldots, \alpha_N) \in \mathbb{F}^{\ell_1} \times \cdots \times \mathbb{F}^{\ell_N} \right\} .$$

If FSS is an additive loglog-depth, $(S_1, \ldots, S_m)$-local FSS scheme (definition 27) for $\mathcal{C}$ and $\mathsf{corrSPIR}$ is a two-round batch SPIR protocol (characterised by $(S_1, \ldots, S_m)$), then the protocol Π_{OE} provided in fig. 7.9 UC-securely implements the two-party functionality $\mathcal{F}_{\mathsf{OE}}^{\mathsf{FSS}}$ against a static, passive adversary in the $\mathcal{F}_{\mathsf{corrOPE}}$-hybrid model.

Proof. The protocol of fig. 7.9 essentially boils down to a single call with no interaction between the players. In the real execution, the server P_i receives no incoming communication (neither from the client P_0 nor from the ideal functionality $\mathcal{F}_{\mathsf{corrOPE}}$), therefore simulation against a corrupted server is trivial. The only message received by the client P_0 is from the ideal functionality $\mathcal{F}_{\mathsf{corrOPE}}$, but since this message is also the output of P_0, the joint view of an adversary corrupting P_0 and of the environment can be perfectly simulated by obtaining said message from the ideal functionality $\mathcal{F}_{\mathsf{OE}}^{\mathsf{FSS}}$. □

7.4 LogLog-Depth FSS from Compact and Additive HSS

In this section we show how to use compact and additive HSS to build a loglog-depth FSS scheme whose share distribution $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}$ can be realised in low communication. When combined with sections 7.2 and 7.3, this yields sublinear secure computation from compact and additive HSS. In section 7.4.3 of the supplementary material, we show how to extend this construction to use the weaker primitive of Las-Vegas HSS.

7.4.1 From Compact and Additive HSS

7.4.1.1 An Overview of the Construction

Let $C\colon \mathbb{F}^n \to \mathbb{F}^m$ be a log log-depth arithmetic circuit with $n = \ell_0 + \ell_1 + \cdots + \ell_N$ inputs over a finite field $\mathbb{F}$, and let $\mathcal{C}$ be the family of "partial evaluations of C":

$$\left\{ g_{\alpha_1,\ldots,\alpha_N}\colon \begin{array}{rcl} \mathbb{F}^{\ell_0} & \to & \mathbb{F}^m \\ x & \mapsto & C(x, \alpha_1, \ldots, \alpha_N) \end{array} : (\alpha_1, \ldots, \alpha_N) \in \mathbb{F}^{\ell_1} \times \cdots \times \mathbb{F}^{\ell_N} \right\} .$$

Our goal in this section is to provide a construction of a loglog-depth FSS scheme for $\mathcal{C}$ such that $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}$ can be realised with low communication, and we do so by using compact and additive single-function HSS (*c.f.* remark 1) for any function in a well-chosen function class (that of $\{\mathsf{coefs}_{c_1,\ldots,c_N} : (c_1,\ldots,c_N) \in \mathbb{F}^{\ell_1} \times \cdots \times \mathbb{F}^{\ell_N}\}$, as defined below).

We provide in fig. 7.10 a construction of loglog-depth additive FSS for $\mathcal{C}$ from single-function additive HSS for the following function coefs:

$$\begin{array}{rccc} \mathsf{coefs}\colon & \mathbb{F}^{\ell_1} \times \cdots \times \mathbb{F}^{\ell_N} & \to & \mathbb{F}^{\star} \\ & (\alpha_1,\ldots,\alpha_N) & \mapsto & (p_0,p_1,\ldots) \end{array}$$

where $(p_0,p_1,\ldots)$ are the coefficients of the polynomial representation of all the $C_j(X,\alpha_1,\ldots,\alpha_N)$, for $j \in [m]$ (which are polynomials in X, whose coefficients are themselves polynomials in $\alpha_1,\ldots,\alpha_N$). Because C has loglog-depth (definition 27), there are at most $m \cdot n \cdot (1+o(1))$ such coefficients. Furthermore, the key generation algorithm of the FSS scheme for $\mathcal{C}$ essentially boils down to a single call to the share generation algorithm of the HSS scheme for coefs. Therefore, we also need to provide an HSS scheme for coefs whose share generation can be distributed using low communication. We use a transformation akin to hybrid encryption in order to ensure this last property: we mask the inputs using pseudorandom generators, and reduce the problem of generating HSS shares of the inputs to that of distributing HSS shares of the keys, which can be done generically using oblivious transfer.

More precisely, for $i \in [N]$ let $G_i\colon \{0,1\}^{\lambda} \to \mathbb{F}^{\ell_i}$ be a PRG and consider the function family $\{\mathsf{coefs}_{c_1,\ldots,c_N} : (c_1,\ldots,c_N) \in \mathbb{F}^{\ell_1} \times \cdots \times \mathbb{F}^{\ell_N}\}$, where:

$$\begin{array}{rccc} \mathsf{coefs}_{c_1,\ldots,c_N}\colon & \mathbb{F}^{\lambda} \times \cdots \times \mathbb{F}^{\lambda} & \to & \mathbb{F}^{\star} \\ & (K_1,\ldots,K_N) & \mapsto & (p_0,p_1,\ldots) \end{array}$$

where $(p_0,p_1,\ldots)$ are the coefficients of the polynomial representation of all the $C_j(X, c_1 - G_1(K_1),\ldots,c_N - G_N(K_N))$, $j \in [m]$ (which are polynomials in X, whose coefficents are polynomials in the bits of $K_1,\ldots,K_N$).
Assuming the existence of compact and additive single-function HSS for any function in $\{\mathsf{coefs}_{c_1,\ldots,c_N}\}$, the construction of fig. 7.11 is an HSS scheme for coefs whose share generation can be distributed using low communication (with the protocol being provided in fig. 7.12).

While this assumption relating to the existence of HSS for $\{\mathsf{coefs}_{c_1,\ldots,c_N}\}$ may not seem standard, it is weaker than each of the following assumptions:

1. HSS for NC^1 and polynomial-stretch PRGs in NC^1;
2. Single-function HSS for any $\log\log$-depth circuit and constant-depth PRGs with some fixed polynomial-stretch.

7.4.2 Defining the LogLog-Depth FSS Scheme.

Observation 1 (Parsing Additive Shares)**.** *Let $\vec{x} \in \{0,1\}^n$ and let $I \subseteq [n]$. If $(\vec{x}^{(1)},\ldots,\vec{x}^{(m)})$ are additive shares of $\vec{x}$, then $([\vec{x}^{(1)}]_I,\ldots,[\vec{x}^{(m)}]_I)$ are additive shares of $[\vec{x}]_I$, where $[\cdot]_I$ denotes the subvector induced by the set of coordinates I.*

LogLog-Depth FSS Scheme from Additive HSS

Parameters: Let $N \geq 2$ be a number of parties, and let $C = C_1 \| \dots \| C_m$ be a loglog-depth circuit with $n = \ell_0 + \ell_1 + \dots + \ell_N$ inputs and m outputs over $\mathbb{F}$ such that the following function family is $(S_1, \dots, S_m)$-local, where $S_1, \dots, S_m$ is some family of $(\log n / \log\log n)$-sized subsets of $[n]$:

$$\mathcal{C} = \left\{ \begin{array}{rccl} g_{\alpha_1,\dots,\alpha_N}\colon & \mathbb{F}^{\ell_0} & \to & \mathbb{F}^m \\ & x & \mapsto & C(x, \alpha_1, \dots, \alpha_N) \end{array} : (\alpha_1, \dots, \alpha_N) \in \mathbb{F}^{\ell_1} \times \dots \times \mathbb{F}^{\ell_N} \right\} .$$

Let $\mathsf{HSS} = (\mathsf{HSS.Share}, \mathsf{HSS.Eval})$ be an N-party additive (single-function) HSS scheme for the function:

$$\mathsf{coefs}\colon \begin{array}{rcl} \mathbb{F}^{\ell_1} \times \dots \times \mathbb{F}^{\ell_N} & \to & \mathbb{F}^\star \\ (\alpha_1, \dots, \alpha_N) & \mapsto & (p_0, p_1, \dots) \end{array}$$

where $(p_0, p_1, \dots)$ are the coefficients of the polynomial representation of all the $C_j(X, \alpha_1, \dots, \alpha_N)$, $j \in [m]$ (which are polynomials in X, whose coefficients are themselves polynomials in $\alpha_1, \dots, \alpha_N$).

// Note that since C has loglog-depth and $\mathcal{C}$ is $(S_1, \dots, S_m)$-local, each of the m polynomials has degree $|S_j|$ and $|S_j|$ variables, and there are therefore at most $\sum_{j=1}^{m} \binom{|S_j|+|S_j|}{|S_j|} = m \cdot n \cdot (1 + o(1))$ coefficients, regardless of $(\alpha_1, \dots, \alpha_N)$.

$\mathsf{FSS.Gen}(1^\lambda, \tilde{g}_{\alpha_1,\dots,\alpha_N})$:

1. Parse $\tilde{g}_{\alpha_1,\dots,\alpha_N}$ to retrieve $(\alpha_1, \dots, \alpha_N)$
2. $(k_1, \dots, k_N) \xleftarrow{\$} \mathsf{HSS.Share}(1^\lambda, i, (\alpha_1, \dots, \alpha_N))$
3. Output $(k_1, \dots, k_N)$

$\mathsf{FSS.Eval}_j(i, k_i, x')$: // $x' \in \mathbb{F}^{|S_j|}$ should be seen as an S_j-subset of some larger $x \in \mathbb{F}^{\ell_0}$ (i.e. $x' = x[S_j]$), input of $\mathsf{FSS.Eval}$.

1. $(p_{0,i}, p_{1,i}, \dots) \xleftarrow{\$} \mathsf{HSS.Eval}(i, k_i)$
2. Parse $(p_{0,i}, p_{1,i}, \dots)$ to retrieve shares $(q_{0,i}, q_{1,i}, \dots)$ of the coefficients of $C_j(\cdot, \alpha_1, \dots, \alpha_N)$ (*c.f.* observation 1).
3. $y_{i,j} \leftarrow (x')^{\otimes |S_j|} \cdot (q_{0,i}, q_{1,i}, \dots)^\intercal$
4. Output $y_{i,j}$

$\mathsf{FSS.Eval}(i, k_i, x)$:

1. For $j \in [m]$, set $y_{i,j} \leftarrow \mathsf{FSS.Eval}_j(i, k_i, x[S_j])$
2. Output $(y_{i,j})_{j \in [m]}$

Figure 7.10: LogLog-Depth FSS Scheme from "Single-Function" Additive HSS for every LogLog-Depth Circuit.

Lemma 15 (LogLog-Depth FSS Scheme from "Single-Function" Additive HSS)**.** *Let $N \geq 2$ be a number of parties, and let $C = C_1 \| \dots \| C_m$ be a loglog-depth circuit with $n = \ell_0 + \ell_1 + \cdots + \ell_N$ inputs and m outputs over $\mathbb{F}$ such that the following function family is $(S_1, \dots, S_m)$-local, where $S_1, \dots, S_m$ is some family of $(\log n / \log\log n)$-sized subsets of $[n]$:*

$$\mathcal{C} = \left\{ \begin{array}{rccl} g_{\alpha_1,\dots,\alpha_N}\colon & \mathbb{F}^{\ell_0} & \to & \mathbb{F}^m \\ & x & \mapsto & C(x, \alpha_1, \dots, \alpha_N) \end{array} : (\alpha_1, \dots, \alpha_N) \in \mathbb{F}^{\ell_1} \times \cdots \times \mathbb{F}^{\ell_N} \right\}.$$

Let $\mathsf{HSS} = (\mathsf{HSS.Share}, \mathsf{HSS.Eval})$ be an N-party (single-function) additive HSS scheme for the function:

$$\begin{array}{rccl} \mathsf{coefs}\colon & \mathbb{F}^{\ell_1} \times \cdots \times \mathbb{F}^{\ell_N} & \to & \mathbb{F}^\star \\ & (\alpha_1, \dots, \alpha_N) & \mapsto & (p_0, p_1, \dots) \end{array}$$

where $(p_0, p_1, \dots)$ are the coefficients of the polynomial representation of all the $C_j(\cdot, \alpha_1, \dots, \alpha_N)$, $j \in [m]$.

Then the construction of fig. 7.10 is an N-party additive loglog-depth and $(S_1, \dots, S_m)$-local FSS scheme for $\mathcal{C}$.

Proof. Consider the construction $\mathsf{FSS} = (\mathsf{FSS.Gen}, \mathsf{FSS.Eval})$ provided in fig. 7.10. The fact that $\mathsf{FSS.Eval}$ has loglog-depth and is $(S_1, \dots, S_m)$-local follows immediately from inspection, and so does FSS correctness (which follows from correctness of HSS). Therefore we only need to show that FSS is a secure function secret sharing scheme. Let $\mathcal{D} \subseteq [N]$. By security of HSS, for any PPT adversaries $\mathcal{A}$ and $\mathcal{A}'$,

$$\left| \Pr\left[b' = b : \begin{array}{l} (x_0, x_1, \mathsf{state}) \leftarrow \mathcal{A}(1^\lambda), \\ b \xleftarrow{\$} \{0,1\} \\ (x^{(1)}, \dots, x^{(N)}) \leftarrow \mathsf{HSS.Share}(x_b) \\ b' \leftarrow \mathcal{A}'\left(\mathsf{state}, (x^{(i)})_{i \in \mathcal{D}}\right) \end{array} \right] - \frac{1}{2} \right| \leq \mathsf{negl}(\lambda).$$

In particular, for every $(\alpha_i)_{i\in[N]} \in \mathbb{F}^{\ell_1} \times \cdots \times \mathbb{F}^{\ell_N}$, $\{(x^{(i)})_{i\in\mathcal{D}} \colon (x^{(i)})_{i\in[N]} \xleftarrow{\$} \mathsf{HSS.Share}(\alpha_1 \| \dots \| \alpha_N)\} \overset{c}{\approx} \{(x^{(i)})_{i\in\mathcal{D}} \colon (x^{(i)})_{i\in[N]} \xleftarrow{\$} \mathsf{HSS.Share}(0^{\mathbb{F}^{\ell_1}} \| \dots \| 0^{\mathbb{F}^{\ell_N}})\}$. If we define the algorithm $\mathsf{Sim}^{\mathsf{HSS}}$ as "$\mathsf{Sim}^{\mathsf{HSS}}(1^\lambda, 1^N, 1^n, 1^m) := ((x^{(i)})_{i\in[N]} \xleftarrow{\$} \mathsf{HSS.Share}(0^{\mathbb{F}^{\ell_1}} \| \dots \| 0^{\mathbb{F}^{\ell_N}})$; Output $(x^{(i)})_{i\in\mathcal{D}}$)" then, because the $\mathsf{FSS.Gen}$ essentially boils down to a single invocation of $\mathsf{HSS.Share}$, for every sequence of functions $(\tilde{g}_{\alpha_{1,\lambda},\dots,\alpha_{N,\lambda}})_{\lambda\in\mathbb{N}^\star}$, $\{(k_i)_{i\in\mathcal{D}} \colon (k_1, \dots, k_N) \xleftarrow{\$} \mathsf{FSS.Gen}(1^\lambda, \tilde{g}_{\alpha_{1,\lambda},\dots,\alpha_{N,\lambda}})\} \overset{c}{\approx} \{\mathsf{Sim}^{\mathsf{HSS}}(1^\lambda, 1^N, 1^n, 1^m)\}$, and therefore FSS is a secure function secret sharing scheme. □

7.4.2.1 Securely Realising $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}$ in Low Communication.

The FSS scheme $\mathsf{FSS} = (\mathsf{FSS.Gen}, \mathsf{FSS.Eval})$ of fig. 7.10 is parameterised by an additive single-function HSS scheme for the function coefs. We provide in fig. 7.11 the full version of the paper such an HSS scheme with the additional property that it yields FSS for which $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}$ can be securely realised in low communication (the protocol is described in fig. 7.13).

HSS Single-Function HSS for coefs

Parameters:

- For $i \in [N]$, $G_i \colon \{0,1\}^\lambda \to \mathbb{F}^{\ell_i}$ is a PRG.
- HSS' = $(\mathsf{HSS}'.\mathsf{Share}, \mathsf{HSS}'.\mathsf{Eval})$ is an N-party compact and additive single-function (remark 1) HSS scheme for any function in $\{\mathsf{coefs}_{c_1,\ldots,c_N} \colon (c_1,\ldots,c_N) \in \mathbb{F}^{\ell_1} \times \cdots \times \mathbb{F}^{\ell_N}\}$, where:

$$\begin{array}{rccc} \mathsf{coefs}_{c_1,\ldots,c_N} \colon & \mathbb{F}^\lambda \times \cdots \times \mathbb{F}^\lambda & \to & \mathbb{F}^\star \\ & (K_1,\ldots,K_N) & \mapsto & (p_0,p_1,\ldots) \end{array}$$

  where $(p_0,p_1,\ldots)$ are the coefficients of the polynomial representation of all the $C_j(X, c_1 - G_1(K_1),\ldots,c_N - G_N(K_N))$, $j \in [m]$ (which are polynomials in X, whose coefficents are polynomials in the bits of $K_1,\ldots,K_N$).

$\mathsf{HSS}.\mathsf{Share}(1^\lambda,(\alpha_1,\ldots,\alpha_N))$:

1. For $i = 1,\ldots,N$:
   (a) Sample a PRG seed $K_i \xleftarrow{\$} \{0,1\}^\lambda$ (for $G_i \colon \{0,1\}^\lambda \to \mathbb{F}^{\ell_i}$)
   (b) Set $c_i \leftarrow \alpha_i + G_i(K_i)$
2. $(k_1^{\mathsf{HSS}'},\ldots,k_N^{\mathsf{HSS}'}) \xleftarrow{\$} \mathsf{HSS}'.\mathsf{Share}(1^\lambda,(K_1\|\ldots\|K_N))$
3. For $i \in [N]$, set $k_i \leftarrow (K_i, k_i^{\mathsf{HSS}'}, (c_1,\ldots,c_N))$
4. Output $(k_1,\ldots,k_N)$

$\mathsf{HSS}.\mathsf{Eval}(i,k_i,\mathsf{coefs})$:

1. Parse k_i as $k_i = (K_i, k_i^{\mathsf{HSS}'}, (c_1,\ldots,c_N))$
2. $(p_{0,i},p_{1,i},\ldots) \xleftarrow{\$} \mathsf{HSS}.\mathsf{Eval}(i,k_i^{\mathsf{HSS}},\mathsf{coefs}_{c_1,\ldots,c_N})$
3. Output $(p_{0,i},p_{1,i},\ldots)$

Figure 7.11: HSS Scheme for coefs.

Lemma 16. *With the notations of fig. 7.11, if $G_1,\ldots,G_N$ are PRGs and HSS' is a compact and additive single-function HSS scheme for any function in $\{\mathsf{coefs}_{c_1,\ldots,c_N} \colon (c_1,\ldots,c_N) \in \mathbb{F}^{\ell_1} \times \cdots \times \mathbb{F}^{\ell_N}\}$, then HSS is an additive single-function HSS for $\{\mathsf{coefs}\}$.*

Proof. Let $\mathcal{D} \subsetneq [N]$, and let $\mathcal{A},\mathcal{A}'$ be *p.p.t.* adversaries. For all $(\alpha_1,\ldots,\alpha_N) \in \mathbb{F}^{\ell_1} \times \cdots \times \mathbb{F}^{\ell_N}$ consider the random variables $(x_0 = (K_{1,0}\|\ldots\|K_{N,0}), x_1 = (K_{1,1}\|\ldots\|K_{N,1}), \mathsf{state}) \xleftarrow{\$} \mathcal{A}(1^\lambda)$, $b \xleftarrow{\$} \{0,1\}$, $(c_1,\ldots,c_N) := (\alpha_1 + G(K_1),\ldots,\alpha_N + G(K_N))$, $(k_{1,b}^{\mathsf{HSS}'},\ldots,k_{N,b}^{\mathsf{HSS}'}) \xleftarrow{\$} \mathsf{HSS}'.\mathsf{Share}(x_b)$, $(x_b^{(i)})_{i\in[N]} := (K_{i,b}, k_{i,b}^{\mathsf{HSS}'}, (c_1,\ldots,c_N))_{i\in[N]}$, and $b' \xleftarrow{\$} \mathcal{A}'(\mathsf{state},(x_b^{(i)})_{i\in\mathcal{D}})$. Observe that security of HSS is equivalent to $|\Pr[b = b'] - 1/2| \leq \mathsf{negl}(\lambda)$. By security of the PRGs

$(G_i)_{i\in[N]\setminus\mathcal{D}}$, $\Pr[b = b']$ is only changed by at most an negligible additive factor if the law of $(c_i)_{i\in\mathcal{D}}$ is changed to uniformly random. Thence, the problem of showing $|\Pr[b = b'] - 1/2| \leq \mathsf{negl}(\lambda)$ in this modified experiment for HSS reduces to the security of HSS' via a standard hybrid argument. □

We now complete the construction by providing the low-communication protocol to distribute the keys of the FSS scheme of fig. 7.10. The protocol itself is relatively straightforward. The parties each sample a PRG seed and use it to mask their inputs, then broadcast these masked values (using communication $\mathcal{O}(N \cdot n)$). The parties then use generic MPC to distribute HSS shares of the concatenation of the N PRG seeds (the ideal functionality is provided in fig. 7.12). Because the HSS scheme is compact and the input size is $N \cdot \lambda$, this step uses only $(N \cdot \lambda)^{\mathcal{O}(1)}$ bits of communication.

Functionality Distributed-Input HSS Share Distribution $\mathcal{F}_{\mathsf{SD}}^{\mathsf{HSS}}$

Parameters: The ideal functionality $\mathcal{F}_{\mathsf{SD}}^{\mathsf{HSS}}$ is parameterised by a number of parties N and an N-party HSS scheme $\mathsf{HSS} = (\mathsf{HSS.Share}, \mathsf{HSS.Eval})$.

$\mathcal{F}_{\mathsf{SD}}^{\mathsf{HSS}}$ interacts with the N parties $P_1, \ldots, P_N$ in the following manner.

Input: Wait to receive $(\texttt{input}, i, x_i)$ from each party P_i.

Output: Run $(x^{(1)}, \ldots, x^{(N)}) \xleftarrow{\$} \mathsf{HSS.Share}(1^\lambda, (x_1 \| \ldots \| x_N))$; Output $x^{(i)}$ to each party P_i $(1 \leq i \leq N)$.

Figure 7.12: Ideal functionality $\mathcal{F}_{\mathsf{SD}}^{\mathsf{HSS}}$ for the generation of HSS shares of the concatenation of the parties inputs.

Protocol FSS Share Distribution Π_{SD}

Parties: $P_1, \ldots, P_N$.

Parameters:

- $\mathsf{FSS} = (\mathsf{FSS.Gen}, \mathsf{FSS.Eval})$ is the N-party loglog-depth FSS scheme defined in fig. 7.10; $(\mathsf{FSS.Eval}_j)_{j\in[m]}$ are defined as in fig. 7.10. Implicitly, Π_{SD} inherits the parameters $C\colon \mathbb{F}^{\ell_0} \times \mathbb{F}^{\ell_1} \times \cdots \times \mathbb{F}^{\ell_N} \to \mathbb{F}^m, \mathcal{C}, (S_j)_{j\in[m]}, \mathsf{HSS}, \mathsf{coefs}$ of FSS.
- For $i \in [N]$, $G_i\colon \{0,1\}^\lambda \to \mathbb{F}^{\ell_i}$ is a constant-depth PRG.

Hybrid Model: The protocol is defined in the $\mathcal{F}_{\mathsf{SD}}^{\mathsf{HSS}}$-hybrid model.

Input: Each party P_i $(i \in [N])$ holds input $x_i \in \mathbb{F}^{\ell_i}$.

The Protocol: Each party P_i (for $i \in [N]$) does the following:

1. Sample $K_i \xleftarrow{\$} \{0,1\}^\lambda$ and set $c_i \leftarrow x_i + G_i(K_i)$
2. For $j \in [N] \setminus \{i\}$ (in parallel), send c_i to P_j and wait to receive c_j from P_j
3. Send $(\texttt{input}, i, K_i)$ to $\mathcal{F}_{\mathsf{SD}}^{\mathsf{HSS}}$ and wait to receive k_i^{HSS} from $\mathcal{F}_{\mathsf{HSS\text{-}SD}}$

4. Set $k_i^{\mathsf{FSS}} \leftarrow (K_i, k_i^{\mathsf{HSS}}, (c_1, \ldots, c_N))$

5. Output k_i^{FSS}

Figure 7.13: N-party protocol for the share distribution of the loglog-depth FSS scheme from additive HSS of fig. 7.10.

Lemma 17 ($\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}$ for the LogLog-Depth FSS scheme of fig. 7.10 can be realised with low communication)**.** *Let $N \geq 2$ be a number of parties, and let $C = C_1 \| \ldots \| C_m$ be a loglog-depth circuit with $n = \ell_0 + \ell_1 + \cdots + \ell_N$ inputs and m outputs over $\mathbb{F}$ such that the following function family is $(S_1, \ldots, S_m)$-local, where $S_1, \ldots, S_m$ is some family of $(\log / \log\log)$-sized subsets of $[n]$:*

$$\mathcal{C} = \left\{ \begin{array}{rccl} g_{\alpha_1,\ldots,\alpha_N} \colon & \mathbb{F}^{\ell_0} & \to & \mathbb{F}^m \\ & x & \mapsto & C(x, \alpha_1, \ldots, \alpha_N) \end{array} : (\alpha_1, \ldots, \alpha_N) \in \mathbb{F}^{\ell_1} \times \cdots \times \mathbb{F}^{\ell_N} \right\} .$$

For $i \in [N]$, let $G_i \colon \{0,1\}^\lambda \to \mathbb{F}^{\ell_i}$ be a constant-depth PRG*.*

Let $\mathsf{HSS} = (\mathsf{HSS.Share}, \mathsf{HSS.Eval})$ *be an N-party compact and additive HSS scheme for any function in* $\{\mathsf{coefs}_{c_1,\ldots,c_N} \colon (c_1, \ldots, c_N) \in \mathbb{F}^{\ell_1} \times \cdots \times \mathbb{F}^{\ell_N}\}$*, where:*

$$\begin{array}{rccl} \mathsf{coefs}_{c_1,\ldots,c_N} \colon & \mathbb{F}^\lambda \times \cdots \times \mathbb{F}^\lambda & \to & \mathbb{F}^\star \\ & (K_1, \ldots, K_N) & \mapsto & (p_0, p_1, \ldots) \end{array}$$

where $(p_0, p_1, \ldots)$ are the coefficients of the polynomial representation of all the $C_j(X, c_1 - G_1(K_1), \ldots, c_N - G_N(K_N))$, $j \in [m]$ (which are polynomials in X).

Then the protocol Π_{SD} provided in fig. 7.13 UC-securely implements the N-party functionality $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}$ in the $\mathcal{F}_{\mathsf{SD}}^{\mathsf{HSS}}$-hybrid model against a static, passive adversary. Furthermore, assuming oblivious transfer, there exists a protocol (in the real world) using $(N \cdot \lambda)^{\mathcal{O}(1)} + N(N-1) \cdot n \cdot \log |\mathbb{F}|$ bits of communication which UC-securely implements the N-party functionality $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}$ against a static, passive adversary.

Due to space considerations, we refer to [BCM23] for the proof of lemma 17.

7.4.3 From Compact and Additive HSS with Errors

In this section, we give a new construction of loglog-depth additive FSS scheme from additive HSS. In contrast to the construction of the previous section, this construction starts from a *Las Vegas* HSS scheme, i.e. , an HSS scheme where each output has a non-negligible correctness error, which can be reduced to an arbitrarily small inverse polynomial function (see Definition 2). A core ingredient of our construction, beyond a Las Vegas HSS scheme, is a *distributed point function* (DPF, [GI14]): a 2-party FSS scheme for the class of point functions $f_{\alpha,\beta} : \{0,1\}^\ell \to \mathbb{G}$ such that $f_{\alpha,\beta}(\alpha) = \beta$, and $f_{\alpha,\beta}(x) = 0$ otherwise. A sequence of works [GI14, BGI15, BGI16b] has led to highly efficient constructions of DPF schemes from any pseudorandom generator (PRG). (In fact, as one of the two parties will know the identity of the secret nonzero evaluation input, the weaker tool of a punctured pseudorandom function will suffice for the construction as given.)

Theorem 15 (PRG-based DPF [BGI16b]). *Given a length-doubling PRG* PRG*, and assuming* $|\mathbb{F}| \leq 2^{O(\ell)}$*, there exists a DPF for point functions* $f_{\alpha,\beta} : \{0,1\}^\ell \to \mathbb{F}$ *with key size* $O(\ell \cdot \lambda)$ *bits. The key generation algorithm* Gen *and the evaluation algorithm* Eval *invoke* PRG *at most* $O(\ell)$ *times.*

7.4.3.1 Additive FSS Scheme from Las Vegas Additive HSS.

In this section, we focus our attention to the $(N+1)$-party setting with $N = 2$, in line with the fact that all known instantiations of Las Vegas HSS are for 2 parties. As in Section 7.4.2, we consider a circuit C with m outputs (denoting C_i the circuit computing the i-th) output and $n = \ell_0 + \ell_1 + \ell_2$ inputs, such that the functions $x \to C(\alpha_0, \alpha_1, x)$ are all $(\log / \log\log)$-local and $(\log / \log\log)$-degree (for any possible choice of (α_0, α_1); note that this holds in particular when C is itself a $(\log\log - \log\log\log)$-depth circuit). For $j = 1$ to m, we let S_j denote the $(\log n / \log\log n)$-sized subset of entries of x that influences $C(\alpha_0, \alpha_1, x)$.
Let $\mathsf{HSS} = (\mathsf{HSS.Share}, \mathsf{HSS.Eval})$ be a 2-party Las Vegas additive HSS scheme for a class $\mathcal{F}$ such that $\mathsf{coefs} \in \mathcal{F}$, where $\mathsf{coefs}(\alpha_0, \alpha_1)$ computes the coefficients of the representation of $C_j(\alpha_0, \alpha_1, \cdot)$ as multivariate polynomials for $j = 1$ to m (there are at most $\binom{2n}{n} = n(1 + o(1))$ such coefficients for each $C_j(\alpha_0, \alpha_1, \cdot)$).
Note that the definition of Las Vegas HSS (Definition 2) guarantees that the evaluation is (verifiably) correct with probability $1 - \delta$, for some inverse polynomial bound δ. Without loss of generality, when we homomorphically evaluate functions with multiple outputs, we can actually assume that *each output* fails with independent probability δ: it suffices for this to evaluate individually the function restricted to each of its output, and to use a nonce in Eval to guarantee (pseudo)-independent failure probabilities (see [BGI16a] for discussions about this). Furthermore, denoting B a bound on the total number of outputs of the target function, setting the individual failure bounds δ of each output to $1/B$ guarantees an expected constant number of failures overall. Then, by a straightforward Chernoff bound, one can assume that the total number of $\perp$ flags obtained by any party is at most λ, except with probability $\mathsf{negl}(\lambda)$. Therefore, to simplify the description, we slightly change the template definition of $\mathsf{HSS.Eval}$:

- We assume that $\mathsf{HSS.Eval}$ returns a list T of outputs, together with a list flags of all the positions of the lists for which a $\perp$ flag was raised;
- we write $\mathsf{HSS.Eval}(\mathsf{coefs}, s, \lambda)$ to indicate that coefs is homomorphically evaluated on a share s such that the *total number of* $\perp$ *flags* output by $\mathsf{HSS.Eval}$ (i.e. the size of flags) is bounded by λ with overwhelming probability.

LogLog-Depth FSS Scheme from Las Vegas HSS

Parameters: A 2-party Las Vegas additive HSS scheme $\mathsf{HSS} = (\mathsf{HSS.Share}, \mathsf{HSS.Eval})$ for a class $\mathcal{F}$ such that $\mathsf{coefs} \in \mathcal{F}$, where $\mathsf{coefs}(\alpha_0, \alpha_1)$ computes the (polynomially many) coefficients of the representation of $C_j(\cdot, \alpha_0, \alpha_1)$ as multivariate polynomials for $j = 1$ to m. Let B be the number of outputs of coefs. In the description below, we consider two virtual parties, P_0 and P_1. We use the expression "sample X for P_i" to indicate that X is sampled using the random tape of P_i.

$\mathsf{FSS.Gen}(1^\lambda, C(\alpha_0, \alpha_1, \cdot))$:

1. Let $(s_0, s_1) \leftarrow \mathsf{HSS.Share}(\alpha_0, \alpha_1)$. For $i \in \{0,1\}$, sample $\mathsf{seed}_i \xleftarrow{\$} \{0,1\}^\lambda$.

Define $R_i \leftarrow \mathsf{PRG}(\mathsf{seed}_i)$ to be the random tape of the (virtual) party P_i.

2. For $i \in \{0,1\}$, compute $(T_i, \mathsf{flags}_i) \xleftarrow{\$} \mathsf{HSS.Eval}(\mathsf{coefs}, s_i, \lambda)$ for P_i. We assume that $|\mathsf{flags}_i| = \lambda$ (if $|\mathsf{flags}_i| > \lambda$, we restart from scratch; else, we pad flags_i with arbitrary indices until $|\mathsf{flags}_i| = \lambda$).
3. $\forall i \in \{0,1\}$, let $\mathsf{flags}_i = \{j_{i,1}, \cdots, j_{i,\lambda}\}$. For $k = 1$ to λ, set $T_i[j_{i,k}]$ to 0.
4. $\forall i \in \{0,1\}$, for $k = 1$ to λ, define $f_{i,k}$ to be the point function which evaluates to $\mathsf{coefs}_{j_{i,k}}(\alpha_0, \alpha_1) - T_{1-i}[j_{i,k}]$ on $j_{i,k}$ (and to 0 everywhere else), and run $(t^i_{0,k}, t^i_{1,k}) \xleftarrow{\$} \mathsf{DPF.Gen}(f_{i,k})$ using fresh random coins.
5. Set $\mathsf{k}_i \leftarrow (s_i, \mathsf{seed}_i, \mathsf{flags}_i, (t^0_{i,k}, t^1_{i,k})_{k \le \lambda})$ for $i = 0, 1$, and output $(\mathsf{k}_0, \mathsf{k}_1)$.

$\mathsf{FSS.Eval}_j(i, \mathsf{k}_i, x')$: we view x' as $x[S_j]$, i.e. , the size-$(\log n / \log\log n)$ subset of entries of some vector x, indexed by S_j. Parse k_i as $(s_i, \mathsf{seed}_i, \mathsf{flags}_i, (t^i_{i,k}, t^i_{1-i,k})_{k \le \lambda})$. In the following, use $R_i = \mathsf{PRG}(\mathsf{seed}_i)$ as random tape, matching the coins used in $\mathsf{FSS.Gen}$.

1. Compute $(T_i, \mathsf{flags}_i) \xleftarrow{\$} \mathsf{HSS.Eval}(\mathsf{coefs}, s_i, \lambda)$. For each $j \in \mathsf{flags}_i$, set $T_i[j]$ to 0.
2. For $k = 1$ to λ, for $j = 1$ to B, compute $c^j_{i,k} \leftarrow \mathsf{DPF.Gen}(t^0_{i,k}, j) + \mathsf{DPF.Gen}(t^1_{i,k}, j)$. Set $T_i[j] \leftarrow T_i[j] + \sum_{k=1}^{\lambda} c^j_{i,k}$.
3. Parse T_i to retrieve shares $(q_{0,i}, q_{1,i}, \dots)$ of the coefficients of $C_j(\alpha_0, \alpha_1, \cdot)$ (such parsing is possible because the shares are additive, *c.f.* observation 1).
4. Set $y_{i,j} \leftarrow (x')^{\otimes |S_j|} \cdot (q_{0,i}, q_{1,i}, \dots)^\intercal$.
5. Output $y_{i,j}$

$\mathsf{FSS.Eval}(i, k_i, x)$:

1. For $j \in [m]$, set $y_{i,j} \xleftarrow{\$} \mathsf{FSS.Eval}_j(i, k_i, x[S_j])$
2. Output $(y_{i,j})_{j \in [m]}$

Figure 7.14: LogLog-Depth FSS Scheme from Las Vegas Additive HSS for every LogLog-Depth Circuit.

Lemma 18 (LogLog-Depth FSS Scheme from Las Vegas HSS)**.** *Let $C = C_1 \| \dots \| C_m$ be a circuit with $n = \ell_0 + \ell_1 + \ell_2$ satisfying the loglog-depth constraint described above. Let $\mathsf{HSS} = (\mathsf{HSS.Share}, \mathsf{HSS.Eval})$ be a 2-party Las Vegas additive HSS scheme for a function class containing the coefs function. Then the construction of fig. 7.14 is a 2-party additive loglog-depth FSS scheme for $\mathcal{C}$.*

Proof. Consider the construction $\mathsf{FSS} = (\mathsf{FSS.Gen}, \mathsf{FSS.Eval})$ provided in fig. 7.14. loglog-depth is identical to that of the previous construction (from Figure fig. 7.10).

Correctness follows from the correctness of the HSS scheme, and from that of the DPF scheme. Concretely:

1. With probability 1, after computing $(T_0, \mathsf{flags}_0) \xleftarrow{\$} \mathsf{HSS.Eval}(\mathsf{coefs}, s_0, \lambda)$ and $(T_1, \mathsf{flags}_1) \xleftarrow{\$} \mathsf{HSS.Eval}(\mathsf{coefs}, s_1, \lambda)$, for every $j \notin \mathsf{flags}_0 \cup \mathsf{flags}_1$, $T_0[j] + T_1[j] = \mathsf{coefs}_j(\alpha_0, \alpha_1)$.

2. After updating $T_i[j]$ to $T_i[j] + \sum_{k=1}^{\lambda} c_{i,k}$ (in step 2), we have

$$T_0[j] + T_1[j] = \mathsf{coefs}_j(\alpha_0, \alpha_1) + \sum_{k=1}^{\lambda} (c^j_{0,k} + c^j_{1,k}),$$

   and it holds that $c^j_{0,k} + c^j_{1,k} = f_{0,k}(j) + f_{1,k}(j) = 0$, by correctness of the DPF scheme, and since $f_{0,k}$ and $f_{1,k}$ are point functions which (in particular) evaluate to 0 on all $j \notin \mathsf{flags}_0 \cup \mathsf{flags}_1$.

3. For every $j \in \mathsf{flags}_0 \setminus \mathsf{flags}_1$, we have $T_0[j] = 0$ after step 1, and $T_1[j]$ equal to some value v. Then after step 2, we have

$$T_0[j] + T_1[j] = v + \sum_{k=1}^{\lambda} (c^j_{0,k} + c^j_{1,k}),$$

   where $c^j_{0,k} + c^j_{1,k} = f_{0,k}(j) + f_{1,k}(j)$. Now, since $j \notin \mathsf{flags}_1$, $f_{1,k}(j) = 0$ for all k, and since $j \in \mathsf{flags}_0$, there is exactly one k such that $f_{0,k}(j) \neq 0$. For this k, $f_{0,k} = \mathsf{coefs}_j(\alpha_0, \alpha_1) - v$. Therefore, $T_0[j] + T_1[j] = v + \mathsf{coefs}_j(\alpha_0, \alpha_1) - v$.

4. The cases $j \in \mathsf{flags}_1 \setminus \mathsf{flags}_0$ and $j \in \mathsf{flags}_0 \cap \mathsf{flags}_1$ are similar (in the latter, there will be exactly one k where $f_{0,k}(j)$ is non-zero, and one k', possibly equal to k, where $f_{0,k'}(j)$ is non-zero).

Therefore, the $(T_0[j], T_1[j])_j$ are indeed additive shares of the outputs of coefs on input (α_0, α_1), hence $(x')^{\otimes |S_j|} \cdot (q_{0,i}, q_{1,i}, \cdots)^\intercal = Q(x')$, where Q is the $|S_j|$-variate polynomials computing $C_j(\alpha_0, \alpha_1, x)$ for any x such that $x' = x[S_j]$.
We clarify a minor technicality regarding the analysis above: since Las Vegas HSS has probability 1 of yielding correct shares of the output whenever the flags are equal to $\top$, this remains true when the randomness of Eval is sampled as $R_i \leftarrow \mathsf{PRG}(\mathsf{seed}_i)$ for party P_i. However, to guarantee correctness, we must also show that with overwhelming probability, $|\mathsf{flags}_i| \leq \lambda$. When using true random coins, this holds with overwhelming probability by a straightforward Chernoff bound. When using pseudorandom coins, as we do here, this *statistical* property holds under the assumption that the PRG is secure (it is a standard fact that the output of a secure PRG must pass all polynomial-time verifiable statistical tests); hence, correctness holds under the existence of a secure PRG. This concludes the proof of correctness.
It remains to show that FSS is a secure function secret sharing scheme. As before, it follows from the fact that HSS is a secure homomorphic secret sharing scheme for which we have a simulator $\mathsf{Sim}^{\mathsf{HSS}}$. We additionally mask each output with 2λ outputs of (2λ independent instances of) a DPF, which can also be simulated using the simulator $\mathsf{Sim}^{\mathsf{DPF}}$. □

7.4.3.2 Securely Realising Gen in Low Communication.

We describe a low-communication two-party protocol for securely distributing $(\mathsf{k}_0, \mathsf{k}_1) \xleftarrow{\$} \mathsf{FSS.Gen}(1^\lambda, C(\alpha_0, \alpha_1, \cdot))$ between two parties, P_0 and P_1, holding α_0 and α_1 as respective inputs, in the honest-but-curious model.

Private information retrieval. A private information retrieval is a two party protocol between a server S holding a vector z (the database) and a client C holding an integer i, where only the client receives an output. The security parameter λ and the length $n(\lambda) = \mathsf{poly}(\lambda) = |z|$ of the server database are a common (public) input. We let $\mathsf{View}_S(\lambda, z, i)$ denote the view of S during its interaction with C on respective inputs (z, i) with common input $(\lambda, n = |z|)$, and by $\mathsf{Out}_C(\lambda, z, i)$ the output of C after the interaction.

Definition 29 (Private Information Retrieval)**.** *A private information retrieval for database size $n = n(\lambda)$ (n-PIR) is an interactive protocol between a PPT server S holding a vector $z \in \mathbb{F}^n$ and a PPT client C holding an index $i \leq n$ which satisfies the following properties:*

- ***Correctness:*** *there exists a negligible function μ such that for every $\lambda \in \mathbb{N}$, $z \in \{0,1\}^n$, $i \in [n]$:*

$$\Pr[\mathsf{Out}_C(\lambda, z, i) = z_i] \geq 1 - \mu(\lambda).$$

- ***Security:*** *there exists a negligible function μ such that for every PPT adversary $\mathcal{A}$, large enough $\lambda \in \mathbb{N}$, $(i, j) \in [n]^2$, and $z \in \{0,1\}^n$:*

$$|\Pr[\mathcal{A}(1^{\lambda+n}, \mathsf{View}_S(\lambda, z, i)) = 1] - \Pr[\mathcal{A}(1^{\lambda+n}, \mathsf{View}_S(\lambda, z, j)) = 1]| \leq \mu(\lambda, n).$$

- ***Efficiency:*** *A PIR is* polylogarithmic *if its communication complexity $c(\lambda, n)$, measured as the worst-case number of bits exchanged between S and C (over their inputs (z, i) and their random coins), satisfies $c(\lambda, n) = \mathsf{poly}(\lambda, \log n)$.*

We define a private *shared* information retrieval (PSIR) analogeously to a private information retrieval, except that the client C with input i and the server S with input z get as output random shares of z_i.

The protocol. As in the previous constructions, we assume that there is a succinct protocol (with communication linear in the input size, up to an additive $\mathsf{poly}(\lambda)$ term) for distributing the shares of HSS. Such succinct protocols are known for all known Las Vegas HSS. Furthermore, we assume that $\mathsf{HSS.Eval}$ can be run on tuples of input shares (generated separately), rather than individual input shares; that is, given $(x^{(0)}, x^{(1)}) \leftarrow \mathsf{HSS.Share}(1^\lambda, x)$ and $(y^{(0)}, y^{(1)}) \leftarrow \mathsf{HSS.Share}(1^\lambda, y)$, $\mathsf{Eval}(i, f, (x^{(i)}, y^{(i)}))$ for $i = 0, 1$ produces shares of $f(x, y)$. We note that known constructions of Las Vegas HSS [BGI16a, BKS19] have a setup phase that generates public parameters, such that $\mathsf{HSS.Eval}$ can be run on any tuple of HSS shares generated from the same public parameters.
The protocol operates in a hybrid model, with access to the following functionalities:

- A share distribution functionality $\mathcal{F}_{\mathsf{HSS\text{-}SD}}$ (as given on Figure fig. 7.12).
- A private shared information retrieval functionality $\mathcal{F}_{\mathsf{PSIR}}$ which receives i from a client, and $(z, r) \in \mathbb{F}^n \times \mathbb{F}$ from a server. It outputs $z_i - r$ to the client, and nothing to the server. (Note that given a PIR, our reduction from PIR to PSIR securely instantiates this functionality).
- An ideal 'selection' functionality $\mathcal{F}_{\mathsf{sel}}$ which, given $(v_1^b, \cdots, v_\lambda^b) \in \mathbb{F}^\lambda$ and $(\mathsf{flag}_1^b, \cdots, \mathsf{flag}_\lambda^b) \in \{\top, \bot\}^\lambda$ from each party P_b, returns uniformly random shares of $v_i^0 + v_i^1$, where i is the first index such that $\mathsf{flag}_i^0 = \mathsf{flag}_i^1 = \top$.

- An ideal functionality $\mathcal{F}_{\mathsf{DPF}}$ which distributes DPF keys given shares of (the description of) a point function:
  - $\mathcal{F}_{\mathsf{DPF}}$ takes as input (w^b, j) from one party P_b, and $(w^{1-b}, \perp)$ from P_{1-b};
  - it defines f the point function which evaluates to $w = w^0 + w^1$ on input j (and to 0 elsewhere);
  - it outputs $(t^0, t^1) \xleftarrow{\$} \mathsf{DPF.Gen}(f)$ to P_0 and P_1.

Succinct Protocol for the LogLog-Depth FSS Scheme from Las Vegas HSS

1. P_0 and P_1 call $\mathcal{F}_{\mathsf{HSS\text{-}SD}}$ on respective inputs α_0, α_1, and receive $(s_0, s_1) \xleftarrow{\$} \mathsf{HSS.Share}(\alpha_0, \alpha_1)$. Each party P_i picks $\mathsf{seed}_i \xleftarrow{\$} \{0,1\}^\lambda$ and stretches $R_i \leftarrow \mathsf{PRG}(\mathsf{seed}_i)$.

2. For $i \in \{0,1\}$, P_i samples $(T_i, \mathsf{flags}_i) \xleftarrow{\$} \mathsf{HSS.Eval}(\mathsf{coefs}, s_i, \lambda)$ using R_i as random tape. We assume that $|\mathsf{flags}_i| = \lambda$ (if $|\mathsf{flag}_i| > \lambda$, P_i aborts the protocol; else, P_i pads $|\mathsf{flag}_i|$ with dummy items until $|\mathsf{flags}_i| = \lambda$).

3. For $i = 0, 1$, and for $k = 1$ to λ, denoting $\{j_{i,1}, \cdots, j_{i,\lambda}\} = \mathsf{flags}_i$, P_i sets $T_i[j_{i,k}]$ to 0.

4. For $i = 0, 1$, and for $k = 1$ to λ,
   (a) P_i and P_{1-i} call $\mathcal{F}_{\mathsf{PSIR}}$, where P_i plays the role of the client with input $j_{i,k}$, and P_{1-i} samples $u_{i,k}^{1-i} \xleftarrow{\$} \mathbb{F}$ and plays the role of the server with inputs $(T_{1-i}, u_{i,k}^{1-i})$. Let $u_{i,k}^i$ denote P_i's output. Note that by correctness of the PSIR scheme, $u_{i,k}^0 + u_{i,k}^1 = T_i[j_{i,k}]$.
   (b) P_i and P_{1-i} call $\mathcal{F}_{\mathsf{HSS\text{-}SD}}$ on input $j_{i,k}$ from P_i, and get respective outputs $(s_{i,k}^0, s_{i,k}^1) \xleftarrow{\$} \mathsf{HSS.Share}(j_{i,k})$.
   (c) Let $\mathsf{coef}(\alpha_0, \alpha_1, j)$ denote the function computing $\mathsf{coefs}_j(\alpha_0, \alpha_1)$. Each party P_b locally run λ independent instances $(v_{i,k,\ell}^b, \mathsf{flag}_{i,k,\ell}^b) \xleftarrow{\$} \mathsf{HSS.Eval}(\mathsf{coef}, (s_b, s_{i,k}^b), 1/2)$ for $b = 0, 1$ and $\ell = 1$ to λ.
   (d) P_0 and P_1 call $\mathcal{F}_{\mathsf{sel}}$, where P_b's inputs are all candidate shares $(v_{i,k,\ell}^b)_{\ell \leq \lambda}$ and corresponding flags $(\mathsf{flag}_{i,k,\ell}^b)_{\ell \leq \lambda}$. Recall that $\mathcal{F}_{\mathsf{sel}}$ identifies the first ℓ such that $\mathsf{flag}_{i,k,\ell}^0 = \mathsf{flag}_{i,k,\ell}^1 = \top$ (which exists with overwhelming probability), reconstructs $v_{i,k} = v_{i,k,\ell}^0 + v_{i,k,\ell}^1$ (which is equal to $\mathsf{coef}(\alpha_0, \alpha_1, j_{i,k}) = \mathsf{coefs}_{j_{i,k}}(\alpha_0, \alpha_1)$, by correctness of the HSS scheme), and outputs fresh random shares $(v_{i,k}^0, v_{i,k}^1)$ of $v_{i,k}$ to P_0 and P_1.
   (e) P_0 and P_1 call $\mathcal{F}_{\mathsf{DPF}}$, where each party P_b has an $w_{i,k}^b \leftarrow v_{i,k}^b - u_{i,k}^b$; P_i has additional input $j_{i,k}$, and P_{1-i} has additional input $\perp$. $\mathcal{F}_{\mathsf{DPF}}$ outputs $(t_{i,k}^0, t_{i,k}^1) \xleftarrow{\$} \mathsf{DPF.Gen}(f_{i,k})$ to P_0 and P_1, which are DPF shares of $f_{i,k}$, the point function evaluating to $w_{i,k} = w_{i,k}^0 + w_{i,k}^1$ on $j_{i,k}$, and 0 elsewhere.
   (f) Each party P_b outputs $\mathsf{k}_b \leftarrow (s_b, \mathsf{seed}_b, \mathsf{flags}_b, (t_{b,k}^0, t_{b,k}^1)_{k \leq \lambda})$.

Figure 7.15: A succinct protocol for distributing shares of the loglog-depth FSS scheme of Figure fig. 7.14

Theorem 16. *Assume that* $(\mathsf{HSS.Share}, \mathsf{HSS.Eval})$ *is a secure Las Vegas additive HSS scheme where* $\mathsf{HSS.Eval}$ *can run on tuples of HSS shares. Then the protocol from Figure fig. 7.15 securely computes the procedure* $\mathsf{FSS.Gen}(1^\lambda, C(\alpha_0, \alpha_1, \cdot))$ *of Figure fig. 7.14 in the* $(\mathcal{F}_{\mathsf{SD}}^{\mathsf{HSS}}, \mathcal{F}_{\mathsf{PSIR}}, \mathcal{F}_{\mathsf{sel}}, \mathcal{F}_{\mathsf{DPF}})$*-hybrid model. Furthermore, instantiating* $\mathcal{F}_{\mathsf{SD}}^{\mathsf{HSS}}$ *with a succinct HSS share distribution protocol (with communication* $\ell_0 + \ell_1 + \mathsf{poly}(\lambda)$*),* $\mathcal{F}_{\mathsf{PSIR}}$ *with a PSIR with polylogarithmic efficiency, and* $\mathcal{F}_{\mathsf{sel}}, \mathcal{F}_{\mathsf{DPF}}$ *with generic secure computation protocols, yield a protocol with total communication* $n + \mathsf{poly}(\lambda) \cdot \mathsf{polylog}(n)$*.*

Proof. Correctness follows by inspection: steps (1) to (3) directly emulate the step (1) to (3) of $\mathsf{FSS.Gen}$. In step (4.a), by correctness of the PSIR, the parties obtain additive shares $(u_{i,k}^0, u_{i,k}^1)$ of $T_i[j_{i,k}]$. Step (4.b) to (4.d) let the parties generate additive shares of the value $\mathsf{coefs}_{j_{i,k}}(\alpha_0, \alpha_1)$ – that is, the value that should have been shared in $(T_0[j_{i,k}], T_1[j_{i,k}])$ if there was no error. This is done by first secure sharing $j_{i,k}$ with $\mathsf{HSS.Share}$, then running λ times the $\mathsf{HSS.Eval}$ algorithm with error probability $1/2$. Except with probability $1/2^\lambda$, it necessarily holds that on one of the λ executions, both parties got a $\top$ flag (indicating no error). Step (4.d) runs a generic protocol which, given the λ outputs of each party and their flags, identifies such an execution, reconstructs the output (which is equal to $\mathsf{coefs}_{j_{i,k}}(\alpha_0, \alpha_1)$ by Las Vegas correctness of HSS), and re-shares it (this is necessary to avoid leaking the position of the first correct share). The generic protocol in (4.e) outputs keys for a point function $f_{i,k}$ which evaluates to $w_{i,k}$ on $j_{i,k}$. We have

$$w_{i,k} = w_{i,k}^0 + w_{i,k}^1 = -u_{i,k}^0 - u_{i,k}^1 + v_{i,k}^0 + v_{i,k}^1 = -T_i[j_{i,k}] + \mathsf{coefs}_{j_{i,k}}(\alpha_0, \alpha_1),$$

hence $f_{i,k}$ matches its definition in $\mathsf{FSS.Gen}$. This concludes the proof of correctness. We turn our attention to security. Assume that P_1 is corrupted; the other case follows symmetrically. The simulator Sim, given the target output $\mathsf{k}_1 \leftarrow (s_1, \mathsf{seed}_1, \mathsf{flags}_1, (t_{1,k}^0, t_{1,k}^1)_{k \le \lambda})$ of P_1, emulates P_0 as follows (note that steps 2, 3, and 4.c only require local computation):

- (Step 1) It emulates $\mathcal{F}_{\mathsf{HSS\text{-}SD}}$ and sets P_1's output to s_1.
- (Step 4.a) It emulates the λ invocations of $\mathcal{F}_{\mathsf{PSIR}}$ where P_1 plays the role of the client (i.e. when $i = 1$), sampling P_1's output $u_{1,k}^1$ uniformly over $\mathbb{F}$.
- (Step 4.b) For $i = 0, 1$ and $k = 1$ to λ, it uses $\mathsf{Sim}^{\mathsf{HSS}}$ to simulate P_1's HSS share of $j_{i,k}$. Then, it emulates $\mathcal{F}_{\mathsf{SD}}^{\mathsf{HSS}}$ and sets P_1's outputs to be the simulated shares of $j_{i,k}$.
- (Step 4.d) It emulates the 2λ invocations of $\mathcal{F}_{\mathsf{sel}}$, and set P_1's outputs from the functionality to independent uniformly random values.
- (Step 4.e) It emulates the 2λ invocations of $\mathcal{F}_{\mathsf{DPF}}$, and set P_1's outputs from the functionality to $(t_{1,k}^0, t_{1,k}^1)_{k \le \lambda}$.

Security follows from a sequence of straightforward hybrids: H_0 is the initial game. $H_1^{i,k}$ replaced the HSS share of $j_{i,k}$ by a simulated HSS share using $\mathsf{Sim}^{\mathsf{HSS}}$, for $i = 0$ to 1 and $k = 1$ to λ. Indistinguishability follows by 2λ invocations of the HSS security. H_2 replaces the λ invocations of $\mathcal{F}_{\mathsf{PSIR}}$ where P_1 plays the role of the client by emulations of the functionality with a uniformly random output $u_{1,k}^1$; this game is perfectly indistinguishable from $H_1^{1,\lambda}$, since the outputs are distributed exactly as in the

honest game. H_3 replaces the 2λ invocations of $\mathcal{F}_{\mathsf{sel}}$ by emulations of the functionality with uniformly random outputs, which is perfectly indistinguishable from H_2, since the outputs of $\mathcal{F}_{\mathsf{sel}}$ are distributed exactly as in H_2. Eventually, H_3 replaces the 2λ invocations of $\mathcal{F}_{\mathsf{DPF}}$ by emulations of the functionality with outputs $t^i_{1,k}$ for $i = 0, 1$ and $k = 1$ to λ. Since the outputs are the same as in H_3, this game is perfectly indistinguishable from the previous game. This concludes the proof. □

7.5 Instantiations

In section 7.5.1, we combine the results of sections 7.2 to 7.4 and achieve sublinear secure computation from generic assumptions (HSS and forms of PIR/OLE). In section 7.5.2, we build four-party compact and additive HSS for loglog-depth correlations from standard assumptions (DCR and constant-locality PRGs). In section 7.5.3, we show how to combine all the above (as well as existing constructions of 2-party HSS) in order to build sublinear secure 3- and 5-party computation from standard assumptions not previously known to imply it (in particular, they are not known to imply FHE).

7.5.1 Sublinear-Communication Secure Multiparty Computation from PIR and Additive HSS

Section 7.3 established that $\mathcal{F}^{\mathsf{FSS}}_{\mathsf{OE}}$ for local FSS schemes can be based on batch OPE (with correlated inputs) and section 7.4 builds local FSS schemes (such that $\mathcal{F}^{\mathsf{FSS}}_{\mathsf{SD}}$ can be realised with low communication) from additive HSS (with or without errors). Plugging these two constructions into the template of section 7.2 yields sublinear secure multiparty computation from batch OPE and additive HSS.

Theorem 17 (Sublinear-Communication Secure $(N+1)$-Party Computation of Shallow Circuits)**.** *Let $N \geq 2$ be a number of parties, and let $C\colon \mathbb{F}^n \to \mathbb{F}^m$ be a depth-d ($d \leq \log\log n - \log\log\log n$) arithmetic circuit with $n = \ell_0 + \ell_1 + \cdots + \ell_N$ inputs over $\mathbb{F}$. Assuming the existence of:*

- *A family of PRGs $G_i\colon \{0,1\}^\lambda \to \mathbb{F}^{\ell_i}$ for $i \in [N]$,*
- *An N-party compact and additive single-function HSS scheme for any function in the class $\{\mathsf{coefs}_{\alpha_1,\ldots,\alpha_N}\colon (\alpha_1,\ldots,\alpha_N) \in \mathbb{F}^{\ell_1} \times \cdots \times \mathbb{F}^{\ell_N}\}$, where $\mathsf{coefs}_{x_1,\ldots,x_N}$ is the function which, on input $(K_1,\ldots,K_N) \in (\{0,1\}^\lambda)^N$, computes the (polynomially many) coefficients of the representation of $C_j(\cdot, \alpha_1 - G_1(K_1), \ldots, \alpha_N - G_N(K_N))$ as ℓ_0-variate polynomials for $j = 1$ to m,*
- *A protocol for UC-securely realising $\mathcal{F}_{\mathsf{corrOPE}}$ using communication $\mathsf{Comm}_{\mathsf{corrOPE}}(k, N_{\mathit{var}}, \deg, w)$, where k is the number of OPEs in the batch, N_{var} is the number of variables of each polynomial, $\deg$ is the degree of each polynomial, and w is the size of the joint input vector,*

There exists a protocol using $(N+\lambda)^{\mathcal{O}(1)} + N \cdot [(N-1)\cdot n + m] \cdot \log|\mathbb{F}| + N \cdot \mathsf{Comm}_{\mathsf{corrOPE}}(m, 2^d, 2^d, n)$ bits of communication to securely compute C amongst $(N+1)$ parties (that is, to UC-securely realise $\mathcal{F}_{\mathsf{SFE}}(C)$) in the presence of a semi-honest adversary statically corrupting any number of parties.

Proof. The proof of theorem 17 is obtained by combining the results of sections 7.2 to 7.4. Our starting point is the generic template of theorem 14 in the $(\mathcal{F}^{\mathsf{FSS}}_{\mathsf{SD}}, \mathcal{F}^{\mathsf{FSS}}_{\mathsf{OE}})$-hybrid model, which uses $N \cdot m \cdot \log|\mathbb{F}|$ bits of communication and makes a single

call to $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}$, and N to $\mathcal{F}_{\mathsf{OE}}^{\mathsf{FSS}}$. We use the FSS scheme of lemmas 15 and 17, for which, by lemma 14, each call to $\mathcal{F}_{\mathsf{OE}}^{\mathsf{FSS}}$ can be implemented using communication $\mathsf{Comm}_{\mathsf{corrOPE}}(m, 2^d, 2^d, n)$ and the single call to $\mathcal{F}_{\mathsf{SD}}^{\mathsf{FSS}}$ can be implemented using communication $(N \cdot \lambda)^{\mathcal{O}(1)} + N(N-1) \cdot n \cdot \log |\mathbb{F}|$. □

7.5.2 Four-Party Additive HSS from DCR

In this section, we build a 4-party compact homomorphic secret sharing scheme for the class of loglog-depth circuits. Our starting point is the (non compact) 4-party HSS for constant degree polynomials recently described in [COS+22]. At a high level, the scheme works by *nesting* a 2-party HSS scheme inside another 2-party HSS scheme. Concretely, let $\mathsf{HSS}_{\mathsf{in}}$ and $\mathsf{HSS}_{\mathsf{out}}$ be two 2-party HSS schemes. Then, the following is a 4-party HSS:

- $\mathsf{HSS.Share}(x)$: run $(x^{(0)}, x^{(1)}) \leftarrow \mathsf{HSS.Share}_{\mathsf{in}}(x)$. For $b = 0, 1$, run $(x^{(b,0)}, x^{(b,1)}) \leftarrow \mathsf{HSS.Share}_{\mathsf{out}}(x^{(b)})$. Output $(x^{(0,0)}, x^{(0,1)}, x^{(1,0)}, x^{(1,1)})$.
- $\mathsf{HSS.Eval}(i, f, x^{(i)})$: parse i as $(b, c) \in \{0, 1\}^2$. Define $G_{\mathsf{in}}(f) : x^{(b,c)} \to \mathsf{HSS}_{\mathsf{in}}.\mathsf{Eval}(b, f, x^{(b,c)})$ and run $y^{(i)} \leftarrow \mathsf{HSS}_{\mathsf{out}}.\mathsf{Eval}(c, G_{\mathsf{in}}(f), x^{(b,c)})$.

Therefore, to get 4-party HSS for a function class $\mathcal{F}$, we need (1) a 2-party $\mathsf{HSS}_{\mathsf{in}}$ for $\mathcal{F}$, and (2) a 2-party $\mathsf{HSS.out}$ for the class $\mathcal{F}' = G_{\mathsf{in}}(\mathcal{F})$. We now state the resulting theorem in theorem 18.

Theorem 18 (Four-Party Additive HSS for Constant-Depth Circuits from DCR)**.** *Assuming the superpolynomial hardness of* DCR *and the existence of PRGs with constant locality, there exists a four-party HSS scheme for the class of loglog-depth circuits with n_{in} inputs; the HSS scheme has share size $n_{in}(1 + o(1))$. Furthermore, there exists a protocol with communication complexity $n_{in} \cdot (4 + o(1))$ (for large enough n_{in}) for securely realising the four-party functionality $\mathcal{F}_{\mathsf{SD}}^{\mathsf{HSS}}$ of fig. 7.12 for the generation of HSS shares of the concatenation of the parties inputs.*

7.5.2.1 4-party HSS from DCR.

The work of [COS+22] shows how to instantiate the above template, using the recent DCR-based HSS scheme of [OSY21] to instantiate both $\mathsf{HSS}_{\mathsf{in}}$ and $\mathsf{HSS}_{\mathsf{out}}$, when the class $\mathcal{F}$ is restricted to constant-degree multivariate polynomials. Consider for simplicity the evaluation of a single degree-c monomial $\prod_{i=1}^{c} x_i$ (handling arbitrary polynomials is done by computing shares of the deg-c monomials separately, and summing the shares). Fix a modulus N for the Paillier encryption scheme, and let d be the Paillier secret key, i.e. , an integer such that given any Paillier encryption $C = (1+N)^x R^n \bmod N^2$ of an input x, we have $C^d = (1+N)^m \bmod N^2$.
In the scheme of [OSY21], an HSS share of x_i contains (1) additive shares (over $\mathbb{Z}$) of the secret key d and (2) Paillier encryption $C_i, (C_{i,j})_{j \le |d|}$ of $x_i, (x_i \cdot d_j)_{j \le |d|}$, where the d_j are the bits of d. A core observation is that the ciphertexts $C_i, C_{i,j}$ can remain public (in the sense that they will be directly included in all 4-party shares, and not reshared), since they are included in all shares: only the shares of d must be re-shared (bitwise) with the outer scheme. Now, the homomorphic evaluation of $f(x_1, \cdots, x_c) = \prod_{i=1}^{c} x_i$ consists in $c - 1$ sequential homomorphic multiplication, where each homomorphic multiplication boils down to parallel calls to the function

$$\mathsf{Mul}_{C,v,N} : d \to \mathsf{DDLog}(C^d \bmod N^2) + v \bmod N,$$

where v is a value known to both partie (concretely, $v = F_k(\mathsf{id})$ where id is public and k is a PRF key known to both parties, which can be included "in the clear" in the four 4-party shares), DDLog is the function $\mathsf{DDLog} : X \to \lfloor X/N \rfloor \cdot (X \bmod N)^{-1} \bmod N$, and C is one of the Paillier ciphertexts. Now, since the ciphertexts C are known during the homomorphic evaluation of $\mathsf{Mul}_{C,v,N}$ by the outer scheme, we can rewrite

$$C^d = \prod_{j=1}^{|d|} (C^{2^{j-1}})^{d_j} \bmod N^2, \qquad (C^d \bmod N)^{-1} = \prod_{j=1}^{|d|} (C^{-2^{j-1}})^{d_j} \bmod N^2.$$

Since iterated products, modular reductions, rounding, and integer division are all in NC^1, the entire computation of Mul is therefore an NC^1 function (and so are parallel calls to Mul). From here, [COS+22] concludes that for every constant c, the homomorphic computation of $\prod_{i=1}^{c} x_i$ inside the inner HSS remains an NC^1 function overall. Therefore, it suffices for the outer HSS to use another DCR-based HSS (with a different Paillier modulus $N' = O(N)$), since the latter handles all NC^1 computations.

7.5.2.2 Handling loglog-depth circuits.

The above nesting approach is limited to constant degree polynomials. At a high level, this stems from the fact that computing c sequential homomorphic Mul requires a runtime of the form $\mathsf{poly}(\lambda)^c$, where $\mathsf{poly}(\lambda)$ denotes the runtime of computing a single Mul operation. This overhead stems from the fact that the outer HSS natively handles only functions represented as restricted multiplication straight-line (RMS) programs, and converting an arbitrary circuit to an RMS program incurs a cost which is exponential in the circuit depth. The inputs to (the parallel calls to) Mul have size $\mathsf{poly}(\log N) = \mathsf{poly}(\lambda)$, the depth of the Mul circuit (which is in NC^1) is therefore $O(\log \lambda)$, hence the overall depth of the homomorphic computation is $O(c \cdot \log \lambda)$, hence the $\mathsf{poly}(\lambda)^c = 2^{O(c \cdot \log \lambda)}$ overhead.
To circumvent this limitation, we rely on complexity leveraging, and make $\log N$ slightly subpolynomial in λ. Looking ahead, this will translate to assuming the superpolynomial hardness of the DCR assumption. Writing $\log N = \kappa$ and by the above analysis, the overall runtime of evaluating a degree-c monomial is $\kappa^{\mathsf{cst} \cdot c}$, for an appropriate constant cst. Setting $\kappa \leftarrow \lambda^{1/c}$, the above becomes polynomial in λ, at the cost of assuming that DCR is secure against $\mathsf{poly}(\lambda) = \kappa^{O(c)}$ adversaries. For example, setting $c = \log(\lambda) = O(\log \kappa)$ means that we need the DCR assumption to hold against adversaries running in time $\kappa^{O(\log \kappa)}$, i.e. , assuming the superpolynomial hardness of DCR. Now, over $\mathbb{F}_2$, any c-local function (hence, for example, every $\log c$-depth boolean circuit) can be written as a sum of monomials of degree at most c.

7.5.2.3 Compactness and succinct share distribution.

The above scheme is not compact: a 4-party HSS share of a length-m vector $\vec{x}$ has size $\mathsf{poly}(\kappa) \cdot m$. We show here how to transform it into a compact scheme, with share size $m + o(m) \cdot \mathsf{poly}(\kappa)$, assuming the existence of local pseudorandom generators. Local PRGs have been introduced in a seminal work of Goldreich [Gol00]. Since their introduction, they have been investigated in numerous works [MST03, AHI04, CEMT09, CEMT14, OW14, AL16, CDM+18], and are regularly used to achieve advanced cryptographic primitives, such as secure computation with constant computational overhead [IKOS08, ADI+17] or more recently indistinguishability obfuscation [JLS21, JLS22]. A local PRG G with stretch s stretches a seed seed

of size t into a length-t^{1+s} pseudorandom string, such that every bit of $G(\mathsf{seed})$ is a function of a constant number of bits of seed.
To achieve compactness, we use a standard *hybrid encapsulation* technique with a local PRG. Concretely, let HSS be the non-compact 4-party HSS scheme constructed above. We construct a new scheme HSS' as follows:

- $\mathsf{HSS}'.\mathsf{Share}(x)$: given a length-m vector x of inputs, sample four seeds $(\mathsf{seed}_i)_{i\leq 4}$ for a local PRG G with stretch s and seed size $m^{1/(1+s)}$. Compute $(x^{(i)})_{i\leq 4} \leftarrow \mathsf{HSS}.\mathsf{Share}(\mathsf{seed}_1||\mathsf{seed}_2||\mathsf{seed}_3||\mathsf{seed}_4)$, and $y \leftarrow x \oplus G(\mathsf{seed}_1) \oplus G(\mathsf{seed}_2) \oplus G(\mathsf{seed}_3) \oplus G(\mathsf{seed}_4)$. Output $(x^{(i)}, z)$ to each party i.

- $\mathsf{HSS}'.\mathsf{Eval}(i, f, (x^{(i)}, z))$: define the function

$$f'_z : (\mathsf{seed}_i)_{i\leq 4} \to f\left(z \oplus \bigoplus_{i=1}^{4} G(\mathsf{seed}_i)\right).$$

  Output $y^{(i)} \leftarrow \mathsf{HSS}.\mathsf{Eval}(i, f'_z, x^{(i)})$.

Correctness is clear by inspection. The security analysis of HSS' proceeds in two simple games: in the first game, using the security of HSS, simulate the shares of $(\mathsf{seed}_1, \cdots, \mathsf{seed}_4)$. In the second game, since the seeds seed_i are not used anymore, invoke the pseudorandomness of G to replace z by a uniformly random string. For efficiency, observe that since G has constant locality (and constant depth), whenever f is a loglog-depth function, so is f'.
It remain to discuss compactness. The size of a share is $|x^{(i)}| + |z| = \mathsf{poly}(\kappa) \cdot m^{1/(1+s)} + m = m + o(m) \cdot \mathsf{poly}(\kappa)$. Furthermore, the scheme admits a straightforward succinct protocol for securely distributing shares of an input vector x. Assume that the parties P_i have shares x_i of the input vector x (the case where x is the concatenation of their joint input is directly implied by this case). Each party P_i locally samples seed_i, and the party jointly run a generic secure computation protocol (e.g. using DCR-based oblivious transfer, with security parameter κ) for securely computing $\mathsf{HSS}.\mathsf{Share}(\mathsf{seed}_1||\mathsf{seed}_2||\mathsf{seed}_3||\mathsf{seed}_4)$. Then, each party P_i broadcasts $z_i \leftarrow x_i \oplus G(\mathsf{seed}_i)$, and all parties reconstruct $z = \bigoplus_{i=1}^{4} z_i$. The total communication of the protocol is $4m + \mathsf{poly}(\kappa, o(m))$, which is $m \cdot (4 + o(1))$ for a large enough m.

7.5.3 Sublinear-Communication Secure Multiparty Computation from New Assumptions

Combining section 7.5.1 with instantiations of $\mathsf{corrSPIR}$ and additive HSS from the literature (and section 7.5.2) yields sublinear-communication secure 3- and 5-party computation of shallow boolean circuits from a variety of assumptions. Layered boolean circuits are boolean circuits whose gates can be arranged into layers such that any wire connects adjacent layers. It is well-known from previous works [BGI16a, Cou19, CM21] that sublinear protocols for low-depth circuits translate to sublinear protocols for general layered circuits: the parties simply cut the layered circuit into low-depth "chunks", and securely evaluate it chunk-by-chunk. For each chunk, a sublinear secure protocol is invoked to compute the low-depth function which maps shares of the values on the first layer to shares of the values on the first layer of the next chunk.

Theorem 19 (Sublinear-Communication (N+1)-PC from New Assumptions)**.**

- **3-PC of Shallow Circuits:** *Let* $C\colon \{0,1\}^n \to \{0,1\}^m$ *be a size-*s*, depth-*d *(*$d \leq \log\log s - \log\log\log s$*) boolean circuit. Let* $\epsilon \in (0,1)$*. Assuming the Learning Parity with Noise (LPN) assumption with dimension* $\mathsf{dim} = \mathsf{poly}(\lambda)$*, number of samples* $\mathsf{num} = (n+m)^{1/3} \cdot \lambda^{\mathcal{O}(1)}$*, and noise rate* $\rho = \mathsf{num}^{\epsilon-1}$ *(for some constant* $0 < \epsilon < 1$*) together with any of the following additional computational assumptions:*

  - *Decisional Diffie-Hellman*
  - *Learning with Errors with polynomial-size modulus*
  - *Quadratic Residuosity and Superpolynomial* $\mathbb{F}_2$-LPN *(*i.e. *assuming the security against time-*$\lambda^{2\log\lambda}$ *adversaries of* $\mathbb{F}_2$-LPN *with dimension* $\lambda^{\log\lambda}$*,* $2\lambda^{\log\lambda}$ *samples, and rate* $\lambda/(2\lambda^{\log\lambda})$*).*

  There exists a 3-party protocol with communication complexity $\lambda^{\mathcal{O}(1)} + \mathcal{O}(n+m+2^{d+2^d}\cdot\mathsf{poly}(\lambda)\cdot\mathsf{polylog}(n)\cdot((n+m)^{2/3}+(n+m)^{(1+2\epsilon)/3}))$ *to securely compute* C *(that is, to UC-securely realise* $\mathcal{F}_{\mathsf{SFE}}(C)$*) in the presence of a semi-honest adversary statically corrupting any number of parties. In particular, if* $d \leq (\log\log s)/4$ *the communication complexity is* $\lambda^{\mathcal{O}(1)} + \mathcal{O}(n+m+\sqrt{s}\cdot\mathsf{poly}(\lambda)\cdot\mathsf{polylog}(n)\cdot((n+m)^{2/3}+(n+m)^{(1+2\epsilon)/3}))$ *(for some arbitrarily small constant* $0 < \delta < 1/2$*), which is sublinear in the circuit-size, as detailed in remark 6.*

- **3-PC of Layered Boolean Circuits:** *Let* $C\colon \{0,1\}^n \to \{0,1\}^m$ *be a size-*s*, depth-*d *layered boolean circuit. Let* $\epsilon \in (0,1)$*. Assuming the Learning Parity with Noise (LPN) assumption with dimension* $\mathsf{dim} = \mathsf{poly}(\lambda)$*, number of samples* $\mathsf{num} = ((s/d)^2/s^{\epsilon})^{1/3} \cdot \mathsf{poly}(\lambda)$*, and noise rate* $\rho = \mathsf{num}^{-1/2}$ *together with any of the following additional computational assumptions:*

  - *Decisional Diffie-Hellman*
  - *Learning with Errors with polynomial-size modulus*
  - *Quadratic Residuosity and Superpolynomial* $\mathbb{F}_2$-LPN *(*i.e. *assuming the security against time-*$\lambda^{2\log\lambda}$ *adversaries of* $\mathbb{F}_2$-LPN *with dimension* $\lambda^{\log\lambda}$*,* $2\lambda^{\log\lambda}$ *samples, and rate* $\lambda/(2\lambda^{\log\lambda})$*).*

  There exists a 3-party protocol with communication complexity $\mathcal{O}(n+m+d^{1/3}\cdot s^{2(1+\epsilon)/3}\cdot\mathsf{poly}(\lambda) + s/(\log\log s))$ *to securely compute* C *(that is, to UC-securely realise* $\mathcal{F}_{\mathsf{SFE}}(C)$*) in the presence of a semi-honest adversary statically corrupting any number of parties. In particular, if* $d = o(s^{1-\epsilon}/\mathsf{poly}(\lambda))$ *(*i.e. *the circuit is not too "tall and skinny") the communication complexity is* $\mathcal{O}(n+m+\frac{s}{\log\log s})$*, which is sublinear in the circuit-size.*

- **5-PC of Shallow Circuits:** *Let* $\epsilon \in (0,1)$*. Assuming the existence of a constant-locality PRG with polynomial stretch, there exists a constant* $c \geq 3$ *such that for any boolean circuit* $C\colon \{0,1\}^n \to \{0,1\}^m$ *of size* s *and depth* d *(*$d \leq (\log\log s - \log\log\log s)/2^c$*), assuming the superpolynomial Decision Composite Residuosity (DCR) assumption, the Learning Parity with Noise (LPN) assumption with dimension* $\mathsf{dim} = \mathsf{poly}(\lambda)$*, number of samples* $\mathsf{num} = (n+m)^{1/3}\cdot\lambda^{\mathcal{O}(1)}$*, and noise rate* $\rho = \mathsf{num}^{\epsilon-1}$ *(for some constant* $0 < \epsilon < 1$*), as well as any of the following computational assumptions:*

– *Decisional Diffie-Hellman (DDH)*

– *Learning with Errors with polynomial-size modulus (poly-modulus LWE)*

– *Quadratic Residuosity (QR) and Superpolynomial* $\mathbb{F}_2$-LPN *(*i.e. *assuming the security against time-*$\lambda^{2\log\lambda}$ *adversaries of* $\mathbb{F}_2$-LPN *with dimension* $\lambda^{\log\lambda}$*,* $2\lambda^{\log\lambda}$ *samples, and rate* $\lambda/(2\lambda^{\log\lambda})$*).*

There exists a 5-party protocol with communication complexity $\lambda^{\mathcal{O}(1)} + \mathcal{O}(n + m + 2^{d/2^c+2^{d/2^c}} \cdot \mathsf{poly}(\lambda) \cdot \mathsf{polylog}(n) \cdot ((n+m)^{2/3} + (n+m)^{(1+2\epsilon)/3}))$ *to securely compute* C *(that is, to UC-securely realise* $\mathcal{F}_{\mathsf{SFE}}(C)$*) in the presence of a semi-honest adversary statically corrupting any number of parties. In particular, if* $d \leq (\log\log s)/2^{c+2}$ *the communication complexity is* $\lambda^{\mathcal{O}(1)} + \mathcal{O}(n + m + \sqrt{s} \cdot \mathsf{poly}(\lambda) \cdot \mathsf{polylog}(n) \cdot ((n+m)^{2/3} + (n+m)^{(1+2\epsilon)/3}))$ *(for some arbitrarily small constant* $0 < \epsilon < 1/2$*), which is sublinear in the circuit-size, as detailed in remark 6.*

- **5-PC of Layered Boolean Circuits:** *Let* $\epsilon \in (0,1)$*. Assuming the existence of a constant-locality PRG with polynomial stretch, there exists a constant* $c \geq 3$ *such that for any layered boolean circuit* $C\colon \{0,1\}^n \to \{0,1\}^m$ *of size* s *and depth* d*, assuming the superpolynomial Decision Composite Residuosity (DCR) assumption, assuming the Learning Parity with Noise (LPN) assumption with dimension* $\mathsf{dim} = \mathsf{poly}(\lambda)$*, number of samples* $\mathsf{num} = ((s2^c/d)^2/s^\epsilon)^{1/3} \cdot \mathsf{poly}(\lambda)$*, and noise rate* $\rho = \mathsf{num}^{-1/2}$ *together with any of the following additional computational assumptions:*

  – *Decisional Diffie-Hellman*

  – *Learning with Errors with polynomial-size modulus*

  – *Quadratic Residuosity and Superpolynomial* $\mathbb{F}_2$-LPN *(*i.e. *assuming the security against time-*$\lambda^{2\log\lambda}$ *adversaries of* $\mathbb{F}_2$-LPN *with dimension* $\lambda^{\log\lambda}$*,* $2\lambda^{\log\lambda}$ *samples, and rate* $\lambda/(2\lambda^{\log\lambda})$*).*

  There exists a 5-party protocol with communication complexity $\mathcal{O}(n + m + d^{1/3} \cdot s^{2(1+\epsilon)/3} \cdot \mathsf{poly}(\lambda) + s/(\log\log s))$ *to securely compute* C *(that is, to UC-securely realise* $\mathcal{F}_{\mathsf{SFE}}(C)$*) in the presence of a semi-honest adversary statically corrupting any number of parties. In particular, if* $d = o(s^{1-\epsilon}/\mathsf{poly}(\lambda))$ *(*i.e. *the circuit is not too "tall and skinny") the communication complexity is* $\mathcal{O}(n + m + \frac{s}{\log\log s})$*, which is sublinear in the circuit-size.*

We refer to [BCM23] for the proof of theorem 19.
Note that combining the works of [BBDP22, OSY21] seems to implicitly yield rate-1 batch OT from DCR, and in turn correlated SPIR [BCM22]: if true, the assumptions for sublinear-communication five-party MPC can be simplifed to constant-locality PRG, LPN, and superpolynomial DCR (without the need for DDH, LWE, or QR). Since this claim was never made formally, we do not use it.
We conclude by remarking that while this may not be immediately apparent due to the complicated expressions, the above communication complexities do indeed qualify as "sublinear in the circuit-size".

Remark 6 (The Expressions of Theorem 19 are Sublinear in the Circuit Size)**.** *Recall that a protocol for securely computing a size-*s *circuit with* n *inputs and* m *outputs*

is sublinear in the circuit-size *if its communication complexity is of the form* $\lambda^{\mathcal{O}(1)} + \mathsf{poly}(n+m) + o(s)$*, where* poly *is some fixed polynomial. The communication of our protocols for loglog-depth circuits, both in the 3- and the 5-party case, are sublinear in the circuit-size. For 3PC and 5PC of loglog-depth circuits, the expression is the following:*

$$\lambda^{\mathcal{O}(1)} + \mathcal{O}(n + m + \sqrt{s} \cdot \mathsf{poly}(\lambda) \cdot \mathsf{polylog}(n) \cdot ((n+m)^{2/3} + (n+m)^{(1+2\epsilon)/3})).$$

where $\epsilon \in (0,1)$ *is some constant tied to the strength of the* LPN *assumption. Because we view* s *as an arbitrarily large polynomial in the security parameter (in other words we are interested in an asymptotic notion of sublinearity), there exists an arbitrarily small constant* $\delta \in (0, \frac{1}{2})$ *such that* $\mathsf{poly}(\lambda) \leq s^{\delta}$*. Therefore the complexity can be simplified as:*

$$\lambda^{\mathcal{O}(1)} + \mathcal{O}(n + m + s^{\frac{1}{2}+\delta} \cdot \mathsf{polylog}(n) \cdot ((n+m)^{2/3} + (n+m)^{(1+2\epsilon)/3})).$$

Whenever $s^{\delta} \geq \mathsf{polylog}(n) \cdot ((n+m)^{2/3} + (n+m)^{(1+2\epsilon)/3})$*, the above expression is* $\lambda^{O(1)} + \mathcal{O}(n + m + s^{1+2\delta})$*. Whenever* $s^{\delta} < \mathsf{polylog}(n) \cdot ((n+m)^{2/3} + (n+m)^{(1+2\epsilon)/3})$*, the entire expression is already some fixed polynomial in* $n+m$*. Therefore, our final complexity is of the form* $\lambda^{\mathcal{O}(1)} + \mathsf{poly}_{\delta}(n+m) + s^{\frac{1}{2}+2\delta}$*.*

Chapter 8

Open Questions

In this thesis, we presented new ways to break the *circuit-size barrier* for secure computation. Specifically, we put forward two new ways of breaking the circuit-size barrier for secure two-party computation, by introducing the notions of "single-circuit HSS" and "correlated symmetric PIR". This extends the set of assumptions known to imply sublinear-communication two-party computation to quasipolynomial LPN and {LPN+QR}. Finally, by combining our two novel approaches we present the first sublinear-communication for secure *multiparty* computation without FHE. Furthermore, in the two-party setting, we show how to upgrade previous HSS-based approaches to *one-sided statistical security*, which is the strongest possible security guarantee.

We now list a few technical questions left open by our work.

Is decomposability inherent to rate-1 batch OT? Our idea of achieving sublinear-communication secure two-party computation from correlated SPIR, and basing the latter on decomposable batch OT, was conceived indendepently of Brakerski, Branco, Döttling, and Pu's [BBDP22] instantiation. This raises the question of whether it is merely a fortunate coincidence that their construction happens to be decomposable, or whether decomposability is an inherent property of two-round rate-1 batch OT.
We cannot hope to prove a statement of the form "any rate-1 batch OT is decomposable", without ruling out the existence of rate-1 batch OT altogether. Indeed, it is possible to turn a decomposable rate-1 batch OT into a rate-1 batch OT which is *not* decomposable (*e.g.* by applying the permutation $(x_1, \ldots, x_k) \mapsto (x_1 \oplus x_2, x_2 \oplus x_3, \ldots, x_{n-1} \oplus x_n, x_n \oplus x_1)$ to the decomposable part, which preserves correctness, security, and rate, but breaks decomposability). Therefore a positive answer to this question would be a statement of the form "without loss of generality, a rate-1 batch OT can be assumed to be decomposable".

Open Question 1. *Does the existence of two-round rate-1 batch OT imply the existence of decomposable batch OT?*

A positive answer to this question would be surprising, but would constitute a significant step towards understanding the minimal assumptions for low-communication secure computations (depending on which assumptions are made, if any, to equip a rate-1 batch OT scheme with decomposablity.).

Beyond Boolean Circuits. All of our schemes which use correlated SPIR (i.e. those of chapters 6 and 7 and [BCM22, BCM23]) are restricted to computing circuits over

constant-size rings. Extending our techniques to arithmetic circuits over more general rings can be done, given the primitive of *correlated Oblivious Polynomial Evaluation (correlated OPE)* [BCM23], which is an arithmetic analog of correlated SPIR.

Open Question 2. *Can correlated OPE (for "Mix-and-Match" queries) [BCM23] over rings of super-constant size be built from assumptions not known to imply FHE?*

Beyond Layered Circuits. The only polynomial-computation sublinear-communication protocols supporting the class $\mathsf{P/poly}$ are based on FHE. All other protocols achieve low-communication secure computation for "low-yet-super-constant-depth" circuits, which in turn yields secure computation of arbitrary depth layered circuits using a slightly sublinear amount of computation. The reason other approaches are stuck at low-depth computations is either because computation scales doubly exponentially in the circuit depth, or because they rely on Barrington's theorem [Bar89].

Open Question 3. *Can the circuit-size barrier be broken* for all polynomial-size circuits *in the correlated randomness model, but using polynomial computation, or with computational security, but "without FHE".*

Bibliography

[ADI+17] Benny Applebaum, Ivan Damgård, Yuval Ishai, Michael Nielsen, and Lior Zichron. Secure arithmetic computation with constant computational overhead. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 223–254. Springer, Heidelberg, August 2017.

[ADOS22] Damiano Abram, Ivan Damgård, Claudio Orlandi, and Peter Scholl. An algebraic framework for silent preprocessing with trustless setup and active security. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part IV*, volume 13510 of *LNCS*, pages 421–452. Springer, Heidelberg, August 2022.

[AHI04] Michael Alekhnovich, Edward A. Hirsch, and Dmitry Itsykson. Exponential lower bounds for the running time of DPLL algorithms on satisfiable formulas. In Josep Díaz, Juhani Karhumäki, Arto Lepistö, and Donald Sannella, editors, *ICALP 2004*, volume 3142 of *LNCS*, pages 84–96. Springer, Heidelberg, July 2004.

[AHI+17] Benny Applebaum, Naama Haramaty, Yuval Ishai, Eyal Kushilevitz, and Vinod Vaikuntanathan. Low-complexity cryptographic hash functions. In Christos H. Papadimitriou, editor, *ITCS 2017*, volume 4266, pages 7:1–7:31, 67, January 2017. LIPIcs.

[AIK07] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography with constant input locality. In Alfred Menezes, editor, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 92–110. Springer, Heidelberg, August 2007.

[AIK09] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography with constant input locality. *Journal of Cryptology*, 22(4):429–469, October 2009.

[AJL+12] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold FHE. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 483–501. Springer, Heidelberg, April 2012.

[AL16] Benny Applebaum and Shachar Lovett. Algebraic attacks against random local functions and their countermeasures. In Daniel Wichs and Yishay Mansour, editors, *48th ACM STOC*, pages 1087–1100. ACM Press, June 2016.

[ALSZ17] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer extensions. *Journal of Cryptology*, 30(3):805–858, July 2017.

[Bar89] David A. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in nc1. *Journal of Computer and System Sciences*, 38(1):150–164, 1989.

[BBC+20] Marshall Ball, Elette Boyle, Ran Cohen, Lisa Kohl, Tal Malkin, Pierre Meyer, and Tal Moran. Topology-hiding communication from minimal assumptions. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part II*, volume 12551 of *LNCS*, pages 473–501. Springer, Heidelberg, November 2020.

[BBC+23] Marshall Ball, Elette Boyle, Ran Cohen, Lisa Kohl, Tal Malkin, Pierre Meyer, and Tal Moran. Topology-hiding communication from minimal assumptions. *to appear in Journal of Cryptology*, 2023.

[BBDP22] Zvika Brakerski, Pedro Branco, Nico Döttling, and Sihang Pu. Batch-OT with optimal rate. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part II*, volume 13276 of *LNCS*, pages 157–186. Springer, Heidelberg, May / June 2022.

[BBKM23] Marshall Ball, Alexander Bienstock, Lisa Kohl, and Pierre Meyer. Towards topology-hiding computation from oblivious transfer. Preprint on webpage at `https://eprint.iacr.org/2023/849.pdf`, 2023.

[BCG+17] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, and Michele Orrù. Homomorphic secret sharing: Optimizations and applications. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 2105–2122. ACM Press, October / November 2017.

[BCG+19a] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Peter Rindal, and Peter Scholl. Efficient two-round OT extension and silent non-interactive secure computation. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 291–308. ACM Press, November 2019.

[BCG+19b] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 489–518. Springer, Heidelberg, August 2019.

[BCG+20] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Correlated pseudorandom functions from variable-density LPN. In *61st FOCS*, pages 1069–1080. IEEE Computer Society Press, November 2020.

[BCGI18] Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. Compressing vector OLE. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 896–912. ACM Press, October 2018.

[BCM22] Elette Boyle, Geoffroy Couteau, and Pierre Meyer. Sublinear secure computation from new assumptions. In Eike Kiltz and Vinod Vaikuntanathan, editors, *TCC 2022, Part II*, volume 13748 of *LNCS*, pages 121–150. Springer, Heidelberg, November 2022.

[BCM23] Elette Boyle, Geoffroy Couteau, and Pierre Meyer. Sublinear-communication secure multiparty computation does not require fhe. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023*, pages 159–189, Cham, 2023. Springer Nature Switzerland.

[BCP03] Emmanuel Bresson, Dario Catalano, and David Pointcheval. A simple public-key cryptosystem with a double trapdoor decryption mechanism and its applications. In Chi-Sung Laih, editor, *ASIACRYPT 2003*, volume 2894 of *LNCS*, pages 37–54. Springer, Heidelberg, November / December 2003.

[BDOZ11] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 169–188. Springer, Heidelberg, May 2011.

[Bea92] Donald Beaver. Efficient multiparty protocols using circuit randomization. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 420–432. Springer, Heidelberg, August 1992.

[BEC+23] Chris Brzuska, Christoph Egger, Geoffroy Couteau, Pihla Karanko, and Pierre Meyer. New random oracle instantiations from extremely lossy functions. Private Communication, 2023.

[Bel84] Edward G. Belaga. Locally synchronous complexity in the light of the trans-box method. In M. Fontet and K. Mehlhorn, editors, *STACS 84*, pages 129–139, Berlin, Heidelberg, 1984. Springer Berlin Heidelberg.

[BFKL94] Avrim Blum, Merrick L. Furst, Michael J. Kearns, and Richard J. Lipton. Cryptographic primitives based on hard learning problems. In Douglas R. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 278–291. Springer, Heidelberg, August 1994.

[BFKR91] Donald Beaver, Joan Feigenbaum, Joe Kilian, and Phillip Rogaway. Security with low communication overhead. In Alfred J. Menezes and Scott A. Vanstone, editors, *CRYPTO'90*, volume 537 of *LNCS*, pages 62–76. Springer, Heidelberg, August 1991.

[BG10] Zvika Brakerski and Shafi Goldwasser. Circular and leakage resilient public-key encryption under subgroup indistinguishability - (or: Quadratic residuosity strikes back). In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 1–20. Springer, Heidelberg, August 2010.

[BGI14] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In Hugo Krawczyk, editor, *PKC 2014*, volume 8383 of *LNCS*, pages 501–519. Springer, Heidelberg, March 2014.

[BGI15] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 337–367. Springer, Heidelberg, April 2015.

[BGI16a] Elette Boyle, Niv Gilboa, and Yuval Ishai. Breaking the circuit size barrier for secure computation under DDH. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 509–539. Springer, Heidelberg, August 2016.

[BGI16b] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 1292–1303. ACM Press, October 2016.

[BGI17] Elette Boyle, Niv Gilboa, and Yuval Ishai. Group-based secure computation: Optimizing rounds, communication, and computation. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part II*, volume 10211 of *LNCS*, pages 163–193. Springer, Heidelberg, April / May 2017.

[BGI+18] Elette Boyle, Niv Gilboa, Yuval Ishai, Huijia Lin, and Stefano Tessaro. Foundations of homomorphic secret sharing. In Anna R. Karlin, editor, *ITCS 2018*, volume 94, pages 21:1–21:21. LIPIcs, January 2018.

[BGI19] Elette Boyle, Niv Gilboa, and Yuval Ishai. Secure computation with preprocessing via function secret sharing. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019, Part I*, volume 11891 of *LNCS*, pages 341–371. Springer, Heidelberg, December 2019.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *20th ACM STOC*, pages 1–10. ACM Press, May 1988.

[BIKK14] Amos Beimel, Yuval Ishai, Ranjit Kumaresan, and Eyal Kushilevitz. On the cryptographic complexity of the worst functions. In Yehuda Lindell, editor, *TCC 2014*, volume 8349 of *LNCS*, pages 317–342. Springer, Heidelberg, February 2014.

[BIM+23] Elette Boyle, Yuval Ishai, Pierre Meyer, Robert Robere, and Gal Yehuda. On low-end obfuscation and learning. In Yael Tauman Kalai, editor, *14th Innovations in Theoretical Computer Science Conference (ITCS 2023)*, volume 251 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 23:1–23:28, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[BKS19] Elette Boyle, Lisa Kohl, and Peter Scholl. Homomorphic secret sharing from lattices without FHE. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 3–33. Springer, Heidelberg, May 2019.

[BKW00] Avrim Blum, Adam Kalai, and Hal Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. In *32nd ACM STOC*, pages 435–440. ACM Press, May 2000.

[BKW03] Avrim Blum, Adam Kalai, and Hal Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. *J. ACM*, 50(4):506–519, jul 2003.

[BLVW19] Zvika Brakerski, Vadim Lyubashevsky, Vinod Vaikuntanathan, and Daniel Wichs. Worst-case hardness for LPN and cryptographic hashing via code smoothing. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part III*, volume 11478 of *LNCS*, pages 619–635. Springer, Heidelberg, May 2019.

[Bon98] Dan Boneh. The decision Diffie-Hellman problem. In *Third Algorithmic Number Theory Symposium (ANTS)*, volume 1423 of *LNCS*. Springer, Heidelberg, 1998. Invited paper.

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part II*, volume 8270 of *LNCS*, pages 280–300. Springer, Heidelberg, December 2013.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In *20th ACM STOC*, pages 11–19. ACM Press, May 1988.

[CDM+18] Geoffroy Couteau, Aurélien Dupin, Pierrick Méaux, Mélissa Rossi, and Yann Rotella. On the concrete security of Goldreich's pseudorandom generator. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part II*, volume 11273 of *LNCS*, pages 96–124. Springer, Heidelberg, December 2018.

[CEMT09] James Cook, Omid Etesami, Rachel Miller, and Luca Trevisan. Goldreich's one-way function candidate and myopic backtracking algorithms. In Omer Reingold, editor, *TCC 2009*, volume 5444 of *LNCS*, pages 521–538. Springer, Heidelberg, March 2009.

[CEMT14] James Cook, Omid Etesami, Rachel Miller, and Luca Trevisan. On the one-way function candidate proposed by goldreich. *ACM Transactions on Computation Theory (TOCT)*, 6(3):14, 2014.

[CGKS95] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *36th FOCS*, pages 41–50. IEEE Computer Society Press, October 1995.

[Cha90] David Chaum. The spymasters double-agent problem: Multiparty computations secure unconditionally from minorities and cryptographically from majorities. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 591–602. Springer, Heidelberg, August 1990.

[CK89] Benny Chor and Eyal Kushilevitz. A zero-one law for Boolean privacy (extended abstract). In *21st ACM STOC*, pages 62–72. ACM Press, May 1989.

[CK93] Benny Chor and Eyal Kushilevitz. A communication-privacy tradeoff for modular addition. *Information Processing Letters*, 45(4):205–210, 1993.

[CLT22] Guilhem Castagnos, Fabien Laguillaumie, and Ida Tucker. Threshold linearly homomorphic encryption on $\mathbf{Z}/2^k\mathbf{Z}$. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part II*, volume 13792 of *LNCS*, pages 99–129. Springer, Heidelberg, December 2022.

[CM21] Geoffroy Couteau and Pierre Meyer. Breaking the circuit size barrier for secure computation under quasi-polynomial LPN. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part II*, volume 12697 of *LNCS*, pages 842–870. Springer, Heidelberg, October 2021.

[CMPR23] Geoffroy Couteau, Pierre Meyer, Alain Passelègue, and Mahshid Riahinia. Constrained pseudorandom functions from homomorphic secret sharing. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023*, pages 194–224, Cham, 2023. Springer Nature Switzerland.

[COS+22] Ilaria Chillotti, Emmanuela Orsini, Peter Scholl, Nigel Paul Smart, and Barry Van Leeuwen. Scooby: Improved multi-party homomorphic secret sharing based on fhe. In Clemente Galdi and Stanislaw Jarecki, editors, *Security and Cryptography for Networks*, pages 540–563, Cham, 2022. Springer International Publishing.

[Cou19] Geoffroy Couteau. A note on the communication complexity of multiparty computation in the correlated randomness model. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 473–503. Springer, Heidelberg, May 2019.

[CS02] Ronald Cramer and Victor Shoup. Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In Lars R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 45–64. Springer, Heidelberg, April / May 2002.

[DFH12] Ivan Damgård, Sebastian Faust, and Carmit Hazay. Secure two-party computation with low communication. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 54–74. Springer, Heidelberg, March 2012.

[DGH+20] Nico Döttling, Sanjam Garg, Mohammad Hajiabadi, Daniel Masny, and Daniel Wichs. Two-round oblivious transfer from CDH or LPN. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part II*, volume 12106 of *LNCS*, pages 768–797. Springer, Heidelberg, May 2020.

[DGI+19] Nico Döttling, Sanjam Garg, Yuval Ishai, Giulio Malavolta, Tamer Mour, and Rafail Ostrovsky. Trapdoor hash functions and their applications. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 3–32. Springer, Heidelberg, August 2019.

[DGS03] Ivan Damgård, Jens Groth, and Gorm Salomonsen. *The Theory and Implementation of an Electronic Voting System*, pages 77–99. Springer US, Boston, MA, 2003.

[DHRW16] Yevgeniy Dodis, Shai Halevi, Ron D. Rothblum, and Daniel Wichs. Spooky encryption and its applications. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 93–122. Springer, Heidelberg, August 2016.

[DLN19] Ivan Damgård, Kasper Green Larsen, and Jesper Buus Nielsen. Communication lower bounds for statistically secure MPC, with or without preprocessing. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part II*, volume 11693 of *LNCS*, pages 61–84. Springer, Heidelberg, August 2019.

[DLS21] Ivan Bjerre Damgård, Boyang Li, and Nikolaj Ignatieff Schwartzbach. More Communication Lower Bounds for Information-Theoretic MPC. In Stefano Tessaro, editor, *2nd Conference on Information-Theoretic Cryptography (ITC 2021)*, volume 199 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 2:1–2:18, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[DNNR17] Ivan Damgård, Jesper Buus Nielsen, Michael Nielsen, and Samuel Ranellucci. The TinyTable protocol for 2-party secure computation, or: Gate-scrambling revisited. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 167–187. Springer, Heidelberg, August 2017.

[DNOR16] Ivan Damgård, Jesper Buus Nielsen, Rafail Ostrovsky, and Adi Rosén. Unconditionally secure computation with reduced interaction. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 420–447. Springer, Heidelberg, May 2016.

[DNPR16] Ivan Damgård, Jesper Buus Nielsen, Antigoni Polychroniadou, and Michael Raskin. On the communication required for unconditionally secure multiplication. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part II*, volume 9815 of *LNCS*, pages 459–488. Springer, Heidelberg, August 2016.

[DPP14] Deepesh Data, Manoj Prabhakaran, and Vinod M. Prabhakaran. On the communication complexity of secure computation. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 199–216. Springer, Heidelberg, August 2014.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 643–662. Springer, Heidelberg, August 2012.

[DZ13] Ivan Damgård and Sarah Zakarias. Constant-overhead secure computation of Boolean circuits using preprocessing. In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 621–641. Springer, Heidelberg, March 2013.

[FGJS17] Nelly Fazio, Rosario Gennaro, Tahereh Jafarikhah, and William E. Skeith III. Homomorphic secret sharing from paillier encryption. In Tatsuaki Okamoto, Yong Yu, Man Ho Au, and Yannan Li, editors, *ProvSec 2017*, volume 10592 of *LNCS*, pages 381–399. Springer, Heidelberg, October 2017.

[FKN94] Uriel Feige, Joe Kilian, and Moni Naor. A minimal model for secure computation (extended abstract). In *26th ACM STOC*, pages 554–563. ACM Press, May 1994.

[FY92] Matthew K. Franklin and Moti Yung. Communication complexity of secure computation (extended abstract). In *24th ACM STOC*, pages 699–710. ACM Press, May 1992.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 169–178. ACM Press, May / June 2009.

[GHAHJ22] Aarushi Goel, Mathias Hall-Andersen, Aditya Hegde, and Abhishek Jain. Secure multiparty computation with free branching. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part I*, volume 13275 of *LNCS*, pages 397–426. Springer, Heidelberg, May / June 2022.

[GI14] Niv Gilboa and Yuval Ishai. Distributed point functions and their applications. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 640–658. Springer, Heidelberg, May 2014.

[Gil99] Niv Gilboa. Two party RSA key generation. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 116–129. Springer, Heidelberg, August 1999.

[GJ11] Anna Gál and Jing-Tang Jang. The size and depth of layered boolean circuits. *Information Processing Letters*, 111(5):213–217, 2011.

[GM82] Shafi Goldwasser and Silvio Micali. Probabilistic encryption and how to play mental poker keeping secret all partial information. In *14th ACM STOC*, pages 365–377. ACM Press, May 1982.

[GMW87a] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987.

[GMW87b] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to prove all NP-statements in zero-knowledge, and a methodology of cryptographic protocol design. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 171–185. Springer, Heidelberg, August 1987.

[Gol00] Oded Goldreich. Candidate one-way functions based on expander graphs. Cryptology ePrint Archive, Report 2000/063, 2000. `https://eprint.iacr.org/2000/063`.

[Har77] Lawrence H. Harper. An log lower bound on synchronous combinational complexity. 1977.

[HK20] David Heath and Vladimir Kolesnikov. Stacked garbling - garbled circuit proportional to longest execution path. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 763–792. Springer, Heidelberg, August 2020.

[HK21] David Heath and Vladimir Kolesnikov. LogStack: Stacked garbling with $O(b \log b)$ computation. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part III*, volume 12698 of *LNCS*, pages 3–32. Springer, Heidelberg, October 2021.

[HKP20] David Heath, Vladimir Kolesnikov, and Stanislav Peceny. MOTIF: (almost) free branching in GMW - via vector-scalar multiplication. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part III*, volume 12493 of *LNCS*, pages 3–30. Springer, Heidelberg, December 2020.

[IK04] Yuval Ishai and Eyal Kushilevitz. On the hardness of information-theoretic multiparty computation. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 439–455. Springer, Heidelberg, May 2004.

[IKM+13] Yuval Ishai, Eyal Kushilevitz, Sigurd Meldgaard, Claudio Orlandi, and Anat Paskin-Cherniavsky. On the power of correlated randomness in secure computation. In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 600–620. Springer, Heidelberg, March 2013.

[IKNP03] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 145–161. Springer, Heidelberg, August 2003.

[IKOS08] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Cryptography with constant computational overhead. In Richard E. Ladner and Cynthia Dwork, editors, *40th ACM STOC*, pages 433–442. ACM Press, May 2008.

[IP07] Yuval Ishai and Anat Paskin. Evaluating branching programs on encrypted data. In Salil P. Vadhan, editor, *TCC 2007*, volume 4392 of *LNCS*, pages 575–594. Springer, Heidelberg, February 2007.

[IPS08] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer - efficiently. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 572–591. Springer, Heidelberg, August 2008.

[JKPT12] Abhishek Jain, Stephan Krenn, Krzysztof Pietrzak, and Aris Tentes. Commitments and efficient zero-knowledge proofs from learning parity with noise. In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 663–680. Springer, Heidelberg, December 2012.

[JLS21] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 60–73, 2021.

[JLS22] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from LPN over $\mathbb{F}_p$, DLIN, and PRGs in NC^0. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part I*, volume 13275 of *LNCS*, pages 670–699. Springer, Heidelberg, May / June 2022.

[Kil91] Joe Kilian. A general completeness theorem for two-party games. In *23rd ACM STOC*, pages 553–560. ACM Press, May 1991.

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *24th ACM STOC*, pages 723–732. ACM Press, May 1992.

[Kil00] Joe Kilian. More general completeness theorems for secure two-party computation. In *32nd ACM STOC*, pages 316–324. ACM Press, May 2000.

[KO97] Eyal Kushilevitz and Rafail Ostrovsky. Replication is NOT needed: SINGLE database, computationally-private information retrieval. In *38th FOCS*, pages 364–373. IEEE Computer Society Press, October 1997.

[KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 669–684. ACM Press, November 2013.

[Kus89] Eyal Kushilevitz. Privacy and communication complexity. In *30th FOCS*, pages 416–421. IEEE Computer Society Press, October / November 1989.

[Lin16] Yehuda Lindell. How to simulate it - A tutorial on the simulation proof technique. Cryptology ePrint Archive, Report 2016/046, 2016. `https://eprint.iacr.org/2016/046`.

[Lyu05] Vadim Lyubashevsky. The parity problem in the presence of noise, decoding random linear codes, and the subset sum problem. In Chandra Chekuri, Klaus Jansen, José D. P. Rolim, and Luca Trevisan, editors, *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques*, pages 378–389, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[MST03] Elchanan Mossel, Amir Shpilka, and Luca Trevisan. On e-biased generators in NC0. In *44th FOCS*, pages 136–145. IEEE Computer Society Press, October 2003.

[NN01] Moni Naor and Kobbi Nissim. Communication preserving protocols for secure function evaluation. In *33rd ACM STOC*, pages 590–599. ACM Press, July 2001.

[NP01] Moni Naor and Benny Pinkas. Efficient oblivious transfer protocols. In S. Rao Kosaraju, editor, *12th SODA*, pages 448–457. ACM-SIAM, January 2001.

[NRR00] Moni Naor, Omer Reingold, and Alon Rosen. Pseudo-random functions and factoring (extended abstract). In *32nd ACM STOC*, pages 11–20. ACM Press, May 2000.

[OSY21] Claudio Orlandi, Peter Scholl, and Sophia Yakoubov. The rise of paillier: Homomorphic secret sharing and public-key silent OT. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 678–708. Springer, Heidelberg, October 2021.

[OW14] Ryan ODonnell and David Witmer. Goldreich's prg: evidence for near-optimal polynomial stretch. In *Computational Complexity (CCC), 2014 IEEE 29th Conference on*, pages 1–12. IEEE, 2014.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 223–238. Springer, Heidelberg, May 1999.

[Pra62] Eugene Prange. The use of information sets in decoding cyclic codes. *IRE Transactions on Information Theory*, 8(5):5–9, 1962.

[RAD78] Ronald L Rivest, Len Adleman, and Michael L Dertouzos. On data banks and privacy homomorphisms. volume 4, pages 169–180. Citeseer, 1978.

[RB89] Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In *21st ACM STOC*, pages 73–85. ACM Press, May 1989.

[RBO89] T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing*, STOC '89, page 73–85, New York, NY, USA, 1989. Association for Computing Machinery.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th ACM STOC*, pages 84–93. ACM Press, May 2005.

[RS21] Lawrence Roy and Jaspal Singh. Large message homomorphic secret sharing from DCR and applications. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part III*, volume 12827 of *LNCS*, pages 687–717, Virtual Event, August 2021. Springer, Heidelberg.

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd FOCS*, pages 160–164. IEEE Computer Society Press, November 1982.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986.

[YZW+19] Yu Yu, Jiang Zhang, Jian Weng, Chun Guo, and Xiangxue Li. Collision resistant hashing from sub-exponential learning parity with noise. In Steven D. Galbraith and Shiho Moriai, editors, *ASIACRYPT 2019, Part II*, volume 11922 of *LNCS*, pages 3–24. Springer, Heidelberg, December 2019.