Chapter 1

Function Secret Sharing and Homomorphic Secret Sharing

A secret sharing scheme enables a dealer holding a secret s to split s into m shares, such that certain subsets of the shares can be used to reconstruct the secret, while others reveal nothing about it. Secret sharing is a fundamental cryptographic primitive whose applications span almost all sub-areas of the field. A simple and useful secret sharing scheme, called *additive secret sharing*, splits a secret s from a finite Abelian group $\mathbb{G}$ into m shares $s_1, \ldots, s_m \in \mathbb{G}$ that are picked uniformly at random subject to the constraint that they add up to s. Equivalently, one can pick the shares $s_1, \ldots, s_{m-1}$ uniformly at random from $\mathbb{G}$ and let $s_m \leftarrow s - \sum_{i=1}^{m-1} s_i$. The secret s can be reconstructed by adding all m shares, while every strict subset of the shares reveal nothing about s. A useful extra feature of additive secret sharing is that it is *additively homomorphic*: if we distribute shares of $k \geq 2$ secrets between m parties, the parties can *locally* add their shares of the k secrets and obtain a valid additive sharing of the sum of the secrets.

Function Secret Sharing (FSS) and Homomorphic Secret Sharing (HSS) are two extensions of standard secret sharing that enable richer forms of homomorphic computations on secrets. We describe both notions informally below.

Function Secret Sharing. An m-party FSS scheme splits a *function* $f : \{0,1\}^n \to \mathbb{G}$ from a function class $\mathcal{F}$ into m additive shares $f_i : \{0,1\}^n \to \mathbb{G}$, each represented by a key k_i, where every strict subset of the keys k_i hides f. The correctness requirement is that the m functions f_i represented by the keys add up to f. Namely, there is a function Eval (defining the function f_i represented by k_i), such that for every $x \in \{0,1\}^n$ we have $f(x) = \sum_i \mathsf{Eval}(k_i, x)$. A trivial solution is to use additive secret sharing to independently share each entry in the truth-table of f. However,

this would lead to keys k_i consisting of 2^n group elements. The challenge is to design efficient FSS schemes in which the key size grows polynomially with the input length n. The short keys k_i can be viewed as *compressed* additive shares of the truth-table of f. This is only possible for classes $\mathcal{F}$ of *structured* functions f that have a short (polynomial-size) description, and inevitably requires one to settle for *computational* hiding of f.

Homomorphic Secret Sharing. HSS extends the above example of additive secret sharing by supporting the evaluation of other (possibly nonlinear) functions on the shares. It can be viewed as a dual notion of FSS where the roles of the function and input are reversed. Concretely, an HSS scheme splits an *input* $x \in \{0,1\}^n$ into m shares x_i such that each strict subset of the shares hides the input; moreover, one can locally evaluate a function $f \in \mathcal{F}$ on the input shares x_i such that the output shares add up to $f(x)$. That is, there exists a local evaluation function Eval such that for every $f \in \mathcal{F}$ and shares x_i of x we have $f(x) = \sum_i \mathsf{Eval}(x_i, f)$.

In recent years, a variety of FSS and HSS schemes with different efficiency and security features have been discovered and used for a wide array of applications. In this chapter, we give an overview of these constructions and applications.

Organization. We start, in Section 1.1, by formally defining the notions of FSS, HSS, and the related notion of pseudorandom correlation generators. In Section 1.2 we give an overview of some of the main constructions of FSS and HSS schemes, classifying them as low-end constructions (concretely efficient constructions from weak assumptions for restricted function classes), mid-range constructions, and high-end constructions (from stronger assumptions and for general functions). We conclude in Section 1.3 with a non-comprehensive list of applications of FSS and HSS as of the writing of this chapter (2021).

1.1 Definitions and Discussion

In this section, we introduce three main definitions: Function Secret Sharing (FSS), Homomorphic Secret Sharing (HSS), and Pseudorandom Correlation Generators (PCG), a related primitive that captures some of the applications of FSS and HSS.

Defining FSS and HSS. Similarly to standard secret sharing schemes, FSS and HSS must satisfy two kinds of requirements: (1) Correctness, captured by a homomorphic evaluation guarantee, and (2) Privacy, requiring that subsets of shares do not reveal the secret. When formalizing these notions, there are several definitional

choices to be made that offer different tradeoffs between simplicity and generality. We present two kinds of definitions that target different applications:

1. An FSS definition which is most suitable to simple function classes that arise in practical applications;

2. A more general HSS definition which is particularly convenient for a theory-oriented study and naturally supports a multi-input variant.

We start with an informal overview of these two definitions, and defer the formal version to the end of this section.

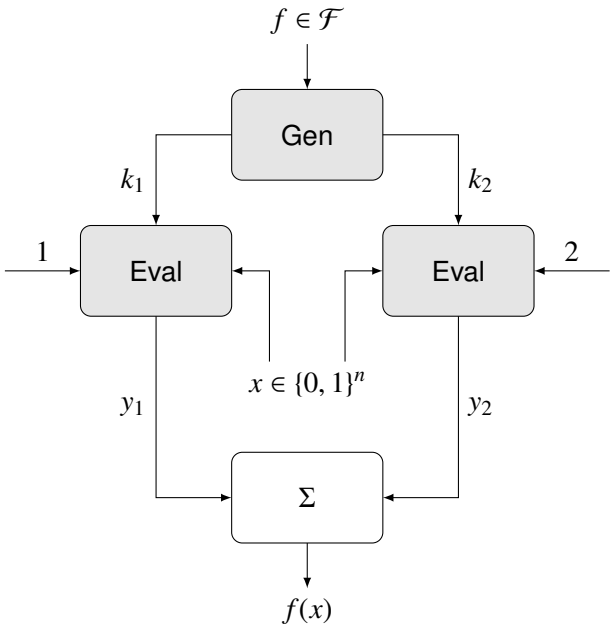

Figure 1.1.1: Representation of a 2-party function secret sharing scheme for a function class $\mathcal{F} = \{f : \{0,1\}^n \to \mathbb{G}\}$. Σ denotes the sum over $\mathbb{G}$.

- **Function Secret Sharing (FSS) [20].** Suppose we are given a class $\mathcal{F}$ of efficiently computable and succinctly described functions $f : \{0,1\}^n \to \mathbb{G}$. Is it possible to split an arbitrary *function* $f \in \mathcal{F}$ into m functions $f_1, \ldots, f_m$ such that: (1) each f_i is described by a short key k_i that enables its efficient evaluation, (2) strict subsets of the keys completely hide f, and (3) $f(x) = \sum_{i=1}^{m} f_i(x)$ (on every input x)? We refer to a solution to this problem as a *function secret sharing* (FSS) scheme for $\mathcal{F}$. We provide on Figure 1.1.1 a pictural representation of the algorithms of a 2-party FSS scheme.

- **Homomorphic Secret Sharing (HSS) [21].** A (m-party) HSS scheme for class of programs[1] $\mathcal{P}$ randomly splits an input x into shares[2] $(x^1, \ldots, x^m)$ such that: (1) each x^i is polynomially larger than x, (2) subsets of shares x^i hide x, and (3) there exists a polynomial-time local evaluation algorithm Eval such that for any "program" $P \in \mathcal{P}$ (e.g., a boolean circuit, formula or branching program), the output $P(x)$ can be efficiently reconstructed from $\mathsf{Eval}(x^1, P), \ldots, \mathsf{Eval}(x^m, P)$. By default, we require the reconstruction to be *additive* over a finite Abelian group $\mathbb{G}$.

On the difference between FSS and HSS. While any FSS scheme can be viewed as an HSS scheme for a suitable class of programs and vice versa, switching the roles of the program and input is often unnatural. We will choose the notion that gives rise to the simpler formulation of each construction or application. When considering universal classes of programs, such as Boolean circuits, a technical difference between the two notions is that FSS allows the share size to grow with the size of the program whereas HSS restricts the share size to grow only with the size of the *input* to the program. Finally, HSS admits a natural multi-input variant (where homomorphic evaluation can apply jointly to shared secrets originating from two or more parties), whereas in FSS the secret function always originates from a single source.

Pseudorandom correlation generators. In this section, we also introduce the notion of a *pseudorandom correlation generator* (PCG), which turns out to be closely related to FSS/HSS. Informally, a PCG is a pair of algorithms denoted $(\mathsf{Gen}, \mathsf{Expand})$, where Gen outputs a pair of short, correlated seeds, and Expand allows to locally stretch these seeds into long, correlated pseudorandom strings. The PCG securely realizes a given target correlation (e.g., n instances of random oblivious transfer) if the expanded seeds are indistinguishable from the target correlation even to insiders who obtain one of the two seeds. While the notion of PCG is not visibly similar to FSS/HSS, there is in fact a two-way relation between the notions: HSS schemes can be used to construct (certain) useful PCGs and vice versa.

Efficient PCG constructions build heavily both on the succinctness and non-interactivity features of FSS/HSS, and enable practical compression of correlated

[1] Function vs. program: Note that in FSS we will consider simple classes of functions where each function has a unique description, whereas in HSS we consider functions with many programs computing it. For this reason we refer to "function" for FSS and "program" for HSS.

[2] f_i vs. x^i: We maintain the subscript/superscript conventions of existing works (primarily [20, 25]). Note that superscript notation is used in HSS where one can consider shares of multiple inputs, $x_j \mapsto (x_j^1, \ldots, x_j^m)$.

randomness that can be used to speed up protocols for secure computation.

1.1.1 Basic Notation

We denote the security parameter by λ.

Modeling function families. A *function family* is defined by a pair $\mathcal{F} = (P_{\mathcal{F}}, E_{\mathcal{F}})$, where $P_{\mathcal{F}} \subseteq \{0,1\}^*$ is an infinite collection of function descriptions $\hat{f}$, and $E_{\mathcal{F}} : P_{\mathcal{F}} \times \{0,1\}^* \to \{0,1\}^*$ is a polynomial-time algorithm defining the function described by $\hat{f}$. Concretely, each $\hat{f} \in P_{\mathcal{F}}$ describes a corresponding function $f : D_f \to R_f$ defined by $f(x) = E_{\mathcal{F}}(\hat{f}, x)$. We assume by default that $D_f = \{0,1\}^n$ for a positive integer n (though will sometimes consider inputs over non-binary alphabets) and always require R_f to be a finite Abelian group, denoted by $\mathbb{G}$. When there is no risk of confusion, we will sometimes write f instead of $\hat{f}$ and $f \in \mathcal{F}$ instead of $\hat{f} \in P_{\mathcal{F}}$. We assume that $\hat{f}$ includes an explicit description of both D_f and R_f as well as a size parameter $S_{\hat{f}}$.

1.1.2 Function Secret Sharing: Targeting Applications

We next present a targeted definition of FSS, which lies most in line with the use of FSS within current practical applications. The definition follows [22], extending the original definition from [20] by allowing a general specification of allowable *leakage*: i.e., partial information about the function that can be revealed.

Recall in the language of FSS, we consider a client holding a secret function $f \in \mathcal{F}$ who splits f into shares f_i supporting homomorphic evaluation on inputs x in the domain of f. We use notation of the shares f_i described by keys k_i.

Modeling leakage. We use a function $\mathsf{Leak} : \{0,1\}^* \to \{0,1\}^*$ to capture the allowable leakage, where $\mathsf{Leak}(f)$ is interpreted as the partial information about f that can be leaked. When Leak is omitted it is understood to output the input domain D_f and the output domain R_f. This will be sufficient for most classes considered; for some classes, one also needs to leak the size S_f. But, one can consider more general choices of Leak, which allow a tradeoff between efficiency/feasibility and revealed information. (E.g., the construction of FSS for decision trees in [22] leaks the topology of the tree but hides the labels; see Section 1.2.)

Definition 1.1.1 (FSS: Syntax)**.** An m-party *function secret sharing (FSS) scheme* is a pair of algorithms $(\mathsf{Gen}, \mathsf{Eval})$ with the following syntax:

- $\mathsf{Gen}(1^\lambda, \hat{f})$ is a PPT *key generation* algorithm, which on input 1^λ (security parameter) and $\hat{f} \in \{0,1\}^*$ (description of a function f) outputs an m-tuple of keys $(k_1, \ldots, k_m)$. We assume that $\hat{f}$ explicitly contains an input length 1^n, group description $\mathbb{G}$, and size parameter.

- $\mathsf{Eval}(i, k_i, x)$ is a polynomial-time *evaluation algorithm*, which on input $i \in [m]$ (party index), k_i (key defining $f_i : \{0,1\}^n \to \mathbb{G}$) and $x \in \{0,1\}^n$ (input for f_i) outputs a group element $y_i \in \mathbb{G}$ (the value of $f_i(x)$, the i-th share of $f(x)$).

When m is omitted, it is understood to be 2.

Definition 1.1.2 (FSS: Requirements)**.** Let $\mathcal{F} = (P_\mathcal{F}, E_\mathcal{F})$ be a function family and $\mathsf{Leak} : \{0,1\}^* \to \{0,1\}^*$ be a function specifying the allowable leakage. Let m (number of parties) and t (secrecy threshold) be positive integers. An *m-party t-secure FSS for $\mathcal{F}$ with leakage* Leak is a pair $(\mathsf{Gen}, \mathsf{Eval})$ as in Definition 1.1.1, satisfying the following requirements.

- **Correctness:** For all $\hat{f} \in P_\mathcal{F}$ describing $f : \{0,1\}^n \to \mathbb{G}$, and every $x \in \{0,1\}^n$, if $(k_1, \ldots, k_m) \leftarrow \mathsf{Gen}(1^\lambda, \hat{f})$ then $\Pr\left[\sum_{i=1}^m \mathsf{Eval}(i, k_i, x) = f(x)\right] = 1$.
- **Secrecy:** For every set of corrupted parties $S \subset [m]$ of size t, there exists a PPT algorithm Sim (simulator), such that for every sequence $\hat{f}_1, \hat{f}_2, \ldots$ of polynomial-size function descriptions from $P_\mathcal{F}$, the outputs of the following experiments Real and Ideal are computationally indistinguishable:
  - $\mathsf{Real}(1^\lambda)$: $(k_1, \ldots, k_m) \leftarrow \mathsf{Gen}(1^\lambda, \hat{f}_\lambda)$; Output $(k_i)_{i \in S}$.
  - $\mathsf{Ideal}(1^\lambda)$: Output $\mathsf{Sim}(1^\lambda, \mathsf{Leak}(\hat{f}_\lambda))$.

When Leak is omitted, it is understood to be the function $\mathsf{Leak}(\hat{f}) = (1^n, S_{\hat{f}}, \mathbb{G})$ where 1^n, $S_{\hat{f}}$, and $\mathbb{G}$ are the input length, size, and group description contained in $\hat{f}$. When t is omitted it is understood to be $m - 1$.

A useful instance of FSS, introduced by Gilboa and Ishai [57], is a *distributed point function* (DPF). A DPF can be viewed as a 2-party FSS for the function class $\mathcal{F}$ consisting of all point functions, namely all functions $f : \{0,1\}^n \to \mathbb{G}$ that evaluate to 0 on all but at most one input.

Definition 1.1.3 (Distributed Point Function)**.** A *point function* $f_{\alpha,\beta}$, for $\alpha \in \{0,1\}^n$ and $\beta \in \mathbb{G}$, is defined to be the function $f : \{0,1\}^n \to \mathbb{G}$ such that $f(\alpha) = \beta$ and $f(x) = 0$ for $x \neq \alpha$. We will sometimes refer to a point function with $|\beta| = 1$ (resp., $|\beta| > 1$) as a *single-bit* (resp., *multi-bit*) point function. A *Distributed Point Function* (DPF) is an FSS for the family of all point functions, with the leakage $\mathsf{Leak}(\hat{f}) = (1^n, \mathbb{G})$.

A concrete security variant. For the purpose of describing and analyzing some FSS constructions, it is sometimes convenient (e.g., in [22]) to consider a *finite* family $\mathcal{F}$ of functions $f : D_f \to R_f$ sharing the same (fixed) input domain and

output domain, as well as a fixed value of the security parameter λ. We say that such a finite FSS scheme is (T, ϵ)-*secure* if the computational indistinguishability requirement in Definition 1.1.2 is replaced by (T, ϵ)-indistinguishability, namely any size-T circuit has at most an ϵ advantage in distinguishing between Real and Ideal. When considering an infinite collection of such finite $\mathcal{F}$, parameterized by the input length n and security parameter λ, we require that Eval and Sim be each implemented by a (uniform) PPT algorithm, which is given 1^n and 1^λ as inputs.

1.1.3 Homomorphic Secret Sharing: A General Definition

Recall that HSS is a dual form of FSS. We now consider more general multi-input HSS schemes that support a compact evaluation of a function F on shares of inputs $x_1, \ldots, x_n$ that originate from different clients. More concretely, each client i randomly splits its input x_i between m servers using the algorithm Share, so that x_i is hidden from any t colluding servers (we assume $t = m - 1$ by default). Each server j applies a local evaluation algorithm Eval to its share of the n inputs, and obtains an output share y^j. The output $F(x_1, \ldots, x_n)$ is reconstructed by applying a decoding algorithm Dec to the output shares $(y^1, \ldots, y^m)$. To avoid triviality, we consider various restrictions on Dec that force it to be "simpler" than direct computation of F.

Finally, for some applications it is useful to let F and Eval take an additional input x_0 that is known to all servers. This is necessary for a meaningful notion of single-input HSS (with $n = 1$) [21], and function secret sharing [20, 22]. Typically, the extra input x_0 will be a description of a function f applied to the input of a single client, e.g., a description of a circuit, branching program, or low-degree polynomial. For the case of FSS, the (single) client's input is a description of a program and the additional input x_0 corresponds to a domain element.

We now give our formal definition of general HSS. We give a definition in the plain model; this definition can be extended in a natural fashion to settings with various forms of setup (e.g., common public randomness or a public-key infrastructure, as considered in [23]). We follow the exposition of [25]. Recall subscripts denote input (client) id and superscripts denote share (server) id.

Definition 1.1.4 (HSS)**.** An n-client, m-server, t-secure homomorphic secret sharing scheme for a function $F : (\{0,1\}^*)^{n+1} \to \{0,1\}^*$, or (n, m, t)-*HSS* for short, is a triple of PPT algorithms (Share, Eval, Dec) with the following syntax:

- Share$(1^\lambda, i, x)$: On input 1^λ (security parameter), $i \in [n]$ (client index), and $x \in \{0,1\}^*$ (client input), the sharing algorithm Share outputs m input shares, $(x^1, \ldots, x^m)$.

- $\mathsf{Eval}(j, x_0, (x_1^j, \ldots, x_n^j))$: On input $j \in [m]$ (server index), $x_0 \in \{0,1\}^*$ (common server input), and $x_1^j, \ldots, x_n^j$ (jth share of each client input), the evaluation algorithm Eval outputs $y^j \in \{0,1\}^*$, corresponding to server j's share of $F(x_0; x_1, \ldots, x_n)$.

- $\mathsf{Dec}(y^1, \ldots, y^m)$: On input $(y^1, \ldots, y^m)$ (list of output shares), the decoding algorithm Dec computes a final output $y \in \{0,1\}^*$.

The algorithms $(\mathsf{Share}, \mathsf{Eval}, \mathsf{Dec})$ should satisfy the following correctness and security requirements:

- **Correctness:** For any $n+1$ inputs $x_0, \ldots, x_n \in \{0,1\}^*$,

$$\Pr\left[\begin{array}{c|c} \forall i \in [n]\ (x_i^1, \ldots, x_i^m) \leftarrow \mathsf{Share}(1^\lambda, i, x_i) & \mathsf{Dec}(y^1, \ldots, y^m) \\ \forall j \in [m]\ y^j \leftarrow \mathsf{Eval}(j, x_0, (x_1^j, \ldots, x_n^j)) & = F(x_0; x_1, \ldots, x_n) \end{array}\right] = 1.$$

  Alternatively, in a *statistically correct* HSS the above probability is at least $1 - \mu(\lambda)$ for some negligible μ and in a δ-correct HSS (or δ-HSS for short) it is at least $1 - \delta - \mu(\lambda)$, where the error parameter δ is given as an additional input to Eval and the running time of Eval is allowed to grow polynomially with $1/\delta$.

- **Security:** Consider the following semantic security challenge experiment for corrupted set of servers $T \subset [m]$:
  1: The adversary gives challenge index and inputs $(i, x, x') \leftarrow \mathcal{A}(1^\lambda)$, with $|x| = |x'|$.
  2: The challenger samples $b \leftarrow \{0,1\}$ and $(x^1, \ldots, x^m) \leftarrow \mathsf{Share}(1^\lambda, i, \tilde{x})$, where $\tilde{x} = \begin{cases} x \text{ if } b = 0 \\ x' \text{ else} \end{cases}$.
  3: The adversary outputs a guess $b' \leftarrow \mathcal{A}((x^j)_{j \in T})$, given the shares for corrupted T.

  Denote by $\mathsf{Adv}(1^\lambda, \mathcal{A}, T) := \Pr[b = b'] - 1/2$ the advantage of $\mathcal{A}$ in guessing b in the above experiment, where probability is taken over the randomness of the challenger and of $\mathcal{A}$.

  For circuit size bound $S = S(\lambda)$ and advantage bound $\alpha = \alpha(\lambda)$, we say that an (n, m, t)-HSS scheme $\Pi = (\mathsf{Share}, \mathsf{Eval}, \mathsf{Dec})$ is (S, α)-secure if for all $T \subset [m]$ of size $|T| \le t$, and all non-uniform adversaries $\mathcal{A}$ of size $S(\lambda)$, we have $\mathsf{Adv}(1^\lambda, \mathcal{A}, T) \le \alpha(\lambda)$. We say that Π is:

  - *computationally secure* if it is $(S, 1/S)$-secure for all polynomials S;

- *statistically α-secure* if it is (S, α)-secure for all S;
- *statistically secure* if it statistically α-secure for some negligible $\alpha(\lambda)$;
- *perfectly secure* if it is statistically 0-secure.

Remark 1.1.5 (Unbounded HSS)**.** Definition 1.1.4 treats the number of inputs n as being fixed. We can naturally consider an unbounded multi-input variant of HSS where F is defined over arbitrary sequences of inputs x_i, and the correctness requirement is extended accordingly. We denote this flavor of multi-input HSS by $(*, m, t)$-HSS. More generally, one can allow all three parameters n, m, t to be flexible, treating them as inputs of the three algorithms Share, Eval, Dec.

Remark 1.1.6 (Comparing to FSS Definition)**.** Function secret sharing (FSS) as per Definition 1.1.2 can be cast in the definition above as $(1, m)$-HSS for the universal function $F(x; P) = P(x)$, where $P \in \mathcal{P}$ is a program given as input to the client and x is the common server input.

Note the security requirement for HSS in Definition 1.1.4 is expressed as an indistinguishability guarantee, whereas the FSS definition from the previous section (Definition 1.1.2) referred instead to efficient simulation given leakage on the secret data. However, the two flavors are equivalent for every function family $\mathcal{F}$ and leakage function Leak for which Leak can be efficiently inverted; that is, given $\mathsf{Leak}(\hat{f})$ one can efficiently find $\hat{f}'$ such that $\mathsf{Leak}(\hat{f}') = \mathsf{Leak}(\hat{f})$. Such an inversion algorithm exists for all instances of $\mathcal{F}$ and Leak considered in existing works.

A careful reader might observe at this stage that Definition 1.1.4 can be trivially realized by setting Eval to be the identity function (in which case Dec reconstructs $(x_1, \cdots, x_n)$ and outputs $F(x_0; x_1 \cdots x_n)$). To make HSS useful, we must impose requirements on the decoding algorithm.

Definition 1.1.7 (Additive and compact HSS)**.** We say that an (n, m, t)-HSS scheme $\Pi = (\mathsf{Share}, \mathsf{Eval}, \mathsf{Dec})$ is:

- *Additive* if Dec outputs the exclusive-or of the m output shares. Alternatively, if Dec interprets its m arguments as elements of an Abelian group $\mathbb{G}$ (instead of bit strings), and outputs their sum in $\mathbb{G}$.[3] Note that this requirement is similar to the correctness property for our definition of FSS (see Definition 1.1.2).

- *Compact* if the length of the output shares is sublinear in the input length when the inputs are sufficiently longer than the security parameter. Concretely:

[3] In this case, we think of the function F and all HSS algorithms Share, Eval, Dec as implicitly receiving a description of $\mathbb{G}$ as an additional input.

- We say that Π is $g(\lambda, \ell)$-*compact* if for every λ, ℓ, and every inputs $x_0, x_1, \ldots, x_n \in \{0,1\}^\ell$, the length of each output share obtained by applying Share with security parameter λ and then Eval is at most $g(\lambda, \ell)$.
- We say that Π is *compact* if it is $g(\lambda, \ell)$-compact for g that satisfies the following requirement: There exists a polynomial $p(\cdot)$ and sublinear function $g'(\ell) = o(\ell)$ such that for any λ and $\ell \geq p(\lambda)$ we have $g(\lambda, \ell) \leq g'(\ell)$.

In the case of perfect security or statistical α-security with constant α, we eliminate the parameter λ and refer to Π as being $g(\ell)$-compact.

Remark 1.1.8 (Other notions of compactness)**.** One could alternatively consider a stronger notion of compactness, requiring that the length of each output share is of the order of the output length (whereas Definition 1.1.7 requires merely for it to be sublinear in the input size). Every additive HSS scheme satisfies this notion. HSS schemes that satisfy this notion but are not additive were used in the context of private information retrieval and locally decodable codes in [7]. A different way of strengthening the compactness requirement is by restricting the *computational complexity* of Dec, e.g., by requiring it to be quasi-linear in the length of the output. See Section 1.3.1 (worst-case to average-case reductions) for motiving applications.

Remark 1.1.9 (Special HSS Cases)**.**

- We will sometimes be interested in additive (multi-input) HSS for a *finite function* F, such as the AND of two bits; this can be cast into Definition 1.1.4 by just considering an extension $\hat{F}$ of F that outputs 0 on all invalid inputs. (Note that our notion of compactness is not meaningful for a finite F.)
- As noted above, the common server input x_0 is often interpreted as a "program" P from a class of programs $\mathcal{P}$ (e.g., circuits or branching programs), and F is the universal function defined by $F(P; x_1, \ldots, x_n) = P(x_1, \ldots, x_n)$. We refer to this type of HSS as *HSS for the class* $\mathcal{P}$.

1.1.4 On the Output Decoding Structure of FSS/HSS

In the definitions above, we focused on FSS with additive reconstruction (that is, $f(x) = \sum_i \mathsf{Eval}(f_i, x)$ over an abelian group $\mathbb{G}$), and on HSS with either additive reconstruction, or general reconstruction but compact shares. In this section, we discuss and motivate these requirements.

Of course, one can consider FSS/HSS with respect to many choices of output decoding structure. Based on the structure of the chosen decoding process, the corresponding scheme will have very different properties: more complex decoding procedures open the possibility of achieving FSS/HSS for more general classes of functions, but place limits on the applicability of the resulting scheme. Many choices for the structure of the output decoding function yield uninteresting notions, as we now discuss (following [20]). For convenience, we adopt the language of FSS, but similar considerations hold for HSS.

Arbitrary reconstruction. Consider, for example, FSS with *no restriction* on the reconstruction procedure for parties' output shares. Such wide freedom renders the notion non-meaningfully trivial. Indeed, for any efficient function family $\mathcal{F}$, one could generate FSS keys for a secret function $f \in \mathcal{F}$ simply by sharing a description of f *interpreted as a string*, using a standard secret sharing scheme. The evaluation procedure on any input x will simply output x together with the party's share of f, and the decoding procedure will first reconstruct the description of f, and then compute and output the value $f(x)$.

This construction satisfies correctness and security as described informally above (indeed, each party's key individually reveals no information on f). But, the scheme clearly leaves much to be desired in terms of utility: From just one evaluation, the entire function f is revealed to whichever party receives and reconstructs these output shares. At such point, the whole notion of function secret sharing becomes moot.

"Function-private" output shares. Instead, from a function secret sharing scheme, one would hope that parties' output shares $f_i(x)$ for input x do not reveal more about the secret function f than is necessary to determine $f(x)$. That is, we may impose a "function privacy" requirement on the reconstruction scheme, requiring that pairs of parties' output shares for each input x can be simulated given just the corresponding outputs $f(x)$.

This requirement is both natural and beneficial, but by itself still allows for undesired constructions. For example, given a secret function f, take one FSS key to be a *garbled circuit* of f, and the second key as the information that enables translating inputs x to garbled input labels. This provides a straightforward function-private solution for one output evaluation, and can easily be extended to the many-output case by adding shared secret randomness to the parties' keys.[4] Yet this construction (and thus definition) is unsatisfying: although the evaluate output shares $f_i(x)$ now hide f, their size is massive—for every output, comparable to a copy of f itself. (Further, this notion does not give any cryptographic power

[4]Namely, for each new x, the parties will first use their shared randomness to coordinately rerandomize the garbled circuit of f and input labels, respectively.

beyond garbled circuits.)

Succinct, function-private output shares. We further restrict the scheme, demanding additionally that output shares be *succinct*: i.e., comparable in size to the function output.

This definition already captures a strong, interesting primitive. For example, as described in Section 1.3, achieving such an FSS scheme for general functions implies a form of communication-efficient secure multi-party computation. Additional lower bounds on this notion are shown in [25]. However, there is one final property that enables an important class of applications, but which is not yet guaranteed: a notion of *share compressibility*.

More specifically: One of the central application regimes of FSS [57, 20, 22] is enabling communication-efficient secure (m-server) Private Information Retrieval (PIR). Intuitively, to privately recover an item x_i from a database held by both servers, one can generate and distribute a pair of FSS keys encoding a point function f_i whose only nonzero output is at secret location i. Each server then responds with a *single* element, computed as the weighted sum of each data item x_j with the server's output share of the evaluation $f_i(x_j)$. Correctness of the DPF scheme implies that the xor of the two servers' replies is precisely the desired data item x_i, while security guarantees the servers learn nothing about the index i. But most importantly, the linear structure of the DPF reconstruction enabled the output shares pertaining to all the different elements of the database to be *compressed* into a single short response.

On the other hand, consider, for example, the PIR scenario but where the servers instead hold shares of the function f_i with respect to a *bitwise AND* reconstruction of output shares in the place of xor/addition. Recovery of the requested data item x_i now implies computing set intersection—and thus requires communication complexity equal to the size of the database [66]! We thus maintain the crucial property that output shares can be combined and compressed in a meaningful way. To do so, we remain in stride with the *linearity* of output share decoding.

Primary Focus: Linear share decoding. We focus predominantly on the setting of FSS where the output decoder is a *linear function* of parties' shares. That is, we assume the output shares $f_i(x)$ lie within an Abelian group $\mathbb{G}$ and consider a decoding function $\mathsf{Dec} : \mathbb{G}^m \to \mathbb{G}$ linear in $\mathbb{G}$. This clean, intuitive structure in fact provides the desired properties discussed above: Linearity of reconstruction provides convenient share *compressibility*. Output shares must themselves be elements of the function output space, immediately guaranteeing share *succinctness*. And as shown in [20], the linear reconstruction in conjunction with basic key security directly implies *function privacy*. Unless otherwise specified we will implicitly

take an "FSS scheme" (or HSS) to be one with a linear reconstruction procedure.

1.1.5 Pseudorandom Correlation Generators

In this section we put forward a general notion of pseudorandom correlation generator (PCG). At a high level, a PCG for some target ideal correlation takes as input a pair[5] of short, correlated seeds and outputs long correlated pseudorandom strings, where the expansion procedure is deterministic and can be applied locally.

For correctness we require that the expanded output of a PCG is indistinguishable from truly random correlated strings that are sampled from the ideal correlation. For security it would be natural and straightforward to require that we can securely replace long correlated strings by short correlated seeds in any secure protocol execution. Unfortunately, as shown in [16], this security requirement would be impossible to meet. Therefore, following [16], we introduce (and subsequently prove useful) an indistinguishability-based security notion. Namely, we require that an adversary given access to one of the short seeds k_σ, cannot distinguish the pseudorandom string $R_{1-\sigma}$ from a pseudorandom string that is chosen at random conditioned on (R_0, R_1) being appropriately correlated (where $R_\sigma = \mathsf{PCG}(\mathsf{k}_\sigma)$ is the expansion of the short seed k_σ). In other words, an adversary given access to a short seed cannot learn more about the other party's pseudorandom string than what is obvious given access to its own pseudorandom *output string*.

In order to formally define PCGs, we first introduce the concept of a *correlation generator* as a PPT algorithm outputting correlated strings. We will use a correlation generation generator to define an ideal target correlation $(\mathcal{R}_0, \mathcal{R}_1)$. For simplicity, we assume that $(\mathcal{R}_0, \mathcal{R}_1)$ are two bit-strings of the same length n, though in some of the useful instances instances we will discuss they are more naturally interpreted as vectors over some finite ring.

Definition 1.1.10 (Correlation Generator)**.** A PPT algorithm C is called a *correlation generator*, if C on input 1^λ outputs a pair of strings in $\{0,1\}^n \times \{0,1\}^n$ for $n = n(\lambda) \in \mathsf{poly}(\lambda)$.

Our security definition requires the target correlation to satisfy a technical requirement, which roughly says that it is possible to efficiently sample from the conditional distribution of $\mathcal{R}_0$ given $\mathcal{R}_1 = r_1$ and vice versa.

Definition 1.1.11 (Reverse-sampleable Correlation Generator)**.** Let C be a correlation generator. We say C is *reverse sampleable* if there exists a PPT algorithm

[5] While both the notion and some of the constructions extend to an arbitrary number of parties, here we focus on the 2-party case for simplicity.

RSample such that for $\sigma \in \{0,1\}$ the correlation obtained via:

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

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

The following definition of pseudorandom correlation generators generalizes an earlier definition of pseudorandom VOLE generator in [14].

Definition 1.1.12 (Pseudorandom Correlation Generator (PCG) [16])**.** Let C be a reverse-sampleable correlation generator. A *PCG for 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 $\mathcal{R}_\sigma \in \{0,1\}^n$.

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

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

  $$\{(\mathcal{R}_0, \mathcal{R}_1) \,|(\mathsf{k}_0, \mathsf{k}_1) \xleftarrow{\$} \mathsf{PCG.Gen}(1^\lambda), (R_\sigma \leftarrow \mathsf{PCG.Expand}(\sigma, \mathsf{k}_\sigma))_{\sigma=0,1}\}$$

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

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

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

  where RSample is the reverse sampling algorithm for correlation C.

Note that the above definition is trivial to achieve in general: We can let PCG.Gen on input 1^λ return $(R_0, R_1) \leftarrow C(1^\lambda)$, and simply define Expand to be the identity. Typically, we will be interested in non-trivial constructions of PCGs, in which PCG.Expand stretches a short seed to a long output. As a simple example, a standard pseudorandom generator $G : \{0,1\}^\lambda \to \{0,1\}^{n(\lambda)}$ naturally defines the following PCG for the target correlation $C(1^\lambda) = (r, r)$, where $r \in_R \{0,1\}^{n(\lambda)}$:

PCG.Gen output a pair of identical random seeds and PCG.Expand applies G locally on each seed. In the following we will consider more involved constructions of PCGs for useful target correlations where neither of the outputs determine the other. These include Oblivious Transfer (OT) correlations, Oblivious Linear Evaluation (OLE) correlations, and (authenticated) multiplication triples.

1.1.6 Homomorphic Secret Sharing vs. Pseudorandom Correlation Generators

In the following we show that HSS for a class of functions of the form $\{f \circ \mathsf{PRG}\}$, where PRG is a pseudorandom generator, implies PCG for a correlation related to f. In Section 1.2.4, we will show a partial converse relation: PCGs for *polynomial correlations* imply HSS for polynomials.

The high-level strategy is as follows: We combine a standard pseudorandom generator (PRG), expanding a short random seed into a long pseudorandom string, with a suitable HSS scheme, which allows to *locally* compute the target correlation on shares of a random input. More precisely, we consider the special case of additive correlations, where R_0, R_1 are uniformly distributed subject to $R_0 + R_1 = f(X)$ for a random input X and fixed function f. Now, consider an HSS scheme with additive reconstruction for $f \circ \mathsf{PRG}$. This gives rise to the following PCG construction: During key generation a short seed k is shared between the players (as HSS shares). For expansion, the players can then locally evaluate $f(\mathsf{PRG}(\mathsf{k}))$ via the HSS operations. By the correctness of the HSS that indeed gives outputs R_0, R_1 with $R_0 + R_1 = f(X)$, where $X = \mathsf{PRG}(\mathsf{k})$. In this section we formally prove that the described construction meets the PCG requirements.

To formalize the above outline, we first give a generalized definition of a pseudorandom generator, then formally define additive correlations corresponding to a function, before presenting the construction.

Let $\mathcal{R}$ be a ring and $\ell, n \in \mathbb{N}$. We consider distributions $\mathcal{D}^\ell$ over a ring $\mathcal{R}^\ell$ and write $X \xleftarrow{\$} \mathcal{D}^\ell(\mathcal{R})$ or simply $X \xleftarrow{\$} \mathcal{D}^\ell$ (if $\mathcal{R}$ is clear from the context) to denote sampling from $\mathcal{R}^\ell$ via $\mathcal{D}^\ell$. Note that the following definition of $\mathcal{D}^\ell$-pseudorandom generator coincides with the standard definition of a PRG, if we choose $\mathcal{D}^\ell(\mathcal{R}) = \mathcal{U}^\ell(\mathcal{R})$. We use this more general notion of a PRG, as for our PRG instantiation from LPN the seed is not chosen uniformly at random.

Definition 1.1.13 ($\mathcal{D}^\ell$-Pseudorandom Generator)**.** Let $\mathcal{R}$ be a ring (parametrized implicitly by λ) and let $\mathcal{D}^\ell$ be a distribution on $\mathcal{R}^\ell$. We say $\mathsf{PRG}\colon \mathcal{R}^\ell \to \mathcal{R}^n$ is a *$\mathcal{D}^\ell$-pseudorandom generator (PRG)*, if the following two distributions are compu-

- $\mathsf{PCG.Setup}(1^\lambda)$: output $(\mathsf{sk}, \{\mathsf{ek}_\sigma\}_{\sigma\in\{0,1\}}) \leftarrow \mathsf{HSS.Gen}(1^\lambda)$.
- $\mathsf{PCG.Gen}(\mathsf{sk})$: Sample $r \leftarrow \mathcal{D}^\ell$; output $(\mathsf{k}_0, \mathsf{k}_1) \leftarrow \mathsf{HSS.Share}(\mathsf{sk}, r)$.
- $\mathsf{PCG.Expand}(\sigma, \mathsf{ek}_\sigma, \mathsf{k}_\sigma, f)$: Output $\mathcal{R}_\sigma \leftarrow \mathsf{HSS.Eval}(\sigma, \mathsf{ek}_\sigma, \mathsf{k}_\sigma, f \circ \mathsf{PRG})$.

Figure 1.1.2: PCG for correlation $C_\mathcal{F}$. Here, PRG is a $\mathcal{D}^\ell$-PRG and HSS = (HSS.Gen, HSS.Share, HSS.Eval) an HSS for the family of functions $\mathcal{F}_{\mathsf{HSS}} := \{f \circ \mathsf{PRG} \colon r \mapsto f(\mathsf{PRG}(r)) \mid f \in \mathcal{F}\}$.

tationally indistinguishable:

$$\left\{Y \mid X \xleftarrow{\$} \mathcal{D}^\ell(\mathcal{R}), Y := \mathsf{PRG}(X)\right\} \text{ and } \left\{Y \mid Y \xleftarrow{\$} \mathcal{U}^n(\mathcal{R})\right\}.$$

We will consider *additive correlations* corresponding to a family of functions $\mathcal{F}$. Such a correlation is generated by outputting an additive secret-sharing of a function from $f \in \mathcal{F}$ applied to a source of randomness.

Definition 1.1.14 (Correlation Generators for Additive Correlations)**.** Let $\mathcal{R}$ be a ring. Let $n, m \in \mathbb{N}$ and $\mathcal{F} \subseteq \{f \colon \mathcal{R}^n \to \mathcal{R}^m\}$ be a family of functions. Then we define a *correlation generator $C_\mathcal{F}$ for $\mathcal{F}$* as follows: On input 1^λ and $f \in \mathcal{F}$ the correlation generator $C_\mathcal{F}$ samples $X \xleftarrow{\$} \mathcal{U}^n(\mathcal{R})$, and returns a pair $(\mathcal{R}_0, \mathcal{R}_1) \in \mathcal{R}^m \times \mathcal{R}^m$, which is distributed uniformly at random conditioned on $\mathcal{R}_0 + \mathcal{R}_1 = f(X)$.

Note that $C_\mathcal{F}$ is reverse-sampleable for any family of functions $\mathcal{F}$, as given a function $f \in \mathcal{F}$ and a share $\mathcal{R}_\sigma$, one can draw an input $X \xleftarrow{\$} \mathcal{U}^n(\mathcal{R})$ and set $\mathcal{R}_{1-\sigma} := \mathcal{R}_\sigma - f(X)$. Further, note that it is straightforward to includes shares of the inputs in the correlation by considering the family $\mathcal{F}' := \{f' \colon \mathcal{R}^n \to \mathcal{R}^{n+m}, X \mapsto (X, f(X)) \mid f \in \mathcal{F}\}$.

Definition 1.1.15 (HSS satisfying Pseudorandomness of Outputs)**.** We say an HSS $\mathsf{HSS} = (\mathsf{HSS.Gen}, \mathsf{HSS.Share}, \mathsf{HSS.Eval})$ for a function family $\mathcal{F} := \{f \colon \mathcal{R}^n \to \mathcal{R}^m\}$ satisfies *pseudorandomness of outputs*, if for all $f \colon \mathcal{R}^n \to \mathcal{R}^m \in \mathcal{F}$, $(\mathsf{sk}, \{\mathsf{ek}_\sigma\}_{\sigma\in\{0,1\}}) \leftarrow \mathsf{HSS.Gen}(1^\lambda)$, $X \xleftarrow{\$} \mathcal{U}^n(\mathcal{R})$, $(\mathsf{k}_0, \mathsf{k}_1) \xleftarrow{\$} \mathsf{Share}(\mathsf{sk}, X)$, and $\sigma \in \{0, 1\}$ the output $R_\sigma \xleftarrow{\$} \mathsf{HSS.Eval}(\sigma, \mathsf{ek}_\sigma, \mathsf{k}_\sigma, f)$ is distributed computationally close to uniformly at random over the output space.

Note that if $f(\mathcal{U}^n(\mathcal{R}))$ is close to being uniformly random on $\mathcal{R}^m$, this property follows from the security of HSS.

Theorem 1.1.16 (PCG for Additive Correlations from HSS)**.** *Let $\mathcal{R}$ be a ring and $n, m, \ell \in \mathbb{N}$. Let $\mathcal{F} \subseteq \{f \colon \mathcal{R}^n \to \mathcal{R}^m\}$ be a family of functions. Let* PRG *be*

a $\mathcal{D}^\ell$-PRG and $\mathsf{HSS} = (\mathsf{HSS.Gen}, \mathsf{HSS.Share}, \mathsf{HSS.Eval})$ *an HSS with overhead* O_{HSS}[6] *for the family of functions* $\mathcal{F}_{\mathsf{HSS}} := \{f \circ \mathsf{PRG} \colon \mathcal{R}^\ell \to \mathcal{R}^m, r \mapsto f(\mathsf{PRG}(r)) \mid f \in \mathcal{F}\}$ *that further satisfies pseudorandomness of outputs. Then,* $\mathsf{PCG} = (\mathsf{PCG.Setup}, \mathsf{PCG.Gen}, \mathsf{PCG.Expand})$ *as defined in Figure 1.1.2 is a* PCG *for the correlation generator $C_{\mathcal{F}}$ with key-length upper bounded by $\ell \cdot O_{\mathsf{HSS}}$.*

Proof. Correctness. Let $f \in \mathcal{F}$. We have

$$
\begin{aligned}
&\{(\mathcal{R}_0, \mathcal{R}_1) \mid (\mathsf{k}_0, \mathsf{k}_1) \xleftarrow{\$} \mathsf{PCG.Gen}(1^\lambda), \mathcal{R}_\sigma \leftarrow \mathsf{PCG.Expand}(\sigma, \mathsf{k}_\sigma) \text{ for } \sigma \in \{0,1\}\} \\
&\overset{c}{\approx} \{(\mathcal{R}_0, \mathcal{R}_1) \mid (\mathsf{sk}, \{\mathsf{ek}_\sigma\}_{\sigma\in\{0,1\}}) \leftarrow \mathsf{HSS.Gen}(1^\lambda), r \xleftarrow{\$} \mathcal{D}^\ell(\mathcal{R}), \\
&\qquad\qquad (\mathsf{k}_0, \mathsf{k}_1) \leftarrow \mathsf{HSS.Share}(\mathsf{sk}, r), \mathcal{R}_0 \leftarrow \mathsf{HSS.Eval}(0, \mathsf{ek}_0, \mathsf{k}_0, f \circ \mathsf{PRG}), \\
&\qquad\qquad \mathcal{R}_1 := f(\mathsf{PRG}(r)) - \mathcal{R}_0\} \\
&\overset{c}{\approx} \{(\mathcal{R}_0, \mathcal{R}_1) \mid r \xleftarrow{\$} \mathcal{D}^\ell(\mathcal{R}), \mathcal{R}_0 \leftarrow \mathcal{R}^m, \mathcal{R}_1 := f(\mathsf{PRG}(r)) - \mathcal{R}_0\} \\
&\overset{c}{\approx} \{(\mathcal{R}_0, \mathcal{R}_1) \mid X \xleftarrow{\$} \mathcal{U}^n(\mathcal{R}), \mathcal{R}_0 \leftarrow \mathcal{R}^m, \mathcal{R}_1 := f(X) - \mathcal{R}_0\}
\end{aligned}
$$

as required, where the first transition follow by correctness of HSS, the second by pseudorandomness of outputs of HSS and the last by pseudorandomness of PRG.

Security. Let $\sigma \in \{0,1\}$. We have

$$
\begin{aligned}
&\{(\mathsf{k}_{1-\sigma}, \mathcal{R}_\sigma) \mid (\mathsf{k}_0, \mathsf{k}_1) \xleftarrow{\$} \mathsf{PCG.Gen}(1^\lambda), \mathcal{R}_\sigma \leftarrow \mathsf{PCG.Expand}(\sigma, \mathsf{k}_\sigma)\} \\
&\overset{c}{\approx} \{(\mathsf{k}_{1-\sigma}, \mathcal{R}_\sigma) \mid (\mathsf{sk}, \{\mathsf{ek}_\sigma\}_{\sigma\in\{0,1\}}) \leftarrow \mathsf{HSS.Gen}(1^\lambda), r \xleftarrow{\$} \mathcal{D}^\ell(\mathcal{R}), \\
&\qquad\qquad (\mathsf{k}_0, \mathsf{k}_1) \leftarrow \mathsf{HSS.Share}(\mathsf{sk}, r), \\
&\qquad\qquad \mathcal{R}_{1-\sigma} \leftarrow \mathsf{HSS.Eval}(1-\sigma, \mathsf{ek}_{1-\sigma}, \mathsf{k}_{1-\sigma}, f \circ \mathsf{PRG}), \\
&\qquad\qquad \mathcal{R}_\sigma := f(\mathsf{PRG}(r)) - \mathcal{R}_{1-\sigma}\} \\
&\overset{c}{\approx} \{(\mathsf{k}_{1-\sigma}, \mathcal{R}_\sigma) \mid (\mathsf{sk}, \{\mathsf{ek}_\sigma\}_{\sigma\in\{0,1\}}) \leftarrow \mathsf{HSS.Gen}(1^\lambda), r \xleftarrow{\$} \mathcal{D}^\ell(\mathcal{R}), r' \xleftarrow{\$} \mathcal{D}^\ell(\mathcal{R}), \\
&\qquad\qquad (\mathsf{k}_0, \mathsf{k}_1) \leftarrow \mathsf{HSS.Share}(\mathsf{sk}, r'), \\
&\qquad\qquad \mathcal{R}_{1-\sigma} \leftarrow \mathsf{HSS.Eval}(1-\sigma, \mathsf{ek}_{1-\sigma}, \mathsf{k}_{1-\sigma}, f \circ \mathsf{PRG}), \\
&\qquad\qquad \mathcal{R}_\sigma := f(\mathsf{PRG}(r)) - \mathcal{R}_{1-\sigma}\} \\
&\overset{c}{\approx} \{(\mathsf{k}_{1-\sigma}, \mathcal{R}_\sigma) \mid (\mathsf{sk}, \{\mathsf{ek}_\sigma\}_{\sigma\in\{0,1\}}) \leftarrow \mathsf{HSS.Gen}(1^\lambda), X \xleftarrow{\$} \mathcal{U}^n(\mathcal{R}), r' \xleftarrow{\$} \mathcal{D}^\ell(\mathcal{R}), \\
&\qquad\qquad (\mathsf{k}_0, \mathsf{k}_1) \leftarrow \mathsf{HSS.Share}(\mathsf{sk}, r'), \\
&\qquad\qquad \mathcal{R}_{1-\sigma} \leftarrow \mathsf{HSS.Eval}(1-\sigma, \mathsf{ek}_{1-\sigma}, \mathsf{k}_{1-\sigma}, f \circ \mathsf{PRG}), \\
&\qquad\qquad \mathcal{R}_\sigma := f(X) - \mathcal{R}_{1-\sigma}\},
\end{aligned}
$$

[6] We say a HSS has *overhead* O_{HSS}, if for every input the share size does not exceed O_{HSS} times the input size.

where the first transition follows by correctness of HSS, the second transition by security of HSS and the last by pseudorandomness of PRG.

□

1.1.7 Further Discussions

Computational security. Unlike secret sharing with basic linear homomorphism, it can be shown that most nontrivial FSS and HSS cannot provide information theoretic hiding [57, 20, 25]. For example, even for simple classes $\mathcal{F}$ (such as the class of point functions), the best possible solution with information theoretic security is to additively share the truth-table representation of f, whose shares consist of 2^n group elements. But if one considers a *computational* notion of hiding, then there are no apparent limitations to what can be done for polynomial-time computable f. This is what we refer to when we speak of FSS/HSS.

Homomorphic Secret Sharing vs. Fully Homomorphic Encryption. HSS can be viewed as a relaxed version of fully homomorphic encryption (FHE) [74, 54], where instead of a single party homomorphically evaluating on encrypted data, we allow homomorphic evaluation to be distributed among two parties who do not interact with each other. As in the case of FHE, we require that the output of Eval be *compact* in the sense that its length depends only on the output length $|P(x)|$ but not on the size of P. But in fact, a unique feature of HSS that distinguishes it from traditional FHE is that the output representation can be *additive*. E.g., we can achieve $\mathsf{Eval}(x^0, P) + \mathsf{Eval}(x^1, P) = P(x) \bmod \beta$ for some positive integer $\beta \geq 2$ that can be chosen arbitrarily. This enables an ultimate level of compactness and efficiency of reconstruction that is impossible to achieve via standard FHE. For instance, if P outputs a single bit and $\beta = 2$, then the output $P(x)$ is reconstructed by taking the exclusive-or of two bits.

Other related notions. We note that other forms of secret sharing of functions and homomorphic secret sharing have been considered in the literature. An initial study of secret sharing homomorphisms is due to Benaloh [9], who presented constructions and applications of additively homomorphic secret sharing schemes. Further exploration of computing on secret shared data took place in [6]. Secret sharing of functions has appeared in the context of threshold cryptography (cf. [43, 42]). However, these other notions either apply only to very specific function classes that enjoy homomorphism properties compatible with the secret sharing, or alternatively they do not require a simple (e.g., additive) representation of the output which is essential for the applications we consider.

1.1.8 Historical Notes

Function secret sharing was initially introduced for the special case of point functions in [57]. The notion was extended to general functions in [20], and the one-way function-based construction for point function was optimized and generalized in [22]. The dual notion of homomorphic secret sharing was introduced in [21], and improvements of the initial construction were subsequently described in [23, 19]. Theoretical foundations for homomorphic secret sharing were developed in [25].

Pseudorandom correlation generators in the 2-party setting were initially considered in [19] (under the name "cryptographic capsules"). Previously, in the multi-party setting, PCGs for *linear* correlations were considered in [56, 41]. PCGs of the latter kind are sometimes referred to as *pseudorandom secret sharing* (PRSS). Constructions of PCGs for increasingly more complex correlations were described in [14, 16, 18].

1.2 Constructions of FSS and HSS

We now cover a variety of selected constructions of FSS and HSS from the literature. Existing constructions can be naturally categorized according to their expressiveness and the strength of the underlying assumption. More precisely, we will cover "low-end" constructions of FSS (capturing restricted classes such as point functions, intervals, comparisons, and decision trees, assuming only the existence of one-way functions), "mid-range" constructions of HSS (capturing all constant-depth circuits, from a weak flavor of the learning parity with noise assumption which is not known to imply public key cryptography), and "high-end" constructions of HSS (for branching programs and general circuits, from public key assumptions such as DDH, DCR, and LWE). We will only consider here schemes with $t = m - 1$, namely security against all-but-one server, focusing mainly on the case $m = 2$. See [25, 67, 65, 31, 53] for constructions of HSS schemes with $t < m - 1$.

1.2.1 Cryptographic Assumptions

The constructions of this section build upon a variety of standard cryptographic assumptions. We assume familiarity with basic cryptographic primitives such as a one-way function (OWF) and a pseudorandom generator (PRG); see [58] for formal definitions. For the sake of completeness, however, we recall two of the main cryptographic assumptions used in this line of work: the Learning Parity with Noise (LPN) assumption, and the Decisional Diffie-Hellman (DDH) assumption.

The LPN Assumption. The Learning Parity with Noise (LPN) assumption [10] states that it is hard to solve a system of uniformly random linear equations over $\mathbb{F}_2$ where each bit is *flipped* with some small probability ε. It can be shown that this assumption is equivalent to the assumption that noisy linear equations are *pseudorandom*. Many variants of LPN are standard in the literature, over other fields than $\mathbb{F}_2$, with different noise distributions, or with a different distribution of the equations.

An equivalent formulation of LPN is the following: it is hard to decode a noisy codeword from a random *linear code*. Changing the distribution over equations is equivalent to considering other types of linear codes. Below, we provide a general definition of LPN over a ring $\mathcal{R}$, with respect to a *code generator* $\mathbf{C}$ which samples the system of equations, and a *noise distribution* $\mathcal{D}$ over $\mathcal{R}$.

Definition 1.2.1 (LPN)**.** Let $\mathcal{D}(\mathcal{R}) = \{\mathcal{D}_{k,q}(\mathcal{R})\}_{k,q\in\mathbb{N}}$ denote a family of distributions over a ring $\mathcal{R}$, such that for any $k, q \in \mathbb{N}$, $\mathsf{Im}(\mathcal{D}_{k,q}(\mathcal{R})) \subseteq \mathcal{R}^q$. Let $\mathbf{C}$ be a probabilistic code generation algorithm such that $\mathbf{C}(k, q, \mathcal{R})$ outputs a matrix $A \in \mathcal{R}^{k\times q}$. For dimension $k = k(\lambda)$, number of samples (or block length) $q = q(\lambda)$, and ring $\mathcal{R} = \mathcal{R}(\lambda)$, the $(\mathcal{D}, \mathbf{C}, \mathcal{R})$-$\mathsf{LPN}(k, q)$ assumption states that

$$\{(A, \vec{b}) \mid A \xleftarrow{\$} \mathbf{C}(k, q, \mathcal{R}), \vec{e} \xleftarrow{\$} \mathcal{D}_{k,q}(\mathcal{R}), \vec{s} \xleftarrow{\$} \mathcal{R}^k, \vec{b} \leftarrow \vec{s} \cdot A + \vec{e}\}$$
$$\overset{\mathrm{c}}{\approx} \{(A, \vec{b}) \mid A \xleftarrow{\$} \mathbf{C}(k, q, \mathcal{R}), \vec{b} \xleftarrow{\$} \mathcal{R}^q\},$$

where $\overset{\mathrm{c}}{\approx}$ denotes computational indistinguishability.

Beyond the standard Bernoulli noise distribution (where each coordinate is noisy with some probability ε), other standard noise distributions include exact noise (a random fixed-size subset of coordinates are noisy), which can be shown to be equivalent to Bernoulli noise, and regular noise (where the output is divided into blocks, where a single random entry of each block is picked uniformly at random).

Example: LPN with Fixed Weight Noise. For a finite field $\mathbb{F}$, we denote by $\mathcal{HW}_{r,n}(\mathbb{F})$ the distribution of uniform, weight r vectors over $\mathbb{F}^n$; that is, a sample from $\mathcal{HW}_{r,n}(\mathbb{F})$ is a uniformly random nonzero field element in r random positions out of n, and zero elsewhere.

Dual LPN. It is often convenient to work with an equivalent *dual* form of the LPN assumption, which we state below:

Definition 1.2.2 (Dual LPN)**.** Let $\mathcal{D}(\mathcal{R})$ and $\mathbf{C}$ be as in Definition 1.2.1, $n, n' \in \mathbb{N}$ with $n' > n$, and define $\mathbf{C}^{\perp}(n', n, \mathcal{R}) = \{B \in \mathcal{R}^{n'\times n} : A \cdot B = 0, A \in \mathbf{C}(n' - n, n', \mathcal{R}), \mathsf{rank}(B) = n\}$.

For $n = n(\lambda), n' = n'(\lambda)$ and $\mathcal{R} = \mathcal{R}(\lambda)$, the $(\mathcal{D}, \mathbf{C}, \mathcal{R})$-dual-LPN$(n', n)$ assumption states that

$$\{(H, \vec{b}) \mid H \xleftarrow{\$} \mathbf{C}^{\perp}(n', n, \mathcal{R}), \vec{e} \xleftarrow{\$} \mathcal{D}(\mathcal{R}), \vec{b} \leftarrow \vec{e} \cdot H\}$$
$$\overset{c}{\approx} \{(H, \vec{b}) \mid H \xleftarrow{\$} \mathbf{C}^{\perp}(n', n, \mathcal{R}), \vec{b} \xleftarrow{\$} \mathcal{R}^n\}.$$

The DDH Assumption. Let DHGen be a deterministic algorithm that on input 1^λ returns a description $\mathcal{G} = (\mathbb{G}, p)$ where $\mathbb{G}$ is a cyclic group of prime order p. Then the decisional Diffie-Hellman assumption is defined as follows.

Definition 1.2.3 (DDH Assumption)**.** We say that the decisional Diffie-Hellman (DDH) assumption holds relative to DHGen if for all probabilistic polynomial time adversaries Adv,

$$\Pr\left[\begin{array}{ll} \mathcal{G} \leftarrow \mathsf{DHGen}(1^\lambda), & \\ g \xleftarrow{\$} \mathbb{G}, \alpha, \beta, \gamma \xleftarrow{\$} \mathbb{Z}_p, & : \mathsf{Adv}(g, g^\alpha, g^\beta, g_b) = b \\ b \xleftarrow{\$} \{0,1\}, g_b \leftarrow (g^{\alpha\beta})^b \cdot (g^\gamma)^{1-b} & \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

Here, note that DHGen outputs a fixed group $\mathbb{G}$ per security parameter.

The DCR Assumption. Let DCRen be a randomized algorithm that on input 1^λ returns (N, p, q), where $N = pq$ and p, q are random safe primes (a safe prime is a prime of the form $2x + 1$ where x is also prime) of length $\ell(\lambda)$.

Definition 1.2.4 (DCR Assumption)**.** We say that the decisional composite residuosity (DCR) assumption holds relative to DCRen if for all probabilistic polynomial time adversaries Adv,

$$\Pr\left[\begin{array}{ll} (N, p, q) \leftarrow \mathsf{DCRen}(1^\lambda), & \\ g_0 \xleftarrow{\$} \mathbb{Z}^*_{N^2}, g_1 \leftarrow g_0^N \bmod N^2, & : \mathsf{Adv}(N, g_b) = b \\ b \xleftarrow{\$} \{0,1\} & \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

1.2.2 An Overview of the State of the Art

FSS/HSS constructions as of the writing of this chapter (November 2021) are as follows. The given complexity measures are with respect to n-bit inputs. Unless otherwise specified, the results of this section are for $m = 2$ servers.

"Low End": FSS from One-Way Functions

For simple but useful function classes, FSS can be constructed from the (minimal) assumption that a OWF exists. The following constructions all make a black-box use of an arbitrary PRG. We let λ denote a concrete computational security parameter that serves as the PRG seed length. In an AES-based implementation, one can take $\lambda = 128$.

- **Point functions** ("Distributed Point Functions").

  The class of point functions consists of functions of the form $f_{\alpha,\beta}$, such that $f_{\alpha,\beta}(x)$ outputs β if $x = \alpha$ and 0 otherwise. Here $x, \alpha \in \{0,1\}^n$ and $\beta \in \mathbb{G}$ for an Abelian group $\mathbb{G}$. We let $|\beta|$ denote the bit-length of a representation of a group element. When $|\beta|$ is omitted it is understood to be 1. A *distributed point function* (DPF) is an FSS scheme for the class of point functions.

  - The first nontrivial (2-party) DPF was implicitly constructed in [35] in the context of computationally private information retrieval. The key size of this construction is $2^{O(\sqrt{n})} \cdot \lambda$, which is not polynomial in the input size n but still a super-polynomial improvement over the naive solution of additively sharing the size-2^n truth table.
  - The notion of DPF was first introduced in [57], which also gave a recursive construction with key size $O(n^{\log_2 3} \cdot \lambda)$. This was improved to $O(n\lambda)$ bits in [20] via a tree-based construction.
  - The current best construction [22] has key size $\approx n\lambda + |\beta|$. More precisely, the key size is $\lambda + n(\lambda + 2) - \lfloor \log \lambda / |\beta| \rfloor$ bits.[7]

  For $m > 2$ servers: the best known construction, presented in [20] has key size $O(2^m 2^{n/2} \cdot \lambda)$ bits. For a small number of servers m, this gives a near-quadratic improvement over the naive solution of secret-sharing the truth-table. The question of improving this bound, e.g., obtaining a 3-server DPF with key size $O(2^{n/3} \cdot \lambda)$ from one-way functions, is one of the central open questions in the area.[8]

- **Comparison and Intervals** [20, 22, 13].

  The class of comparison functions consists of functions f_a which output 1 on inputs x with $a < x$. Interval functions $f_{(a,b)}$ output 1 precisely for inputs

[7] In particular: $\lambda + n(\lambda + 2)$ for λ-bit outputs, and $\lambda + n(\lambda + 2) - \lfloor \log \lambda \rfloor$ for 1-bit outputs.

[8] Here we only consider the case where security should hold against $t = m - 1$ colluding servers. Settling for a smaller security threshold, better DPF constructions can be obtained from information-theoretic private information retrieval schemes [36] or from OWF-based DPF [31].

x that lie within the interval $a < x < b$, and 0 otherwise. Constructions of FSS for such functions follow a similar structure as DPFs. The best key size for comparison function is comparable to a DPF and for interval functions it is roughly twice the size [22, 13]. Optimized FSS schemes for the related classes of shifted-ReLU and spline (piecewise-polynomial) functions were presented in [24, 13].

- NC^0 **predicates** (i.e., functions with constant locality) [20].

  For locality d, the key size grows as $O(\lambda \cdot n^d)$. For example, this includes bit-matching predicates that check a constant number of bits d.

- **Decision trees with topology leakage** [22].

  A decision tree is defined by: (1) a tree topology, (2) variable labels on each node v (where the set of possible values of each variable is known), (3) value labels on each edge (the possible values of the originating variable), and (4) output labels on each leaf node.

  In the construction of [22], the key size is roughly $\lambda \cdot |V|$ bits, where V is the set of nodes, and evaluation on a given input requires $|V|$ executions of a pseudorandom generator, and a comparable number of additions. The FSS is guaranteed to hide the secret edge value labels and leaf output labels, but (in order to achieve this efficiency) reveals the base tree topology and the identity of which variable is associated to each node.

  Constant-dimensional intervals. A sample application of FSS for decision trees is constant d-dimensional interval queries: that is, the functions $f(x_1, \cdots, x_d)$ which evaluate to a selected nonzero value precisely when $a_i \leq x_i \leq b_i$ for some secret interval ranges $(a_i, b_i)_{i \in [d]}$. For n-bit inputs x_i, FSS for d-dimensional intervals can be obtained with key size and computation time $O(\lambda \cdot n^d)$. For small values of d, such as $d = 2$ for supporting a conjunction of intervals, this yields solutions with reasonably good concrete efficiency.

Other PRG-based constructions of FSS schemes for function classes that are motivated by mixed-mode secure computation are presented in [24, 13]. In the context of private information retrieval, FSS schemes for the above function classes can be combined with server-side database operations to emulate private database search with richer query classes, such as Max/Min and top-k [81]. See Section 1.3.2 for discussion of these and other applications of FSS from symmetric cryptography.

"Mid Range": HSS from LPN-Style Assumptions

The LPN assumption and its variants can be viewed as lying in between symmetric cryptography and public-key cryptography. While LPN with low noise is known to imply public-key encryption [1], it is not known to imply additively homomorphic encryption (or even collision-resistant hashing, except in an extreme parameter regime [28]). Still, in the context of HSS and especially in the related context of PCGs, LPN turns out to be surprisingly useful. Under LPN, the following HSS-based constructions are known:

- **Constant-degree polynomials**, from standard LPN [16];
- **Circuits of** $\log\log$**-depth**, from the superpolynomial hardness of LPN [39].

In both of the above cases, the share size is sublinear in the description size of the function being evaluated. This rules out, for instance, HSS for constant-degree polynomials that uses additive sharing of all monomials. An HSS scheme for degree-2 polynomials with compact (but not additive) output shares was previously constructed in [32] from threshold additively homomorphic encryption.

"High End": HSS from Public-Key Cryptography

Using standard public-key cryptography assumptions, HSS schemes exist for much richer function representation classes, including branching programs (capturing functions in NC^1 and logspace) and even general circuits (capturing general polynomial time computations). Concretely, the following results are known.

- **Branching programs**: Allowing for inverse-polynomial error (namely, δ-correctness for inverse polynomial δ), a construction from Decisional Diffie-Hellman (DDH) was presented in [21]. In this construction the running time of Eval is $\tilde{O}(s^2/\delta)$, where s is the branching program size and δ is the error probability (which can be made detectable, namely of a Las-Vegas type). This was subsequently improved to $\tilde{O}(s^{1.5}/\delta^{0.5})$ in [46]. Optimized variants of the DDH-based constructions for simple but useful function classes are given in [23, 19]. A similar construction from the Decisional Composite Residuosity (DCR) assumption was presented in [52].
- **Branching programs**: With full (negligible-error) correctness, HSS constructions from the DCR assumption were presented in [69, 76]. In these constructions, the running time scales linearly with s.

- **Circuits**: A general-purpose HSS scheme for Boolean circuits based on the Learning With Errors (LWE) assumption was presented in [49]. More concretely, in the language of Definition 1.1.4: Additive (n, m)-HSS for arbitrary n, m and circuits of a fixed polynomial size can be obtained from the Learning With Errors (LWE) assumption, by a simple variation of the FSS construction from spooky encryption of [49] (more specifically, their techniques for obtaining 2-round MPC). Here the share size depends on the bound on the circuit size. The HSS shares can be made independent of the circuit size by using a standard circular-secure variant of the LWE assumption that is used in constructions of fully homomorphic encryption. See [25] for details. (Earlier constructions of HSS for circuits from [20] were either based on indistinguishability obfuscation or only yielded *noisy* HSS from LWE.)

1.2.3 Distributed Point Function from OWF

We give an intuitive description of the (2-party) DPF $(\mathsf{Gen}^\bullet, \mathsf{Eval}^\bullet)$ construction from [22] (following the text therein). Recall that a DPF is an FSS scheme for the class of point functions $f_{\alpha,\beta} : \{0,1\}^n \to \mathbb{G}$ whose only nonzero evaluation is $f_{\alpha,\beta}(\alpha) = \beta$. For simplicity, consider the case of a DPF with a single-bit output $\mathbb{G} = \{0,1\}$ and $\beta = 1$.

Basic key structure. At a high level, each of the two DPF keys k_0, k_1 defines a GGM-style binary tree [59] with 2^n leaves, where the leaves are labeled by inputs $x \in \{0,1\}^n$. We will refer to a path from the root to a leaf labeled by x as the *evaluation path* of x, and to the evaluation path of the special input α as the *special evaluation path*. Each node v in a tree will be labeled by a string of length $\lambda + 1$, consisting of a *control bit* t and a λ-bit *seed* s, where the label of each node is fully determined by the label of its parent. The function $\mathsf{Eval}^\bullet$ will compute the labels of all nodes on the evaluation path to the input x, using the root label as the key, and output the control bit of the leaf.

Generating the keys. We would like to maintain the invariant that for each node outside the special path, the two labels (on the two trees) are identical, and for each node on the special path the two control bits are different and the two seeds are indistinguishable from being random and independent. Note that since the label of a node is determined by that of its parent, if this invariant is met for a node outside the special path then it is automatically maintained by its children. Also, we can easily meet the invariant for the root (which is always on the special path) by just explicitly including the labels in the keys. The challenge is to ensure that the invariant is maintained also when leaving the special path.

Towards describing the construction, it is convenient to view the two labels of a

node as a mod-2 additive secret sharing of its label, consisting of shares $[t] = (t_0, t_1)$ of the control bit t and shares $[s] = (s_0, s_1)$ of the λ-bit seed s. That is, $t = t_0 \oplus t_1$ and $s = s_0 \oplus s_1$. The construction employs two simple ideas.

1. In the 2-party case, additive secret sharing satisfies the following weak homomorphism: If G is a PRG, then $G([s]) = (G(s_0), G(s_1))$ extends shares of the 0-string $s = 0$ into shares of a longer 0-string $S = 0$, and shares of a random seed s into shares of a longer (pseudo-)random string S, where S is pseudo-random even given one share of s.

2. Additive secret sharing is additively homomorphic: given shares $[s], [t]$ of a string s and a bit t, and a public correction word CW, one can locally compute shares of $[s \oplus (t \cdot CW)]$. We view this as a *conditional correction* of the secret s by CW conditioned on $t = 1$.

To maintain the above invariant along the evaluation path, we use the two types of homomorphism as follows. Suppose that the labels of the i-th node v_i on the evaluation path are $[s], [t]$. To compute the labels of the $(i + 1)$-th node, the parties start by locally computing $[S] = G([s])$ for a PRG $G : \{0,1\}^\lambda \to \{0,1\}^{2\lambda+2}$, parsing $[S]$ as $[s^L, t^L, s^R, t^R]$. The first two values correspond to labels of the left child and the last two values correspond to labels of the right child.

To maintain the invariant, the keys will include a correction word CW for each level i. As discussed above, we only need to consider the case where v_i is on the special path. By the invariant we have $t = 1$, in which case the correction will be applied. Suppose without loss of generality that $\alpha_i = 1$. This means that the left child of v_i is off the special path whereas the right child is on the special path. To ensure that the invariant is maintained, we can include in both keys the correction $CW^{(i)} = (s^L, t^L, s^R \oplus s', t^R \oplus 1)$ for a random seed s'. Indeed, this ensures that after the correction is applied, the labels of the left and right child are $[0], [0]$ and $[s'], [1]$ as required. But since we do not need to control the value of s', except for making it pseudo-random, we can instead use the correction $CW^{(i)} = (s^L, t^L, s^L, t^R \oplus 1)$ that can be described using $\lambda + 2$ bits. This corresponds to $s' = s^L \oplus s^R$. The n correction values $CW^{(i)}$ are computed by $\mathsf{Gen}^\bullet$ from the root labels by applying the above iterative computation along the special path, and are included in both keys.

Finally, assuming that $\beta = 1$, the output of $\mathsf{Eval}^\bullet$ is just the shares $[t]$ of the leaf corresponding to x. A different value of β (from an arbitrary Abelian group) can be handled via an additional correction $CW^{(n+1)}$.

The above construction can be generalized to arbitrary groups $\mathbb{G}$ and value β; additional optimizations are described in [22]. The resulting construction is summarized in the following theorem.

Theorem 1.2.5 (PRG-based DPF [22], Theorems 3.3 and 3.4)**.** *Given a PRG* $G : \{0,1\}^\lambda \to \{0,1\}^{2\lambda+2}$*, there exists a DPF for point functions* $f_{\alpha,\beta} : \{0,1\}^n \to \mathbb{G}$ *with key size* $n\cdot(\lambda+2)+\lambda+\lceil\log_2|\mathbb{G}|\rceil$ *bits. For* $m = \lceil\frac{\log|\mathbb{G}|}{\lambda+2}\rceil$*, the key generation algorithm* Gen *invokes* G *at most* $2(n+m)$ *times, the evaluation algorithm* Eval *invokes* G *at most* $n+m$ *times, and the full evaluation algorithm* FullEval *invokes* G *at most* $2^n(1+m)$ *times.*

In the above, FullEval refers to an algorithm evaluating Eval on all points of the domain. While FullEval can always be obtained by running Eval on all points in parallel, the PRG-based DPF allows for optimizations taking advantage of the tree structure of the construction. The FullEval algorithm is particularly useful when constructing pseudorandom correlation generators using DPFs.

From DPF to FSS for Multi-Point Functions. A *multi-point function* generalizes a point function in the natural way: $f_{S,\vec{\beta}} : \{0,1\}^n \mapsto \mathbb{G}$ maps to 0 everywhere, except on t points $S = (s_1, \cdots, s_t)$ (which we will typically view as an ordered multi-set) where it evaluates respectively to $\vec{\beta} = (\beta_1, \cdots, \beta_t)$. Due to the additive structure of $\mathbb{G}$, any t-multipoint function can be written as the sum of t point functions. This immediately implies that an FSS for t-multipoint FSS (t-MPFSS) can be constructed from t instances of a DPF.

Theorem 1.2.6 (PRG-based MPFSS [14])**.** *Given a PRG* $G : \{0,1\}^\lambda \to \{0,1\}^{2\lambda+2}$*, there exists an MPFSS for* t*-multipoint functions* $f_{S,\vec{\beta}} : \{0,1\}^n \to \mathbb{G}$ *with key size* $t\cdot(n\cdot(\lambda+2)+\lambda+\lceil\log_2|\mathbb{G}|\rceil)$ *bits.*

Multi-Party DPF. A multi-party DPF is a DPF scheme for $m > 2$ parties with full threshold, i.e. the adversary controls $m-1$ parties. The key size of the best multi-party DPF scheme we know [20] is roughly a square root of the size of the trivial DPF scheme, as opposed to the exponential reduction of the key size in the two-party case. The main difference between the two schemes is that the multi-party scheme does not have a recursive structure. We first describe the scheme for binary outputs and then generalize to output domains $\mathbb{Z}_q$ for any integer q.

A DPF in this setting cannot use the implicit encoding of values in the two-party setting in which a 0 value is represented by two identical seeds for a PRG and a 1 value is represented by two pseudorandom seeds for the PRG. In multi-party DPF this representation does not hide the values 0 or 1 since they can be easily distinguished by an adversary that controls at least two parties.

As an alternative encoding consider an m-party additive secret sharing of $a \in \mathbb{Z}_2$. Every sharing consists of m shares $sh_1, \ldots, sh_m \in \mathbb{Z}_2$ such that $\sum_{i=1}^m sh_i \equiv a \bmod 2$. The proposed encoding of a consists of *all* 2^{m-1} vectors of additive shares of a in randomly permuted order. Party i receives the i-th share in each vector.

Clearly, the view of any proper subset of parties is the same regardless of the secret a.

The key generation algorithm for multi-party DPF can utilize this encoding by assigning a random seed s_j to the j-th vector of shares $(sh_{j,1}, \ldots, sh_{j,m}) \in \mathbb{Z}_q^m$. The i-th party receives (j, s_j) if and only if $sh_{j,i} = 1$. Now, given a public correction word CW, the parties can locally expand the seeds add them together and conditionally correct the shared secret only for encoded 1 values.

Unlike the two-party case, the local evaluation procedure cannot first expand a PRG seed and then use CW to modify an encoding of 1 into an encoding of a longer vector with Hamming weight 1. In the multi-party case this would leak information on the location of the point α via shared values across parties. However, this procedure can turn an *encoding* of 1 into an *additive sharing* of a vector with Hamming weight 1.

Putting the pieces together, a key in a multi-party DPF divides the input domain $\{0,1\}^n$ into n' segments, where n' is chosen to minimize the key size, and assigns a block of seeds to each segment. The segment that includes α is assigned a block that encodes 1 and the blocks in all other segments encode 0. Each party expands its seeds for each segment to $2^n/n'$ bits and XORs all the expanded seeds associated with a segment. The parties now have an additive sharing of a vector of zeroes for each segment that does not include α and an additive sharing of a pseudorandom vector for the segment that includes α.

To convert this sharing into an additive sharing of the truth table of $f_{\alpha,\beta}$ the Gen algorithm includes a correction word of length $2^n/n'$ that each party adds to its share of a segment if it received $(1, s_1)$ for that segment. Since s_1 is distributed to an odd number of parties in the segment that contains α and to an even number of parties in any other segment, the correction word affects only the "interesting" segment and can be chosen so that the sum of all shares is exactly the truth table of $f_{\alpha,\beta}$. In detail, if v is a binary vector of length $2^n/n'$ that represents the restriction of the truth table of $f_{\alpha,\beta}$ to the interesting segment and $s_1, \ldots, s_{2^{m-1}}$ are all the seeds associated with the interesting segment then $CW = v + \sum_{j=1}^{2^{m-1}} G(s_j)$ with addition in $\mathbb{F}_2^{2^n/n'}$.

We generalize this scheme to any input domain $\mathbb{Z}_q$ in two steps, first for a prime power q and then for general integers. For a prime power q, the encoding of $a \in \{0,1\} \subseteq \mathbb{Z}_q$ is a random permutation of all m-vectors $(sh_{j,1}, \ldots, sh_{j,m}) \in \mathbb{Z}_q^m$ such that $\sum_{i=1}^m sh_{j,i} \equiv a \bmod q$. Gen chooses an encoding for each of n' segments of the input domain and assigns a seed s_j to each m-vector of shares. The i-th party receives a tuple $(j, s_j, sh_{j,i})$ if and only if $sh_{j,i} \neq 0$. In the Eval procedure, the i-th party computes $\sum_j sh_{j,i} \cdot G(s_j) \in \mathbb{Z}_q^{2^n/n'}$ for every segment. The key also includes a public correction word $CW \in \mathbb{Z}_q^{2^n/n'}$ chosen so that after each party adds $sh_1 \cdot CW$

to its share, the parties share the truth table of the point function.

In the general case of output domain $\mathbb{Z}_q$ where q has prime factorization $q = p_1^{e_1} \cdots p_\ell^{e_\ell}$, for primes $p_1, \ldots, p_\ell$ and integers $e_1, \ldots, e_\ell$, the above construction is repeated independently for all prime powers $q_k = p_k^{e_k}$. The additive output shares over $\mathbb{Z}_{q_1} \times \ldots \times \mathbb{Z}_{q_\ell}$ are locally combined to output shares over $\mathbb{Z}_q$ using the Chinese Remainder Theorem.

The above construction is captured by the following theorem, which generalizes a similar construction from [20] to any output domain $\mathbb{Z}_q$.

Theorem 1.2.7 (PRG-based multi-party DPF, generalizing [20], Section 3.1)**.** *Let λ be a security parameter, let m, n and q be integers such that $m > 2$ and $q = q_1 \cdots q_\ell$ is the factorization of q into mutually prime factors, with $q_1 = \max\{q_1, \ldots, q_\ell\}$. Given a PRG with a λ-bit seed, there exists an m-party DPF for point functions $f_{\alpha,\beta} : \{0,1\}^n \to \mathbb{Z}_q$ with key size $O(2^{n/2} \cdot \ell(\lambda q_1^{m-1} \log q_1)^{1/2})$.*

A consequence of Theorem 1.2.7 is that the multi-party DPF scheme is efficient only for output domains $\mathbb{Z}_q$ with smooth q due to the exponential dependence of the key size on the largest prime power that factors q.

We next provide a more precise bound on the key size for a prime power q and output domain $\mathbb{Z}_q$. Gen samples q^{m-1} PRG seeds s_j for each of n' segments. Each party's key includes for q^{m-2} of these seeds a tuple $(j, s_j, sh_{j,i})$. Since $j \le q^{m-1}$ and $sh_{i,j} \in \mathbb{Z}_q$ the total size of a tuple is $\lambda + m \log q$. Note that $\lambda + m \log q \le (q+1)\lambda$ since q^{m-1} is a feasible size (Gen samples q^{m-1} seeds), while 2^λ is infeasible. In addition, each key includes the public correction word, which is of size $\frac{2^n \log q}{n'}$. Therefore, the key size is at most $n' q^{m-2}(q+1)\lambda + 2^n \log q / n'$. Setting $n' = \left((2^n \log q)/(q^{m-1}\lambda)\right)^{1/2}$ we get that the key size is $(2^{n+2}\lambda q^{m-1} \log q)^{1/2} \cdot (1 + o(1))$. For general integers q the key size is thus bound by plugging each relatively prime factor q_i of q in the above expression and summing over all these terms.

The dominant factor in the running time of Gen, Eval and FullEval is the number of times that the PRG is called. Assuming a PRG $G : \{0,1\}^\lambda \to \{0,1\}^{C(n,q,\lambda)}$, for $C(n, q, \lambda) = \lceil (2^n \lambda q_1^{m-1} \log q_1)^{1/2} \rceil$, the algorithm Gen invokes G at most $\sum_{i=1}^{\ell} q_i^{m-1}$ times, the evaluation algorithm Eval invokes G at most $\sum_{i=1}^{\ell} q_i^{m-2}$ times, and the full evaluation algorithm FullEval invokes G at most $n' \sum_{i=1}^{\ell} q_i^{m-2}$ times.

1.2.4 HSS for Constant-Degree Polynomials from LPN

We now turn our attention to HSS for a more general class: the class of constant-degree polynomials. It is straightforward to build an information-theoretic scheme for degree-d n-variate polynomials, simply by additively sharing all monomials of degree d, which leads to a construction with share size $O(n^d)$. The construction we

describe in this section, on the other hand, has much smaller shares, of size $O(\lambda^d)$ (independent of n), where λ is a security parameter. It turns out that the low-end construction of (multi-point) FSS is one of the key building block in this more advanced construction, the other being the LPN assumption. At a high level, the construction proceeds as follows: it builds upon MPFSS to construct a *pseudorandom correlation generator* for a constant-degree correlation over *sparse vectors*. Then, it builds upon the linearity of the dual-LPN assumption to convert this into a constant-degree correlation over general pseudorandom generator, thus giving a full-fledged PCG for constant-degree correlation. Eventually, we show that a PCG for constant-degree correlations implies an HSS for constant-degree polynomials, providing a partial converse to the reverse implication which we described in Section 1.1.6.

We first describe the construction for the case of bilinear correlations. More precisely, we consider the following type of additive correlations: the party P_σ receives pseudorandom vectors $(\vec{x}_\sigma, \vec{z}_\sigma)$ such that $B(\vec{x}_0, \vec{x}_1) = \vec{z}_0 + \vec{z}_1$, where B is a bilinear function. We note that this type of correlation generalizes naturally to the setting where the entries $\vec{x}_0$ and $\vec{x}_1$ are additively shared between the parties (instead of being respectively known to one party).

Theorem 1.2.8. *Suppose the* $(\mathcal{HW}_{t,n'}, \mathbf{C}, \mathbb{F}_p)$-dual-LPN$(n', n)$ *assumption holds, and that* MPFSS *is a secure multi-point FSS scheme. Then the construction* G_{bil} *(Fig. 1.2.1) is a secure PCG for general bilinear correlations.*

Correctness follows by inspection, using the correctness of the MPFSS, and the bilinearity of the tensor product. We provide a sketch of the security analysis, focussing on the viewpoint of the player 0 (the other direction is symmetrical): the informations $(S_0, \vec{y}_0)$ and $(S_1, \vec{y}_1)$ uniquely specifies two error vectors $\vec{\mu}_0$ and $\vec{\mu}_1$. The party P_0 knows $(S_0, \vec{y}_0)$. By the security of the MPFSS, there is a simulator which can construct K_0^{fss} given only the allowed leakage on the point function, in a way that is indistinguishable from the true K_0^{fss}. In particular, this means that we can switch to a hybrid scenario where K_0^{fss} is generated truly independently of $\vec{\mu}_1$. Once $\vec{\mu}_1$ is not used anymore, the value $\vec{x}_1 = \vec{\mu}_1 \cdot H_{n',n}$ can be replaced by a truly random vector: this change is indistinguishable under the (dual) LPN assumption with respect to the code generation algorithm $\mathbf{C}$. This concludes the sketch of the proof.

Efficiency. Instantiating the MPFSS as in [14], the setup algorithm of G_{bil} outputs seeds of size $t^2 \cdot (\lceil \log n' \rceil (\lambda + 2) + \lambda + \log_2 |\mathbb{F}|)$ bits, which amounts to $\tilde{O}(t^2)$ field elements over a large field ($\log_2 |\mathbb{F}| = O(\lambda)$). Expanding the seed involves $(tn')^2$ PRG evaluations and $O(n \cdot n')^2 = O(n^4)$ arithmetic operations.

Construction G_{bil}

PARAMETERS: $1^\lambda, n, n', t, p \in \mathbb{N}$, where $n' > n$. A code generation algorithm $\mathbf{C}$ and $H_{n',n} \xleftarrow{\$} \mathbf{C}(n', n, \mathbb{F}_p)$. A bilinear function $B_{\vec{c}} : (\vec{\alpha}, \vec{\beta}) \to \vec{c} \cdot (\vec{\alpha} \otimes \vec{\beta})^\intercal$, where $\otimes$ denotes the tensor product.

Gen: On input 1^λ:

1. Pick two random size-t subsets (S_0, S_1) of $[n']$, sorted in increasing order.
2. Pick two random vector $(\vec{y}_0, \vec{y}_1) \in (\mathbb{F}_p^t)^2$.
3. Compute $(K_0^{\mathsf{fss}}, K_1^{\mathsf{fss}}) \xleftarrow{\$} \mathsf{MPFSS.Gen}(1^\lambda, f_{S_0 \times S_1, \vec{y}_0 \otimes \vec{y}_1})$.
4. Let $\mathsf{k}_0 \leftarrow (n, K_0^{\mathsf{fss}}, S_0, \vec{y}_0)$ and $\mathsf{k}_1 \leftarrow (n, K_1^{\mathsf{fss}}, S_1, \vec{y}_1)$.
5. Output $(\mathsf{k}_0, \mathsf{k}_1)$.

Expand: On input $(\sigma, \mathsf{k}_\sigma)$, parse k_σ as $(n, K_\sigma^{\mathsf{fss}}, S_\sigma, \vec{y}_\sigma)$. Define $\vec{\mu}_\sigma$ to be $\mathsf{spread}_{n'}(S_\sigma, \vec{y}_\sigma)$ in $\mathbb{F}_p^{n'}$ and $\vec{x}_\sigma \leftarrow \vec{\mu}_\sigma \cdot H_{n',n}$. Compute $\vec{v}_\sigma$ as $\mathsf{MPFSS.FullEval}(\sigma, K_\sigma^{\mathsf{fss}})$ in $\mathbb{F}_p^{(n')^2}$ and set $\vec{z}_\sigma$ to be $-\vec{c} \cdot (\vec{v}_\sigma \cdot (H_{n',n} \otimes H_{n',n}))^\intercal$. Output $(\vec{x}_\sigma, \vec{z}_\sigma)$.

Figure 1.2.1: PCG for Bilinear Correlations

Generalization. The scheme G_{bil} immediately generalizes to a PCG for arbitrary constant-degree polynomials,[9] where the size of the shares grows as $\tilde{O}(t^d)$ and the computational complexity is $\tilde{O}((tn')^d + (nn')^d)$. It allows two parties to locally compute, given the shares, additive shares of $(\vec{r}, P(\vec{r}))$, where $\vec{r}$ is pseudorandom (under LPN) and P is a degree-d multivariate polynomial over $\mathbb{F}$.

To see this, notice that we can replace $\vec{y}_0 \otimes \vec{y}_1$ in **Gen** with $\otimes_d \vec{y} = \vec{y} \otimes \cdots \otimes \vec{y}$, where $\otimes_d \vec{y}$ denotes the tensor product of $\vec{y}$ with itself d times (that is, the list of all degree-d monomials of $\vec{y}$). The parties can then compute shares of all degree-d terms in $P(\vec{r})$ for a random $\vec{r}$; to obtain shares of $\vec{r}$ and the lower-degree terms, we extend the MPFSS values to include $(\vec{y}, \vec{y} \otimes \vec{y}, \cdots, \otimes_{d-1} \vec{y})$ as well as $\otimes_d \vec{y}$.

Corollary 1.2.9. *Suppose the* $(\mathcal{HW}_{t,n'}, \mathbf{C}, \mathbb{F}_p)$-dual-LPN$(n', n)$ *assumption holds,*

[9] In fact, assuming that dual-LPN has $2^{O(t)}$ security (which is in line with the best known attacks), t can be taken as small as $\omega(\log \lambda)$, in which case the degree $d(\lambda)$ of the polynomial can be larger, up to $O(\log \lambda / \log\log \lambda)$. The shares are still of polynomial size $\tilde{O}(t^d)$, although the computational cost $O((n \cdot n')^d)$ is slightly superpolynomial.

and that MPFSS *is a secure multi-point FSS scheme. Then there exists a secure PCG for general constant-degree correlations, with share size* $\tilde{O}(t^d)$ *and computational complexity* $O((n \cdot n')^d)$.

In particular, using $n' = O(n)$, we get:

Corollary 1.2.10. *Assuming the standard LPN assumption over* $\mathbb{F}_p$ *with noise rate* $r = o(n^{1/d-1})$ *and linear number of samples, there exists a PCG for general degree-d polynomials, with sublinear share size (in the output size n) and polynomial computation.*

From PCG to HSS for degree-d polynomials. Given a PCG for general, additive degree-d correlations for a constant d, we show how to construct a homomorphic secret sharing scheme for degree-d multivariate polynomials, a primitive which is interesting in its own right. Consider two parties who wish to compute shares of $P(\vec{x})$ for some public multivariate polynomial P, given shares of $\vec{x}$. Let $\otimes_d \vec{x}$ denote the degree-d tensor product $\vec{x} \otimes \cdots \otimes \vec{x}$. Given a PCG for the additive, degree-d correlation $(\vec{r}, \otimes_2 \vec{r}, \cdots, \otimes_d \vec{r})$, we construct a (secret-key) homomorphic secret sharing scheme $\mathsf{HSS} = (\mathsf{HSS.Gen}, \mathsf{HSS.Share}, \mathsf{HSS.Eval})$ for P as follows.

- $\mathsf{HSS.Share}(\vec{x})$: generate PCG keys $(\mathsf{k}_0, \mathsf{k}_1)$ which expand to shares of $(\vec{r}, \otimes_2 \vec{r}, \cdots, \otimes_d \vec{r})$, set $\vec{x}' \leftarrow \vec{x} + \vec{r}$, and give to each party P_σ a share $s_\sigma = (\mathsf{k}_\sigma, \vec{x}')$.
- $\mathsf{HSS.Eval}(\sigma, s_b, P)$: On input party index $\sigma \in \{0, 1\}$, share s_σ of a size-n input, and a degree-d multivariate polynomial P, compute a share P'_σ of the polynomial P' satisfying $P'(X) = P(X - \vec{r})$. Note that the coefficients of P' are public degree $\leq d$ polynomials in $\vec{r}$, hence shares of the coefficients can be locally computed given shares of the monomials $\vec{r}, \cdots, \otimes_d \vec{r}$. Output $P'_\sigma(\vec{x}')$.

Correctness follows immediately by inspection, and security reduces to the security of the underlying PCG. Therefore, we get:

Corollary 1.2.11. *Suppose that the* $(\mathcal{HW}_{t,n'}, \mathbf{C}, \mathbb{F}_p)$-$\mathsf{dual\text{-}LPN}(n', n)$ *assumption holds, and that* MPFSS *is a secure multi-point FSS scheme. Then there exists a secure HSS for general degree-d multivariate polynomials over* $\mathbb{F}$*, with shares of size* $n + \tilde{O}(t^d)$ *and computational complexity* $O((n \cdot n')^d)$.

1.2.5 HSS for Branching Programs from DDH

We next give a simplified overview of the HSS construction from [21], following exposition from [19]. Cast into the framework of Definition 1.1.4, this yields an

additive public-key $(*, 2)$-δ-HSS for the class of branching programs under the DDH assumption.

For simplicity of notation (and for greater efficiency), we assume circular security of ElGamal encryption. This assumption can be replaced by standard DDH by replacing ElGamal encryption with the circular secure public-key encryption scheme of Boneh, Halevi, Hamburg, and Ostrovsky [12], as shown in [21].

RMS Programs. The construction of [21] supports homomorphic evaluation of straight-line programs of the following form over inputs $w_i \in \mathbb{Z}$, provided that all intermediate computation values in $\mathbb{Z}$ remain "small," bounded by a parameter M (where the required runtime grows with this size bound).

Definition 1.2.12 (RMS programs)**.** The *Restricted Multiplication Straight-line (RMS)* programs consist of a magnitude bound 1^M and an arbitrary sequence of the four following instructions, each with a unique identifier id:

- Load an input into memory: $(\mathsf{id}, \hat{y}_j \leftarrow \hat{w}_i)$.
- Add values in memory: $(\mathsf{id}, \hat{y}_k \leftarrow \hat{y}_i + \hat{y}_j)$.
- Multiply value in memory by an input value: $(\mathsf{id}, \hat{y}_k \leftarrow \hat{w}_i \cdot \hat{y}_j)$.
- Output value from memory, as element of $\mathbb{Z}_\beta$: $(\mathsf{id}, \beta, \hat{O}_j \leftarrow \hat{y}_i)$.

If at any step of execution the size of a memory value exceeds the bound M, the output of the program on the corresponding input is defined to be $\perp$. We define the *size* of an RMS program P as the number of its instructions.

In particular, RMS programs allow only multiplication of a memory value with an *input* (not another memory value). RMS programs with $M = 2$ are powerful enough to efficiently simulate boolean formulas, logarithmic-depth boolean circuits, and deterministic branching programs (capturing logarithmic-space computations). For concrete efficiency purposes, their ability to perform arithmetic computations on larger inputs can also be useful.

Encoding $\mathbb{Z}_q$ Elements. Let $\mathbb{H}$ be a prime-order group, with a subgroup $\mathbb{G}$ of prime order q (the DDH group). Let g denote a generator of $\mathbb{G}$. For any $x \in \mathbb{Z}_q$, consider the following 3 types of two-party encodings:

Level 1: "Encryption." For $x \in \mathbb{Z}_q$, we let $[x]$ denote g^x, and $[\![x]\!]_c$ denote the pair $([r], [r \cdot c + x])$ for a uniformly random $r \in \mathbb{Z}_q$, which corresponds to an ElGamal encryption of x with a secret key $c \in \mathbb{Z}_q$. (With short-exponent ElGamal, c is a 160-bit integer.) We assume that c is represented in base B ($B = 2$ by default)

as a sequence of s digits $(c_i)_{1\le i\le s}$ We let $[\![[x]]\!]_c$ denote $([\![x]\!]_c, ([\![x \cdot c_i]\!]_c)_{1\le i\le s})$. All level-1 encodings are known to both parties.

Level 2: "Additive shares." Let $\langle x\rangle$ denote a pair of shares $x_0, x_1 \in \mathbb{Z}_q$ such that $x_0 = x_1 + x$, where each share is held by a different party. We let $\langle\!\langle x\rangle\!\rangle_c$ denote $(\langle x\rangle, \langle x \cdot c\rangle) \in (\mathbb{Z}_q^2)^2$, namely each party holds one share of $\langle x\rangle$ and one share of $\langle x \cdot c\rangle$. Note that both types of encodings are additively homomorphic over $\mathbb{Z}_q$, namely given encodings of x and x' the parties can locally compute a valid encoding of $x + x'$.

Level 3: "Multiplicative shares." Let $\{x\}$ denote a pair of shares $x_0, x_1 \in \mathbb{G}$ such that the difference between their discrete logarithms is x. That is, $x_0 = x_1 \cdot g^x$.

Operations on Encodings. We manipulate the above encodings via the following two types of operations, performed locally by the two parties:

1. $\mathsf{Pair}([\![x]\!]_c, \langle\!\langle y\rangle\!\rangle_c) \mapsto \{xy\}$. This pairing operation exploits the fact that $[a]$ and $\langle b\rangle$ can be locally converted to $\{ab\}$ via exponentiation.

2. $\mathsf{Convert}(\{z\}, \delta) \mapsto \langle z\rangle$, with failure bound δ. The implementation of Convert is also given an upper bound M on the "payload" z ($M = 1$ by default), and its expected running time grows linearly with M/δ. We omit M from the following notation.

The Convert algorithm works as follows. Each party, on input $h \in \mathbb{G}$, outputs the minimal integer $i \ge 0$ such that $h \cdot g^i$ is "distinguished," where roughly a δ-fraction of the group elements are distinguished. Distinguished elements were picked in [21] by applying a pseudo-random function to the descriptii)on of the group element. An optimized conversion procedure from [23] (using a special choice of "conversion-friendly" choices of $\mathbb{G} \subset \mathbb{Z}_p^*$ and $g = 2$) applies the heuristic of defining a group element to be distinguished if its bit-representation starts with $d \approx \log_2(M/\delta)$ leading 0's; this was further optimized by considering instead the $(d+1)$-bit string $1||0^d$ in [19]. Note that this heuristic only affects the running time and not security, and thus it can be validated empirically. Correctness of Convert holds if no group element *between* the two shares $\{z\} \in \mathbb{G}^2$ is distinguished.

We note that a significantly improved (an optimal) conversion procedure was proposed in [46], using a more involved "Kangaroo walk" method to agree on a distinguished point. The expected running time of of this improved conversion is reduced from M/δ to $\sqrt{M/\delta}$, a quadratic runtime improvement. Finally, Convert can signal that there is a potential failure if there is a distinguished point in the "danger zone." Namely, Party $b = 0$ (resp., $b = 1$) raises a potential error flag $\perp$ if $h \cdot g^{-i}$ (resp., $h \cdot g^{i-1}$) is distinguished for some $i = 1, \ldots, M$.

Note that we used the notation M both for the payload upper bound in $\mathsf{Convert}$ and for the bound on the memory values in the definition of RMS programs (Definition 1.2.12). In the default case of RMS program evaluation using base 2 for the secret key c in level 1 encodings, both values are indeed the same. (However, when using larger basis, they can differ in parts of the computation, and a more careful analysis can improve error bound guarantees.)

Let $\mathsf{PairConv}$ be an algorithm that sequentially executes the two operations Pair and $\mathsf{Convert}$ above: $\mathsf{PairConv}(\llbracket x \rrbracket_c, \langle\!\langle y \rangle\!\rangle_c, \delta) \mapsto \langle xy \rangle$, with error δ. We denote by Mult the following algorithm:

- **Functionality:** $\mathsf{Mult}(\llbracket\llbracket x \rrbracket\rrbracket_c, \langle\!\langle y \rangle\!\rangle_c, \delta) \mapsto \langle\!\langle xy \rangle\!\rangle_c$
  - Parse $\llbracket\llbracket x \rrbracket\rrbracket_c$ as $(\llbracket x \rrbracket_c, (\llbracket x \cdot c_i \rrbracket_c)_{1 \le i \le s})$.
  - Let $\langle xy \rangle \leftarrow \mathsf{PairConv}(\llbracket x \rrbracket, \langle\!\langle y \rangle\!\rangle_c, \delta')$ for $\delta' = \delta/(s+1)$.
  - For $i = 1$ to s, let $\langle xy \cdot c_i \rangle \leftarrow \mathsf{PairConv}(\llbracket xc_i \rrbracket_c, \langle\!\langle y \rangle\!\rangle_c, \delta')$.
  - Let $\langle xy \cdot c \rangle = \sum_{i=1}^{s} B^{i-1} \langle xy \cdot c_i \rangle$.
  - Return $(\langle xy \rangle, \langle xy \cdot c \rangle)$.

HSS for RMS programs. Given the above operations, an additive δ-HSS for RMS programs is obtained as follows. This can be cast as HSS in Definition 1.1.4 with a key generation setup.

- KEY GENERATION: $\mathsf{Gen}(1^\lambda)$ picks a group $\mathbb{G}$ of order q with λ bits of security, generator g, and secret ElGamal key $c \in \mathbb{Z}_q$. It output a public key $\mathsf{pk} = (\mathbb{G}, g, h, \llbracket c_i \rrbracket_c)_{1 \le i \le s}$, where $h = g^c$, and $(\mathsf{ek}_0, \mathsf{ek}_1) \leftarrow \langle c \rangle$, a random additive sharing of c.

- SHARE: $\mathsf{Share}(\mathsf{pk}, x)$ uses the homomorphism of ElGamal to compute and output $\llbracket\llbracket x \rrbracket\rrbracket_c$.

- RMS PROGRAM EVALUATION: For an RMS program P of multiplicative size S, the algorithm $\mathsf{Eval}(b, \mathsf{ek}_b, (\mathsf{ct}_1, \ldots, \mathsf{ct}_n), P, \delta, \beta)$ processes the instructions of P, sorted according to id, as follows. We describe the algorithm for both parties b jointly, maintaining the invariant that whenever a memory variable $\hat{y}$ is assigned a value y, the parties hold level-2 shares $Y = \langle\!\langle y \rangle\!\rangle_c$.
  - $\hat{y}_j \leftarrow \hat{x}_i$: Let $Y_j \leftarrow \mathsf{Mult}(\llbracket\llbracket x_i \rrbracket\rrbracket_c, \langle\!\langle 1 \rangle\!\rangle_c, \delta/S)$, where $\langle\!\langle 1 \rangle\!\rangle_c$ is locally computed from $(\mathsf{ek}_0, \mathsf{ek}_1)$ using $\langle 1 \rangle = (1, 0)$.
  - $\hat{y}_k \leftarrow \hat{y}_i + \hat{y}_j$: Let $Y_k \leftarrow Y_i + Y_j$.
  - $\hat{y}_k \leftarrow \hat{x}_i \cdot \hat{y}_j$: Let $Y_k \leftarrow \mathsf{Mult}(\llbracket\llbracket x_i \rrbracket\rrbracket_c, Y_j, \delta/S)$.

– $(\beta, \hat{O}_j \leftarrow \hat{y}_i)$: Parse Y_i as $(\langle y_i \rangle, \langle y_i \cdot c \rangle)$ and output $O_j = \langle y_i \rangle + (r, r) \bmod \beta$ for a fresh (pseudo-)random $r \in \mathbb{Z}_q$.

The confidence flag is $\perp$ if any of the invocations of Convert raises a potential error flag, otherwise it is $\top$.

The pseudorandomness required for generating the outputs and for Convert is obtained by using a common pseudorandom function key that is (implicitly) given as part of each ek_b, and using a unique nonce as an input to ensure that different invocations of Eval are indistinguishable from being independent.

A single-input ("secret-key") HSS variant is simpler in two ways. First, Share can directly run Gen and generate $[\![\![x]\!]\!]_c$ from the secret key c. Second, an input loading instruction $\hat{y}_j \leftarrow \hat{x}_i$ can be processed directly, without invoking Mult, by letting Share compute $Y_j \leftarrow \langle\!\langle x_i \rangle\!\rangle_c$ and distribute Y_j as shares to the two parties.

Performance. The cost of each RMS multiplication or input loading is dominated by $s + 1$ invocations of PairConv, where each invocation consists of Pair and Convert. The cost of Pair is dominated by one group exponentiation (with roughly 200-bit exponent in [19]). The basis of the exponent depends only on the key and the input, which allows for optimized fixed-basis exponentiations when the same input is involved in many RMS multiplications. When the RMS multiplications apply to 0/1 values (this is the case when evaluating branching programs), the cost of Convert is linear in BS/δ, where the B factor comes from the fact that the payload z of Convert is bounded by the size of the basis. When δ is sufficiently small, the overall cost is dominated by the $O(BS^2 s/\delta)$ "conversion" steps, where each step consists of multiplying by g and testing whether the result is a distinguished group element.

1.2.6 Full-Fledged HSS for Branching Program from DCR

The construction of Section 1.2.5 suffers from two main drawbacks: (1) it is limited to computation on small (polynomially bounded) inputs over $\mathbb{Z}$, and (2) correctness holds only with probability $1 - \delta$, for some inverse polynomial error bound δ. It turns out that these two limitations can be lifted using a variant of the construction over RSA groups – i.e., subgroups of $\mathbb{Z}_n$, where n is a product of two safe primes. We sketch the main ideas underlying this alternative construction below, whose security reduces to the Decisional Composite Residuosity assumption.

Replacing ElGamal by Paillier. The celebrated Paillier encryption scheme [72] is an additively homomorphic cryptosystem over $\mathbb{Z}_n$. Unlike ElGamal, it is not

limited to encrypting small plaintexts. We briefly recall its description: an encryption of a message $m \in \mathbb{Z}_n$ is of the form $c = (1+n)^m \cdot R^n \bmod n^2$, where R is a random element of $\mathbb{Z}_n$. It is easy to check that the scheme is additively homomorphic over $\mathbb{Z}_n$. The decryption key is an integer $d \in \mathbb{Z}$ such that $d = 1 \bmod n$ and $d = 0 \bmod \phi(n)$, where ϕ is Euler's totient function. The decryption procedure first computes $c^d \bmod n^2$, which is equal to $(1+n)^m \bmod n^2$. Then, since $(1+n)^m = 1 + n \cdot m \bmod n^2$, m can be recovered by substracting one and dividing by n over $\mathbb{Z}$.

The "level 1" type encodings are again of the form $(\llbracket x \rrbracket_d, (\llbracket x \cdot d_i \rrbracket_d)_{1 \le i \le s})$. However, $\llbracket \cdot \rrbracket$ now denotes Paillier encryption. Furthermore, x can now be exponentially large, e.g. $0 \le w \le n^{1/4}$. Similarly, the d_i are "chunks" of the secret key d, of length (say) roughly $n/4$ bits each. Since $|d| \approx 2n$, this allows to set s to be a small integer (here, 8) instead of a security parameter in the previous construction. Level 2 encodings are of the form $(\langle x \rangle, \langle x \cdot d \rangle)$, i.e., additive shares of x and $x \cdot d$ over the integers.

Errorless discrete logarithm. As before, the parties with level 1 shares of x and level 2 shares of an input y will compute partial decryptions, and recover multiplicative (i.e., level 3) shares of $(1+n)^{xy} \bmod n^2$ (as well as of $(1+n)^{d_i \cdot xy} \bmod n^2$ for $i = 1$ to 8). However, the distributed discrete logarithm procedure, which converts level 3 shares back to level 2 shares, is fundamentally different and errs only with negligible probability. Let (h, h') denote the two multiplicative shares of $(1+n)^{xy} \bmod n^2$. Each party write its share in base-n: $h = h_0 + n \cdot h_1 \bmod n^2$, and $h' = h'_0 + n \cdot h'_1 \bmod n^2$.

We have $(1+n)^{xy} = h_0 h'_0 + n \cdot (h_0 h'_1 + h'_0 h_1) \bmod n^2$. The key observation is that $h_0 \cdot h'_0 = (h \bmod n) \cdot (h' \bmod n) = hh' = 1 \bmod n$. Therefore, since $h_0 < n$ and $h'_0 < n$, h'_0 is equal to $[h_0^{-1} \bmod n]$ (over the integers), and h_0 is equal to $[(h'_0)^{-1} \bmod n]$. Since furthermore $(1+n)^{xy} = 1 + nxy \bmod n^2$, it holds that

$$\begin{aligned} xy &= h_0 h'_1 + h'_0 h_1 + \div_n(h_0 h'_0) \bmod n \\ &= [h_0^{-1} \bmod n] \cdot h_1 + \div_n(h_0 \cdot [h_0^{-1} \bmod n]) + [(h'_0)^{-1} \bmod n] \cdot h'_1 \bmod n \\ &= z + z' \bmod n, \end{aligned}$$

where $\div_n$ denotes the euclidean division by n and where we let $z \leftarrow [h_0^{-1} \bmod n] \cdot h_1$ and $z' \leftarrow \div_n(h_0 \cdot [h_0^{-1} \bmod n]) + [(h'_0)^{-1} \bmod n] \cdot h'_1$. Observe that z and z' can be *locally* computed, each by one of the parties. At this stage, we are almost done: each party locally computes its additive share (z or z') of $xy \bmod n$. With the same procedure, the parties also obtain additive shares of $d_i \cdot xy \bmod n$. It remains to convert these shares over $\mathbb{Z}_n$ into shares over $\mathbb{Z}$. Here, the core observation stems from the work of [26] (in the context of building HSS from the LWE assumption):

since xy and the $xy \cdot d_i$ are exponentially smaller than the modulus n, it suffices to *do nothing*. Indeed, given shares (z, z') over $\mathbb{Z}_n$ of xy, it holds that $(z, z' - n)$ form shares of xy over $\mathbb{Z}$, except when $z < xy$ (in which case $z' = z - xy$ and $(z, z' - n)$ are shares of $xy - n$ instead). Since $xy < n^{1/2}$ and that z is uniformly distributed over $\mathbb{Z}_n$, the probability that this happens is at most $1/\sqrt{n}$, which is negligible. The same observation shows that the shares of $xy \cdot d_i$ can be converted into shares over $\mathbb{Z}$ with error at most $1/n^{1/4}$. From the shares of $xy \cdot d_i$, the parties can reconstruct $\langle xy \rangle, \langle xy \cdot d \rangle$.

Evaluating RMS programs. The construction of HSS for RMS programs proceeds identically to the construction based on DDH. Because level 1 encodings contain encryptions of functions of the secret key d, the security relies on the assumption that the Paillier cryptosystem remains secure when encrypting its secret key, leading to a circular-security variant of the DCR assumption. As for the ElGamal-based construction, one can base the security solely on the plain DCR assumption by replacing Paillier by a circularly-secure variant [27]. With this replacement, one gets an errorless DCR-based HSS scheme for evaluating arbitrary RMS programs on shared inputs, where the RMS program can directly handle exponentially large elements of $\mathbb{Z}$ (up to some fixed bound, here $n^{1/4}$).

Historical notes. The construction sketched above was presented in [69]. The original presentation is more abstract, we provide here a more direct and algorithmic description for simplicity. In addition to following the blueprint of the DDH-based construction from [21] (see Section 1.2.5), the construction of [69] also borrows ideas from a previous construction of HSS from LWE from [26]. A very similar construction, also realizing HSS for branching programs from DCR, was proposed in a concurrent and independent work [76].

1.3 Applications and Implications

In this section, we turn to implications of FSS and HSS constructions. We begin by describing what is known about the relation of FSS/HSS to other primitives, and then address applications of "low-end", "mid-range", and "high-end" construction regimes.

1.3.1 Relation to Other Primitives

Below are the primary known theoretical implications of FSS/HSS primitives.

One-way functions. FSS for any "sufficiently rich" function class $\mathcal{F}$ (e.g., point functions) necessitates the existence of OWF [57]. Further, in such an FSS, each output share f_i viewed as a function on its own must define a pseudorandom function [20]. Note that this is not a-priori clear from the security definition, which only requires that the shares hide f.

(Amortized) Low-communication secure computation. It was shown in [20] that FSS for a function class $\mathcal{F}$ strictly containing the decryption circuit for a secure symmetric-key encryption scheme implies amortized low-communication protocols for secure two-party computation of a related function class, relying on a reusable source of correlated randomness (that can be realized via one-time offline preprocessing). Given HSS for $\mathcal{F}$, the same result holds without needing to amortize over the preprocessing.[10]

At the time of this result, all known approaches for obtaining such protocols relied on fully homomorphic encryption or related primitives, and as such this was viewed as a "barrier" against achieving such FSS without FHE. In an interesting twist, this was reversed by the work of [21], which succeeded in constructing a form of HSS for NC^1 (and thus succinct secure computation) from DDH. However, the "barrier" still seems legitimate as evidence against the possibility of constructing general FSS/HSS (or even classes such as NC^1 or possibly AC^0) from weak cryptographic assumptions such as the existence of one-way functions or oblivious transfer.

Non-interactive key exchange (NIKE) & 2-message oblivious transfer (OT). The power of additive *multi-input* HSS (where inputs from different parties can be homomorphically computed on together; c.f. Definition 1.1.4) seems to be much greater than its single-input counterpart. Whereas constructions for single-input HSS exist for some function classes from OWF, to date *all* constructions of multi-input HSS rely on a select list of heavily structured assumptions: DDH, LWE, and obfuscation [21, 49].

It appears this is in some sense inherent: As shown in [25], even a minimal version of 2-party, 2-server additive HSS for the AND of two input bits implies the existence of non-interactive key exchange (NIKE) [44], a well-studied cryptographic notion whose known constructions are similarly limited to select structured assumptions. NIKE is black-box separated from one-way functions and highly unlikely to be implied by generic public-key encryption or oblivious transfer. On the other hand, this same type of $(2, 2)$-additive-HSS for AND is unlikely to be implied *by* NIKE, as the primitive additionally implies the existence of 2-message oblivious transfer (OT) [21], unknown to follow from NIKE alone. Further connections from HSS to 2-round secure computation have been demonstrated in [23, 25].

[10] Recall in HSS the secret share size scales with input size and not function description size.

Worst-case to average-case reductions. A different type of implication of HSS is in obtaining worst-case to average-case reductions in P. Roughly speaking, the HSS evaluation function Eval for homomorphically evaluating a function F defines a new function F' such that computing F on any given input x can be reduced to computing F' on two or more inputs that are individually pseudorandom (corresponding to the HSS secret shares of x). A similar application was pointed out in [37] using fully homomorphic encryption (FHE) (and a significantly weaker version in [57] using DPF). Compared to the FHE-based reductions, the use of HSS has the advantages of making only a constant number of queries to a *Boolean* function F' (as small as 2), and minimizing the complexity of recovering the output from the answers to the queries. The latter can lead to efficiency advantages in the context of applications (including the settings of fine-grained average-case hardness and verifiable computation; see [25]). It also gives rise to worst-case to average-case reductions under assumptions that are not known to imply FHE, such as the DDH assumption.

1.3.2 Applications in the One-Way Function Regime

FSS in the "low-end" regime has interesting applications to efficient private manipulation of remotely held databases, extending the notions of Private Information Retrieval (PIR) [36] and Private Information Storage [70] to more expressive instruction sets. Recently, FSS has also been shown to yield concrete efficiency improvements in secure 2-party computation protocols for programs with data-dependent memory accesses. We describe these in greater detail below.

Multi-server PIR and secure keyword search. Suppose that each of m servers holds a database D of keywords $w_j \in \{0,1\}^n$. A client wants to count the number of occurrences of a given keyword w without revealing w to any strict subset of the servers. Letting $\mathbb{G} = \mathbb{Z}_{m+1}$ and $f = f_{w,1}$ (the point function evaluating to 1 on target value w), the client can split f into m additive shares and send to server i the key k_i describing f_i. Server i computes and sends back to the client $\sum_{w_j \in D} f_i(w_j)$. The client can then find the number of matches by adding the m group elements received from the servers. Standard PIR corresponds to the same framework with point function $f_{i,1}$ for target data index i. In this application, FSS for other classes $\mathcal{F}$ can be used to accommodate richer types of search queries, such as counting the number of keywords that lie in an interval, satisfy a fuzzy match criterion, etc. We note that by using standard randomized sketching techniques, one can obtain similar solutions that do not only count the number of matches but also return the payloads associated with a bounded number of matches (see, e.g., [71]).

Splinter [81]. In this fashion, FSS for point functions and intervals are the core

of the system Splinter [81], serving private search queries on a Yelp clone of restaurant reviews, airline ticket search, and map routing. On top of the functionalities offered directly by the FSS, the system supports more expressive queries, such as MAX/MIN and TOP-k, by manipulating the database on the server side such that a point function / interval search on the modified database answers the desired query. (Here the type of query is revealed, but the search parameters are hidden.) Splinter reports end-to-end latencies below 1.6 seconds for realistic workloads, including search within a Yelp-like database comparable to 40 cities, and routing within real traffic-map data for New York City.

Incremental secret sharing. Suppose that we want to collect statistics about web usage of mobile devices without compromising the privacy of individual users, and while allowing fast collection of real-time aggregate usage data. A natural solution is to maintain a large secret-shared array of group elements between m servers, where each entry in the array is initialized to 0 and is incremented whenever the corresponding web site is visited. A client who visits URL u can now secret-share the point function $f = f_{u,1}$, and each server i updates its shared entry of each URL u_j by locally adding $f_i(u_j)$ to this share. The end result is that only position u_j in the shared array is incremented, while no collusions involving strict subsets of servers learn which entry was incremented. Here too, applying general FSS can allow for more general "attribute-based" writing patterns, such as secretly incrementing all entries whose public attributes satisfy some secret predicate. The above incremental secret sharing primitive can be used to obtain low-communication solutions to the problem of private information storage [70], the "writing" analogue of PIR.

Riposte [38]. FSS for point functions on a 2^{20}-entry database are used in this way in the anonymous broadcast system Riposte of Corrigan-Gibbs *et al.* [38]. Roughly, in the system each user splits his message msg as a point function $f_{r,\mathsf{msg}}$ for a random position index $r \in [2^{20}]$. Shares of such functions across many users are combined additively by each server, and ultimately the *aggregate* is revealed. FSS security guarantees that the link from each individual user to his contributed message remains hidden. An improved variant of Riposte, called Spectrum, was recently presented in [68].

Protecting against malicious clients. In some applications, malicious clients may have incentive to submit bogus FSS shares to the servers, corresponding to illegal manipulations of the database. This can have particularly adverse effects in writing applications, e.g., casting a "heavy" vote in a private poll, or destroying the current set of anonymous broadcast messages. Because of this, it is desirable to have efficient targeted protocols that enable a client to prove the validity of his request before it is implemented, via minimal interaction between the client and

servers. Such protocols have been designed for certain forms of DPFs and related primitives in [38, 22, 11].

Secure 2-party computation (2PC) of RAM programs. A standard challenge in designing secure computation protocols is efficiently supporting *data-dependent* memory accesses, without leaking information on which items were accessed (and in turn on secret input values). Since the work of [70], this is typically addressed using techniques of *Oblivious RAM* (ORAM) [61] to transform a memory access to a secret index i from data size N into a sequence of $\mathsf{polylog}(N)$ memory accesses whose indices appear independent of i. Indeed, a recent line of works have implemented and optimized systems for ORAM in secure computation.

Floram [79]. In a surprising development, Doerner and shelat [79] demonstrated an *FSS-based* 2PC system that—despite its inherent poor $O(N)$ asymptotic computation per private access of each secret index i (instead of $\mathsf{polylog}(N)$)—concretely outperforms current ORAM-based solutions. In their construction, similar to use of ORAM in 2PC, the two parties in the secure computation act as the two servers in the FSS scheme, and an underlying secure computation between the parties emulates the role of the client. The core savings of their approach is that, while the overall computation is high, the emulation of "client" operations in the FSS requires a very small secure computation in comparison to prior ORAM designs (up to one hundred times smaller for the memory sizes they explore). A key technical ingredient of Floram, that was used in subsequent works on PCGs, is a concretely efficient protocol for distributing the key generation of a DPF in which the communication cost is comparable to the key size and the computation cost to the input domain size. Their implemented 2PC system Floram [79] ("**F**SS **L**inear **ORAM**") outperforms the fastest previously known ORAM implementations, Circuit ORAM [82] and Square-root ORAM [88], for datasets that are 32 KiB or larger, and outperforms prior work on applications such as secure stable matching [50] or binary search [62] by factors of two to ten.

Distributed ORAM. Following the construction of Doerner and shelat, distributed ORAM protocol were described in the 2-party [63] and 3-party [30] settings. The construction of [30] was very recently improved in [31].

Mixed-mode secure computation. Secure computation protocols typically operate over secret-shared values and employ atomic protocols to convert shares of inputs into shares of the outputs of some component of the computation (e.g. a product gate). In [24], a new approach was developed to allow the atomic computation of more complex gates, with a minimal online communication, following an input-independent preprocessing. This approach handles any gate that admits an efficient FSS scheme, and achieves exponential improvement over previous methods for many important types of gates, such as equality tests or comparison

predicates. This provides a unifying framework for secure computation with correlated randomness of functions that involve various interconnected, possibly complex components. The approach was further generalized and optimized in [13]. An application of this method to privacy-preserving deep learning was developed in [77].

Private heavy hitters. Private heavy hitters allow multiple clients, each holding a bitstring, to securely recover the set of all popular bitstrings, by interacting with a small number of data-collection servers. The problem of privately realizing this functionality comes up in a variety of real-world data-collection applications. A solution to the problem with two servers using function secret sharing was recently described in [11].

Private set intersection and contact tracing. A private set intersection (PSI) protocol allows two parties to compute the intersection of their respective sets, without leaking elements outside of the intersection. PSI protocols have been the subject of an intense research effort, and can be used for contact recovery, and in a variety of medical applications. The latest, most optimized PSI protocols [73] employ the FSS-based PCG for VOLE of [14]. FSS-based constructions of PSI tailored to contact tracing applications were also recently described in [80, 47].

1.3.3 Applications in the LPN and Public-Key Regime

In the "high-end" regime, HSS can serve as a competitive alternative to FHE in certain application settings. Fully homomorphic encryption (FHE) [75, 54] is commonly viewed as a "dream tool" in cryptography, enabling one to perform arbitrary computations on encrypted inputs. For example, in the context of secure multiparty computation (MPC) [87, 60, 8, 33], FHE can be used to minimize the communication complexity and the round complexity, and shift the bulk of the computational work to any subset of the participants. However, despite exciting progress in the past years, even the most recent implementations of FHE [64, 51, 34] are still quite slow and require large ciphertexts and keys. This is due in part to the limited set of assumptions on which FHE constructions can be based [45, 29, 55], which are all related to lattices and are therefore susceptible to lattice reduction attacks. As a result, it is arguably hard to find realistic application scenarios in which current FHE implementations outperform optimized versions of classical secure computation techniques (such as garbled circuits) when taking both communication and computation costs into account.

A main motivating observation is that unlike standard FHE, HSS can be useful even for *small computations* that involve short inputs, and even in application scenarios in which competing approaches based on traditional secure computation

techniques do not apply at all.

As with FHE, HSS enables secure computation protocols that simultaneously offer a minimal amount of interaction and collusion resistance. However, the *optimal output compactness* of HSS makes it the only available option for applications that involve computing long outputs (or many short outputs) from short secret inputs (possibly along with public inputs). More generally, this feature enables applications in which the communication and computation costs of output reconstruction need to be minimized, e.g., for the purpose of reducing power consumption. For instance, a mobile client may wish to get quickly notified about live news items that satisfy certain secret search criteria, receiving a fast real-time feed that reveals only pointers to matching items. Further advantages of group-based HSS over existing FHE implementations include smaller keys and ciphertexts and a lower startup cost.

Low-communication 2PC using LPN-based HSS. Following a template established in [21], any homomorphic secret sharing scheme for a class C gives rise to a low-communication secure computation protocol for the same class. Then, when larger circuits can be divided into consecutive "chunks" where each chunk is a circuit from the smaller class C, the result generalized to low-communication protocols for this larger class (albeit with much smaller asymptotic savings). An interesting consequence of this connection is that it allows expanding the set of assumptions under which secure computation is feasible with sublinear communication – something that was long restricted to the assumptions that imply fully-homomorphic encryption. We summarize below the results obtained by instantiating this approach with the HSS construction from Section 1.2.4.

Corollary 1.3.1. *Suppose the* $(\mathcal{HW}_{t,n'}, \mathbf{C}, \mathbb{F}_p)$-dual-LPN$(n', n)$ *assumption holds, and that* MPFSS *is a secure multi-point FSS scheme. Then there exists a 2-party secure computation protocol with semi-honest security for general degree-d multivariate polynomials over* $\mathbb{F}$*, with communication* $\tilde{O}(n + t^d)$ *and computational complexity* $O((n \cdot n')^d)$.

In particular, applying the above corollary to layered circuits, we obtain a generic secure two-party protocol from LPN with communication smaller than the circuit size:

Corollary 1.3.2. *Suppose the* $(\mathcal{HW}_{t,n'}, \mathbf{C}, \mathbb{F}_p)$-dual-LPN$(n', n)$ *assumption holds, and that* MPFSS *is a secure multi-point FSS scheme. Then for any constant c, there exists a 2-party secure computation protocol with semi-honest security for arbitrary layered circuits of size s, with total communication bounded by* s/c*, and computational complexity bounded by* $s \cdot \lambda^{2^{O(c)}}$.

Using a more intricate construction, Couteau and Meyer [39] have extended the above result to show that assuming the *superpolynomial* hardness of LPN, there exists an HSS scheme for the class of log log-depth circuits. Applying this HSS scheme to layered circuits leads to *sublinear communication* two-party secure computation protocols (with total communication bounded by $O(s/\log\log s)$).

Low-communication 2PC using DDH-based HSS. Instantiating the approach of Boyle et al. [21] using the DDH-based HSS scheme of Section 1.2.5 instead, we get the following corollary:

Corollary 1.3.3. *Suppose that the DDH assumption holds. Then there exists a 2-party secure computation protocol with semi-honest security for arbitrary layered circuits of size s, with total communication bounded by $O(s/\log s)$, and computational complexity bounded by $O(s^{1+\epsilon})$, for an arbitrary constant $\epsilon > 0$.*

The above follows from the fact that log-depth circuits can be computed by polynomial-size RMS programs. Since the DDH-based HSS only enjoys approximate correctness, some additional machinery is required to achieve this result; in particular, one must use appropriate error-correcting codes computable by RMS programs to encode the output of each HSS computation, and correct the errors before going on. We omit the details in this high level presentation. Of course, the same result can be established under the DCR assumption using the HSS scheme of [69, 76], with a significantly simpler analysis since no error correction is required.

Secure MPC with minimal interaction. Using multi-input HSS, a set of clients can outsource a secure computation to two non-colluding servers by using the following minimal interaction pattern: each client independently sends a single message to the servers (based on its own input and the public key), and then each server sends a single message to each client. Alternatively, servers can just publish shares of the output if the output is to be made public. The resulting protocol is resilient to any (semi-honest) collusion between one server and a subset of the clients, and minimizes the amount of work performed by the clients. It is particularly attractive in the case where many "simple" computations are performed on the same inputs. In this case, each additional instance of secure computation involves just local computation by the servers, followed by a minimal amount of communication and work by the clients.

Secure data access. HSS yields several different applications in the context of secure access to distributed data. For example, HSS can be used to construct a 2-server variant of attribute based encryption, in which each client can access an encrypted file only if its (public or encrypted) attributes satisfy an encrypted policy set up by the data owner. Other sample applications include 2-server private

RSS feeds, in which clients can receive succinct notifications about new data that satisfies their encrypted matching criteria, and 2-server PIR schemes with general boolean queries. These applications benefit from the optimal output compactness feature of HSS discussed above, minimizing the communication from servers to clients and the computation required for reconstructing the output.

Unlike competing solutions based on classical secure computation techniques, HSS-based solutions only involve minimal interaction between clients and servers and no direct interaction between servers. In fact, for the RSS feed and PIR applications, the client is free to choose an arbitrary pair of servers who have access to the data being privately searched. These servers do not need to be aware of each other's identity, and do not even need to know they are participating in an HSS-based cryptographic protocol: each server can simply run the code provided by the client on the (relevant portion of) the data, and return the output directly to the client.

Pseudorandom correlation generators and secure computation with silent preprocessing. As shown in this chapter, function secret sharing is closely related to the notion of PCGs. In turn, PCGs enable local generation of large amounts of correlated pseudorandomness that can be used to speed up classical protocols for secure two-party computation. Concretely, following a small-communication setup phase, the parties can *locally* expand these shares (without any communication) into useful forms of correlated randomness. This approach leads to the notion of secure computation with *silent preprocessing*, which was put forth in [16, 15].

PCGs were informally introduced in [19]. Efficient constructions were subsequently developed in [14, 16, 18]. These constructions led to the development of increasingly efficient silent secure computation protocols in [15, 78, 86, 83], culminating with the recent work of [40]. Pseudorandom correlation generators were generalized to pseudorandom correlation *functions* in [17], enabling an unbounded generation of correlated randomness from a one-time short interaction.

PCGs and zero-knowledge proofs. PCGs have been shown to be a key primitive for the construction of extremely efficient zero-knowledge proof systems, leading to an impressive sequence of works [14, 83, 5, 48, 84, 85, 4].

Secure computation with minimal interaction. We conclude this chapter by mentioning the application of HSS and PCGs to minimizing interaction in secure computation. The work of [2] showed how HSS can be used to construct general two-round protocols for secure multiparty computation in which the messages of the first round are reusable. Similar protocols were previously known only under the LWE assumption. This result was later shown to hold also under the LPN assumption in [3].

Bibliography

[1] Alekhnovich, M.: More on average case vs approximation complexity. In: 44th Symposium on Foundations of Computer Science (FOCS 2003), 11-14 October 2003, Cambridge, MA, USA, Proceedings. pp. 298–307. IEEE Computer Society (2003), https://doi.org/10.1109/SFCS.2003.1238204

[2] Bartusek, J., Garg, S., Masny, D., Mukherjee, P.: Reusable two-round mpc from ddh. In: Theory of Cryptography Conference. pp. 320–348. Springer (2020)

[3] Bartusek, J., Garg, S., Srinivasan, A., Zhang, Y.: Reusable two-round mpc from lpn. IACR Cryptol. ePrint Arch. 2021, 316 (2021)

[4] Baum, C., Braun, L., Munch-Hansen, A., Scholl, P.: Appenzeller to brie: Efficient zero-knowledge proofs for mixed-mode arithmetic and z2k (2021)

[5] Baum, C., Malozemoff, A.J., Rosen, M.B., Scholl, P.: Mac'n'cheese: Zero-knowledge proofs for boolean and arithmetic circuits with nested disjunctions. In: Annual International Cryptology Conference. pp. 92–122. Springer (2021)

[6] Beimel, A., Burmester, M., Desmedt, Y., Kushilevitz, E.: Computing functions of a shared secret. SIAM J. Discrete Math. 13(3), 324–345 (2000)

[7] Beimel, A., Ishai, Y., Kushilevitz, E., Orlov, I.: Share conversion and private information retrieval. In: CCC 2012. pp. 258–268 (2012)

[8] Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In: STOC. pp. 1–10 (1988)

[9] Benaloh, J.C.: Secret sharing homomorphisms: Keeping shares of a secret secret. In: CRYPTO. pp. 251–260 (1986)

[10] Blum, A., Furst, M.L., Kearns, M.J., Lipton, R.J.: Cryptographic primitives based on hard learning problems. In: Advances in Cryptology - CRYPTO '93, 13th Annual International Cryptology Conference, Santa Barbara, California, USA, August 22-26, 1993, Proceedings. pp. 278–291 (1993), https://doi.org/10.1007/3-540-48329-2_24

[11] Boneh, D., Boyle, E., Corrigan-Gibbs, H., Gilboa, N., Ishai, Y.: Lightweight techniques for private heavy hitters. In: 2021 IEEE Symposium on Security and Privacy (SP). pp. 762–776. IEEE (2021)

[12] Boneh, D., Halevi, S., Hamburg, M., Ostrovsky, R.: Circular-secure encryption from decision diffie-hellman. In: Advances in Cryptology - CRYPTO. pp. 108–125 (2008)

[13] Boyle, E., Chandran, N., Gilboa, N., Gupta, D., Ishai, Y., Kumar, N., Rathee, M.: Function secret sharing for mixed-mode and fixed-point secure computation. In: Canteaut, A., Standaert, F.X. (eds.) EUROCRYPT 2021, Part II. LNCS, vol. 12697, pp. 871–900. Springer, Heidelberg, Germany, Zagreb, Croatia (Oct 17–21, 2021)

[14] Boyle, E., Couteau, G., Gilboa, N., Ishai, Y.: Compressing vector OLE. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) ACM CCS 2018. pp. 896–912. ACM Press, Toronto, ON, Canada (Oct 15–19, 2018)

[15] Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Rindal, P., Scholl, P.: Efficient two-round OT extension and silent non-interactive secure computation. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019. pp. 291–308. ACM Press (Nov 11–15, 2019)

[16] Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Scholl, P.: Efficient pseudorandom correlation generators: Silent OT extension and more. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019, Part III. LNCS, vol. 11694, pp. 489–518. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 18–22, 2019)

[17] Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Scholl, P.: Correlated pseudorandom functions from variable-density LPN. In: 61st FOCS. pp. 1069–1080. IEEE Computer Society Press, Durham, NC, USA (Nov 16–19, 2020)

[18] Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Scholl, P.: Efficient pseudorandom correlation generators from ring-LPN. In: Micciancio,

D., Ristenpart, T. (eds.) CRYPTO 2020, Part II. LNCS, vol. 12171, pp. 387–416. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 17–21, 2020)

[19] Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Orrù, M.: Homomorphic secret sharing: Optimizations and applications. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017. pp. 2105–2122. ACM Press, Dallas, TX, USA (Oct 31 – Nov 2, 2017)

[20] Boyle, E., Gilboa, N., Ishai, Y.: Function secret sharing. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part II. LNCS, vol. 9057, pp. 337–367. Springer, Heidelberg, Germany, Sofia, Bulgaria (Apr 26–30, 2015)

[21] Boyle, E., Gilboa, N., Ishai, Y.: Breaking the circuit size barrier for secure computation under DDH. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016, Part I. LNCS, vol. 9814, pp. 509–539. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 14–18, 2016)

[22] Boyle, E., Gilboa, N., Ishai, Y.: Function secret sharing: Improvements and extensions. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 2016. pp. 1292–1303. ACM Press, Vienna, Austria (Oct 24–28, 2016)

[23] Boyle, E., Gilboa, N., Ishai, Y.: Group-based secure computation: Optimizing rounds, communication, and computation. In: Coron, J.S., Nielsen, J.B. (eds.) EUROCRYPT 2017, Part II. LNCS, vol. 10211, pp. 163–193. Springer, Heidelberg, Germany, Paris, France (Apr 30 – May 4, 2017)

[24] Boyle, E., Gilboa, N., Ishai, Y.: Secure computation with preprocessing via function secret sharing. In: Hofheinz, D., Rosen, A. (eds.) TCC 2019, Part I. LNCS, vol. 11891, pp. 341–371. Springer, Heidelberg, Germany, Nuremberg, Germany (Dec 1–5, 2019)

[25] Boyle, E., Gilboa, N., Ishai, Y., Lin, H., Tessaro, S.: Foundations of homomorphic secret sharing. In: Karlin, A.R. (ed.) ITCS 2018. vol. 94, pp. 21:1–21:21. LIPIcs, Cambridge, MA, USA (Jan 11–14, 2018)

[26] Boyle, E., Kohl, L., Scholl, P.: Homomorphic secret sharing from lattices without FHE. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019, Part II. LNCS, vol. 11477, pp. 3–33. Springer, Heidelberg, Germany, Darmstadt, Germany (May 19–23, 2019)

[27] Brakerski, Z., Goldwasser, S.: Circular and leakage resilient public-key encryption under subgroup indistinguishability - (or: Quadratic residuosity strikes back). In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 1–20. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 15–19, 2010)

[28] Brakerski, Z., Lyubashevsky, V., Vaikuntanathan, V., Wichs, D.: Worst-case hardness for LPN and cryptographic hashing via code smoothing. In: Ishai, Y., Rijmen, V. (eds.) Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part III. Lecture Notes in Computer Science, vol. 11478, pp. 619–635. Springer (2019), https://doi.org/10.1007/978-3-030-17659-4_21

[29] Brakerski, Z., Vaikuntanathan, V.: Efficient fully homomorphic encryption from (standard) LWE. SIAM J. Comput. 43(2), 831–871 (2014)

[30] Bunn, P., Katz, J., Kushilevitz, E., Ostrovsky, R.: Efficient 3-party distributed ORAM. In: Galdi, C., Kolesnikov, V. (eds.) SCN 20. LNCS, vol. 12238, pp. 215–232. Springer, Heidelberg, Germany, Amalfi, Italy (Sep 14–16, 2020)

[31] Bunn, P., Kushilevitz, E., Ostrovsky, R.: CNF-FSS and its applications. IACR Cryptol. ePrint Arch. 2021, 163 (2021)

[32] Catalano, D., Fiore, D.: Using linearly-homomorphic encryption to evaluate degree-2 functions on encrypted data. In: Ray, I., Li, N., Kruegel, C. (eds.) Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015. pp. 1518–1529. ACM (2015), https://doi.org/10.1145/2810103.2813624

[33] Chaum, D., Crépeau, C., Damgård, I.: Multiparty unconditionally secure protocols (extended abstract). In: STOC. pp. 11–19 (1988)

[34] Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In: Asiacrypt'16. pp. 3–33 (2016)

[35] Chor, B., Gilboa, N.: Computationally private information retrieval (extended abstract). In: Proceedings of 29th Annual ACM Symposium on the Theory of Computing. pp. 304–313 (1997)

[36] Chor, B., Goldreich, O., Kushilevitz, E., Sudan, M.: Private information retrieval. J. ACM 45(6), 965–981 (1998)

[37] Chung, K.M., Kalai, Y., Vadhan, S.P.: Improved delegation of computation using fully homomorphic encryption. In: CRYPTO (2010)

[38] Corrigan-Gibbs, H., Boneh, D., Mazières, D.: Riposte: An anonymous messaging system handling millions of users. In: IEEE Symposium on Security and Privacy, SP. pp. 321–338 (2015)

[39] Couteau, G., Meyer, P.: Breaking the circuit size barrier for secure computation under quasi-polynomial LPN. In: Canteaut, A., Standaert, F.X. (eds.) EUROCRYPT 2021, Part II. LNCS, vol. 12697, pp. 842–870. Springer, Heidelberg, Germany, Zagreb, Croatia (Oct 17–21, 2021)

[40] Couteau, G., Rindal, P., Raghuraman, S.: Silver: Silent vole and oblivious transfer from hardness of decoding structured ldpc codes. In: Annual International Cryptology Conference. pp. 502–534. Springer (2021)

[41] Cramer, R., Damgård, I., Ishai, Y.: Share conversion, pseudorandom secret-sharing and applications to secure computation. In: Kilian, J. (ed.) Theory of Cryptography, Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005, Proceedings. Lecture Notes in Computer Science, vol. 3378, pp. 342–362. Springer (2005), https://doi.org/10.1007/978-3-540-30576-7_19

[42] De Santis, A., Desmedt, Y., Frankel, Y., Yung, M.: How to share a function securely. In: Proceedings of 26th Annual ACM Symposium on Theory of Computing. pp. 522–533 (1994)

[43] Desmedt, Y., Frankel, Y.: Threshold cryptosystems. In: Advances in Cryptology - CRYPTO. pp. 307–315 (1989)

[44] Diffie, W., Hellman, M.: New directions in cryptography. IEEE Transactions on Information Theory 22(6), 644–654 (1976)

[45] van Dijk, M., Gentry, C., Halevi, S., Vaikuntanathan, V.: Fully homomorphic encryption over the integers. In: EUROCRYPT. pp. 24–43 (2010)

[46] Dinur, I., Keller, N., Klein, O.: An optimal distributed discrete log protocol with applications to homomorphic secret sharing. In: Shacham, H., Boldyreva, A. (eds.) Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part III. Lecture Notes in Computer Science, vol. 10993, pp. 213–242. Springer (2018), https://doi.org/10.1007/978-3-319-96878-0_8

[47] Dittmer, S., Ishai, Y., Lu, S., Ostrovsky, R., Elsabagh, M., Kiourtis, N., Schulte, B., Stavrou, A.: Function secret sharing for psi-ca: With applications to private contact tracing. arXiv preprint arXiv:2012.13053 (2020)

[48] Dittmer, S., Ishai, Y., Ostrovsky, R.: Line-point zero knowledge and its applications. In: 2nd Conference on Information-Theoretic Cryptography (ITC 2021). Schloss Dagstuhl-Leibniz-Zentrum für Informatik (2021)

[49] Dodis, Y., Halevi, S., Rothblum, R.D., Wichs, D.: Spooky encryption and its applications. In: CRYPTO 2016. pp. 93–122 (2016)

[50] Doerner, J., Evans, D., Shelat, A.: Secure stable matching at scale. In: ACM SIGSAC CCS. pp. 1602–1613 (2016)

[51] Ducas, L., Micciancio, D.: FHEW: bootstrapping homomorphic encryption in less than a second. In: EUROCRYPT. pp. 617–640 (2015)

[52] Fazio, N., Gennaro, R., Jafarikhah, T., Skeith, W.E.: Homomorphic secret sharing from paillier encryption. In: ProvSec. pp. 381–399 (2017)

[53] Fosli, I., Ishai, Y., Kolobov, V.I., Wootters, M.: On the download rate of homomorphic secret sharing. In: ITCS 2022 (2022), full version: Cryptology ePrint Archive, Report 2021/1532.

[54] Gentry, C.: Fully homomorphic encryption using ideal lattices. In: STOC. pp. 169–178 (2009)

[55] Gentry, C., Sahai, A., Waters, B.: Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In: Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I. pp. 75–92 (2013), http://dx.doi.org/10.1007/978-3-642-40041-4_5

[56] Gilboa, N., Ishai, Y.: Compressing cryptographic resources. In: Wiener, M.J. (ed.) CRYPTO'99. LNCS, vol. 1666, pp. 591–608. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 15–19, 1999)

[57] Gilboa, N., Ishai, Y.: Distributed point functions and their applications. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 640–658. Springer, Heidelberg, Germany, Copenhagen, Denmark (May 11–15, 2014)

[58] Goldreich, O.: Foundations of Cryptography — Basic Tools. Cambridge University Press (2001)

[59] Goldreich, O., Goldwasser, S., Micali, S.: How to construct random functions. J. ACM 33(4), 792–807 (1986)

[60] Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or a completeness theorem for protocols with honest majority. In: STOC. pp. 218–229 (1987)

[61] Goldreich, O., Ostrovsky, R.: Software protection and simulation on oblivious RAMs. J. ACM 43(3), 431–473 (1996)

[62] Gordon, S.D., Katz, J., Kolesnikov, V., Krell, F., Malkin, T., Raykova, M., Vahlis, Y.: Secure two-party computation in sublinear (amortized) time. In: ACM CCS. pp. 513–524 (2012)

[63] Gordon, S.D., Katz, J., Wang, X.: Simple and efficient two-server ORAM. In: Peyrin, T., Galbraith, S. (eds.) ASIACRYPT 2018, Part III. LNCS, vol. 11274, pp. 141–157. Springer, Heidelberg, Germany, Brisbane, Queensland, Australia (Dec 2–6, 2018)

[64] Halevi, S., Shoup, V.: Bootstrapping for helib. In: Advances in Cryptology - EUROCRYPT. pp. 641–670 (2015)

[65] Ishai, Y., Lai, R.W.F., Malavolta, G.: A geometric approach to homomorphic secret sharing. In: Garay, J.A. (ed.) Public-Key Cryptography - PKC 2021 - 24th IACR International Conference on Practice and Theory of Public Key Cryptography, Virtual Event, May 10-13, 2021, Proceedings, Part II. Lecture Notes in Computer Science, vol. 12711, pp. 92–119. Springer (2021), https://doi.org/10.1007/978-3-030-75248-4_4

[66] Kalyanasundaram, B., Schnitger, G.: The probabilistic communication complexity of set intersection. SIAM J. Discrete Math. 5(4), 545–557 (1992)

[67] Lai, R.W.F., Malavolta, G., Schröder, D.: Homomorphic secret sharing for low degree polynomials. In: Peyrin, T., Galbraith, S.D. (eds.) Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part III. Lecture Notes in Computer Science, vol. 11274, pp. 279–309. Springer (2018), https://doi.org/10.1007/978-3-030-03332-3_11

[68] Newman, Z., Servan-Schreiber, S., Devadas, S.: Spectrum: High-bandwidth anonymous broadcast with malicious security. IACR Cryptol. ePrint Arch. 2021, 325 (2021)

[69] Orlandi, C., Scholl, P., Yakoubov, S.: The rise of paillier: Homomorphic secret sharing and public-key silent OT. In: Canteaut, A., Standaert, F.X. (eds.) EUROCRYPT 2021, Part I. LNCS, vol. 12696, pp. 678–708. Springer, Heidelberg, Germany, Zagreb, Croatia (Oct 17–21, 2021)

[70] Ostrovsky, R., Shoup, V.: Private information storage (extended abstract). In: ACM Symposium on the Theory of Computing. pp. 294–303 (1997)

[71] Ostrovsky, R., Skeith III, W.: Private searching on streaming data. In: Advances in Cryptology - CRYPTO 2005. pp. 223–240 (2005)

[72] Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) EUROCRYPT'99. LNCS, vol. 1592, pp. 223–238. Springer, Heidelberg, Germany, Prague, Czech Republic (May 2–6, 1999)

[73] Rindal, P., Schoppmann, P.: VOLE-PSI: Fast OPRF and circuit-PSI from vector-OLE. In: Canteaut, A., Standaert, F.X. (eds.) EUROCRYPT 2021, Part II. LNCS, vol. 12697, pp. 901–930. Springer, Heidelberg, Germany, Zagreb, Croatia (Oct 17–21, 2021)

[74] Rivest, R.L., Adleman, L., Dertouzos, M.L.: On data banks and privacy homomorphisms. In: Foundations of secure computation (Workshop, Georgia Inst. Tech., Atlanta, Ga., 1977), pp. 169–179

[75] Rivest, R.L., Adleman, L., Dertouzos, M.L.: On data banks and privacy homomorphisms. In: Foundations of secure computation (Workshop, Georgia Inst. Tech., Atlanta, Ga., 1977), pp. 169–179. Academic, New York (1978)

[76] Roy, L., Singh, J.: Large message homomorphic secret sharing from DCR and applications. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part III. LNCS, vol. 12827, pp. 687–717. Springer, Heidelberg, Germany, Virtual Event (Aug 16–20, 2021)

[77] Ryffel, T., Tholoniat, P., Pointcheval, D., Bach, F.: Ariann: Low-interaction privacy-preserving deep learning via function secret sharing. arXiv preprint arXiv:2006.04593 (2020)

[78] Schoppmann, P., Gascón, A., Reichert, L., Raykova, M.: Distributed vector-OLE: Improved constructions and implementation. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019. pp. 1055–1072. ACM Press (Nov 11–15, 2019)

[79] Shelat, A., Doerner, J.: Scaling ORAM for secure computation. In: ACM SIGSAC CCS (2017)

[80] Trieu, N., Shehata, K., Saxena, P., Shokri, R., Song, D.: Epione: Lightweight contact tracing with strong privacy. arXiv preprint arXiv:2004.13293 (2020)

[81] Wang, F., Yun, C., Goldwasser, S., Vaikuntanathan, V., Zaharia, M.: Splinter: Practical private queries on public data. In: 14th USENIX Symposium on Networked Systems Design and Implementation, NSDI. pp. 299–313 (2017)

[82] Wang, X., Chan, T.H., Shi, E.: Circuit ORAM: on tightness of the goldreich-ostrovsky lower bound. In: ACM SIGSAC CCS. pp. 850–861 (2015)

[83] Weng, C., Yang, K., Katz, J., Wang, X.: Wolverine: fast, scalable, and communication-efficient zero-knowledge proofs for boolean and arithmetic circuits. In: 2021 IEEE Symposium on Security and Privacy (SP). pp. 1074–1091. IEEE (2021)

[84] Weng, C., Yang, K., Xie, X., Katz, J., Wang, X.: Mystique: Efficient conversions for zero-knowledge proofs with applications to machine learning. In: 30th {USENIX} Security Symposium ({USENIX} Security 21). pp. 501–518 (2021)

[85] Yang, K., Sarkar, P., Weng, C., Wang, X.: Quicksilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field. IACR Cryptol. ePrint Arch. 2021, 76 (2021)

[86] Yang, K., Weng, C., Lan, X., Zhang, J., Wang, X.: Ferret: Fast extension for correlated ot with small communication. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security. pp. 1607–1626 (2020)

[87] Yao, A.C.C.: How to generate and exchange secrets (extended abstract). In: FOCS. pp. 162–167 (1986)

[88] Zahur, S., Wang, X.S., Raykova, M., Gascón, A., Doerner, J., Evans, D., Katz, J.: Revisiting square-root ORAM: efficient random access in multi-party computation. In: IEEE Symposium on Security and Privacy, SP. pp. 218–234 (2016)