Function Secret Sharing*

Elette Boyle[1], Niv Gilboa[2], and Yuval Ishai[1]

[1] Computer Science Department, Technion
`eboyle@alum.mit.edu, yuvali@cs.technion.ac.il`
[2] Dept. of Communication Systems Engineering, Ben Gurion University
`gilboan@bgu.ac.il`

Abstract. Motivated by the goal of securely searching and updating distributed data, we introduce and study the notion of *function secret sharing* (FSS). This new notion is a natural generalization of distributed point functions (DPF), a primitive that was recently introduced by Gilboa and Ishai (Eurocrypt 2014). Given a positive integer $p \geq 2$ and a class $\mathcal{F}$ of functions $f : \{0,1\}^n \to \mathbb{G}$, where $\mathbb{G}$ is an Abelian group, a p-party FSS scheme for $\mathcal{F}$ allows one to split each $f \in \mathcal{F}$ into p succinctly described functions $f_i : \{0,1\}^n \to \mathbb{G}$, $1 \leq i \leq p$, such that: (1) $\sum_{i=1}^{p} f_i = f$, and (2) any strict subset of the f_i hides f. Thus, an FSS for $\mathcal{F}$ can be thought of as method for succinctly performing an "additive secret sharing" of functions from $\mathcal{F}$. The original definition of DPF coincides with a two-party FSS for the class of point functions, namely the class of functions that have a nonzero output on at most one input.

We present two types of results. First, we obtain efficiency improvements and extensions of the original DPF construction. Then, we initiate a systematic study of general FSS, providing some constructions and establishing relations with other cryptographic primitives. More concretely, we obtain the following main results:

– Improved DPF. We present an improved (two-party) DPF construction from a pseudorandom generator (PRG), reducing the length of the key describing each f_i from $O(\lambda \cdot n^{\log_2 3})$ to $O(\lambda n)$, where λ is the PRG seed length.
– Multi-party DPF. We present the first nontrivial construction of a p-party DPF for $p \geq 3$, obtaining a near-quadratic improvement over a naive construction that additively shares the truth-table of f. This constrcution too can be based on any PRG.
– FSS for simple functions. We present efficient PRG-based FSS constructions for natural function classes that extend point functions, including interval functions and partial matching functions.
– A study of general FSS. We show several relations between general FSS and other cryptographic primitives. These include a construction of general FSS via obfuscation, an indication for the implausibility of constructing general FSS from weak cryptographic assumptions such as the existence of one-way functions, a completeness result, and a relation with pseudorandom functions.

* Research supported by the European Union's Tenth Framework Programme (FP10/2010-2016) under grant agreement no. 259426 ERC-CaC. The first and third authors were additionally supported by ISF grants 1361/10 and 1709/14 and BSF grant 2012378.

1 Introduction

A secret sharing scheme [44] allows a dealer to randomly split a secret s into p shares, such that certain subsets of the shares can be used to reconstruct the secret and others reveal nothing about it. The simplest type of secret sharing is *additive secret sharing*, where the secret is an element of an Abelian group $\mathbb{G}$, it can be reconstructed by adding all p shares, and every subset of $p-1$ shares reveals nothing about the secret. A useful feature of this secret sharing scheme is that it is *homomorphic* in the sense that if p parties hold shares of many secrets, they can locally compute shares of the sum of all secrets. This feature of additive secret sharing (more generally, linear secret sharing) is useful for many cryptographic applications.

In this work we study the following natural extension of additive secret sharing. 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 $f \in \mathcal{F}$ into p functions $f_1, \ldots, f_p$ such that: (1) $f(x) = \sum_{i=1}^{p} f_i(x)$ (on every input x), (2) each f_i is described by a short key k_i that enables its efficient evaluation, yet (3) any strict subset of the keys completely hides f? We refer to a solution to this problem as a *function secret sharing* (FSS) scheme for $\mathcal{F}$.

If one insists on perfectly hiding f, then it can be shown that, even for very simple classes $\mathcal{F}$, the best possible solution is to additively share the truth-table representation of f, whose shares consist of 2^n group elements. But if one considers the computational notion of hiding, then there are no apparent limitations to what can be done for polynomial-time computable f. The power of such computationally hiding FSS schemes is the main question considered in this work.

We note that other types of secret sharing of functions have been considered in the literature, mostly in the context of threshold cryptography (cf. [18, 16]). 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 an additive (or homomorphic) representation of the output which is essential for the applications we consider.

A useful instance of FSS, recently introduced by Gilboa and Ishai [26], 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. For $x \in \{0,1\}^n$ and $y \in \mathbb{G}$, we denote by $f_{x,y}$ the point function that evaluates to y on input x and to 0 on all other inputs. The main result of [26] was an efficient construction of a DPF from any pseudorandom generator (PRG), or equivalently any one-way function [34].[3] More concretely, given a PRG with seed length λ, the length of each key k_i is $O(\lambda \cdot n^{\log_2 3})$.

The DPF problem was motivated in [26] by applications to improving the communication and computation complexity of 2-server private informa-

[3] The construction from [26] is described for the special case where $\mathbb{G} = \mathbb{Z}_2^m$, but it can be easily extended to the case of a general Abelian $\mathbb{G}$.

tion retrieval (PIR) [15, 14, 38] and related problems, as well as by the complexity theoretic problem of worst-case to average-case reductions. To further motivate the questions considered in this work, we discuss two typical application scenarios for DPF and the benefits that could be gained by extending DPF to more general instances of FSS.

Multi-server PIR and secure keyword search. Suppose that each of p servers holds a database D of m 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 client splits f into p additive shares and sends 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 find the number of matches by adding the p group elements received from the servers. 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., [41]).

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 p 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.[4] 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 [40], the "writing" analogue of PIR.

1.1 Our Contribution

In this work we improve and extend the work of [26], presenting two types of results. First, we improve the efficiency of the previous DPF construction and obtain the first nontrivial p-party DPF constructions for $p \geq 3$. Second, we initiate a systematic study of general FSS, providing some

[4] Handling malicious clients who may try to tamper with this process is beyond the scope of this work; we note, however, that due to the succinctness and simple structure of FSS shares one could employ general techniques for secure multiparty computation for this purpose without a major toll on efficiency.

constructions and establishing relations with other cryptographic primitives. More concretely, we obtain the following main results:

Improved DPF. We present an improved (two-party) DPF construction from one-way functions, reducing the length of the key describing each f_i from $O(\lambda \cdot n^{\log_2 3})$ to $O(\lambda n)$, where λ is a security parameter (that can be thought of as the seed length of a PRG) and n is the input and output length. We also obtain a similar improvement in the evaluation time. This improvement can have relevance to the practical efficiency of 2-server PIR and related primitives.

Multi-party DPF. We provide the first nontrivial construction of a p-party DPF for $p \geq 3$, obtaining a near-quadratic improvement over a naive construction that additively shares the truth-table of f. This construction too can be based on the (necessary) assumption that a one-way function exists. More concretely, letting $N = 2^n$ denote the input domain size and λ a PRG seed length, the length of each DPF key k_i is $O(\lambda \cdot 2^{p/2} \cdot N^{1/2})$. Improving the asymptotic dependence on N (without relying on stronger assumptions) is one of the main questions left open by this work. For $p \geq 3$, our p-party DPF implies the first p-server, $(p-1)$-private PIR protocols with sublinear query length and constant answer length, as well as the first $(p-1)$-private sublinear-communication storage schemes in the model of [40].

FSS for simple functions. We present efficient PRG-based FSS constructions for natural function classes that go beyond point functions. These include interval functions and instances of partial matching functions. As illustrated above, such extensions can be used to support more general search queries or selection criteria.

A study of general FSS. We initiate a study of general FSS by showing several relations between FSS and other primitives. In particular, we obtain the following results:

- We observe that FSS for general polynomial-time computable functions can be obtained from an ideal obfuscation and one-way functions. This implies (using [2]) a provable construction in the generic multilinear map model, as well as a heuristic construction using existing candidates. Furthermore, building on a recent work of Canetti et al. [13], we obtain a similar result based on Indistinguishability Obfuscation (iO) with sub-exponential security.
- Complementing the above, we give evidence against the possibility of constructing general FSS from weak cryptographic assumptions such as the existence of one-way functions or even oblivious transfer. We do this by showing that general FSS implies low-communication protocols for secure two-party computation that rely on a *reusable* source of correlated randomness (that can be realized via one-time offline preprocessing). Currently all known approaches for obtaining such protocols rely on fully homomorphic encryption or related primitives. We show that a similar "barrier" applies even to FSS for the complexity class AC^0. This should be contrasted with our PRG-based positive results, which apply to strict sub-classes of AC^0.

– We prove the following completeness result: assuming the hardness of LWE, there is a class $\mathcal{F}$ of functions in NC^1 such that an efficient FSS for $\mathcal{F}$ implies an efficient FSS for arbitrary polynomial-time computable functions.
– We show that in an FSS scheme for any "sufficiently rich" function class $\mathcal{F}$ (which covers point functions as a special case), each share f_i must define a pseudorandom function. Note that this is not a-priori clear from the security definition, which only requires that the shares hide f.

1.2 Related Work

In this section we discuss alternative approaches for tackling the motivating applications for DPF and FSS discussed above. Compared to our PRG-based constructions, all of these approaches have significant limitations in efficiency or security.

Information-theoretic multi-server PIR. The notion of p-party DPF roughly corresponds to a p-server PIR protocol with 1-bit answers and computational privacy against any $p-1$ servers. In this setting, insisting on information-theoretic privacy implies that the length of the query sent to each server must be linear in the database size [5, 45]. This barrier can be overcome by either settling for a lower privacy threshold $t < p-1$ or allowing for longer answers. (The latter relaxation is not suitable for applications that involve "writing," and results in PIR protocols that have poor information rate when applied to databases with long records.) Even with the above relaxations, the asymptotic communication complexity of the best known information-theoretic PIR protocols [15, 47, 21, 4, 6, 19] is worse than that of DPF-based protocols.

Single-server PIR. Single-server, computationally-private PIR protocols [38, 12, 39] can achieve similar communication complexity to DPF-based 2-server protocols, and moreover they have the advantages of requiring only one server and not being vulnerable to colluding servers. However, they are not suitable for applications that involve writing, they cannot support constant-size answers, and they do not extend to the richer type of queries supported by our PRG-based FSS constructions (except when using fully homomorphic encryption, discussed below). Perhaps most importantly, single-server PIR protocols make an intensive (and in some sense inherent [20]) use of public-key cryptography, compared to our PRG-based constructions for DPF and simple instances of FSS. Thus, the computational overhead on the server side, which typically forms the practical efficiency bottleneck, can be much lower in DPF-based protocols.

FHE and TFHE. Fully homomorphic encryption (FHE) [23] can be used to accommodate the richer query types implied by general FSS. However, the other limitations of PIR discussed above apply also to FHE-based protocols, and moreover the concrete computational cost of current implementations is even worse. Constructions of a threshold variant of FHE (TFHE) from [1] can be used to realize a relaxed form of FSS, where the output of the function f is secret-shared in a more redundant way that nevertheless still supports homomorphic additions and allows for efficient decoding of the output from the shares without the knowledge of a secret key. However, TFHE is a stronger primitive than standard FHE and its implementations are even less efficient. We note that our barriers for general FSS from weak assumptions do not apply to FHE-based constructions, leaving open the possibility of realizing our general notion of FSS from FHE or specific assumptions such as LWE.

Oblivious RAM. Oblivious RAM (ORAM) [31] allows a client to efficiently access data stored on a remote server while hiding the contents of the data and the locations being accessed. However, despite the superficial similarity to the PIR scenario considered here, ORAM addresses a very different problem. In particular, ORAM requires that the client "own" the data and does not directly apply in the case where the data to be accessed comes from other sources, nor does it scale efficiently in the case of read and write operations by many clients who do not trust each other.

Organization. In Section 2 we formally define our notion of FSS and discuss several variants and relaxations of this notion. In Section 3 we describe new PRG-based constructions of DPF schemes and FSS schemes for simple function classes, as well as a general FSS construction via general-purpose obfuscation. Finally, in Section 4 we relate the FSS primitive to other cryptographic primitives and present some barriers to basing general FSS on weak primitives such as a one-way function.

2 Function Secret Sharing

We now formally define our notion of a function secret sharing (FSS) scheme. Recall that, unlike "standard" secret sharing for individual elements, we begin with the description of a *function* f that we wish to share among parties. The FSS scheme provides a means to split this function into separate keys, where each party's key enables him to efficiently generate a standard secret share of the evaluation $f(x)$, and yet each key individually does not reveal information about which function f has been shared.
Note that FSS schemes can differ in the underlying procedure for recovering $f(x)$ from the parties' key-computed shares (including the number of shares), and also in the relevant function class $\mathcal{F}$ for which correctness and security are supported. In what follows, we present a general version of this definition, allowing arbitrary output decoding procedures; however, in this work we focus on the setting in which the output decoder is a fixed *linear function* of parties' output shares. Namely, decoding will correspond to taking the sum of the output shares over an Abelian group structure. We discuss this choice of decoding structures below.

Definition 1 (Output Decoder). *A p-party share* output decoder DEC *is a tuple* $(S_1, \ldots, S_p, R, \mathsf{Dec})$ *specifying: share spaces* $S_1, \ldots, S_p$ *for each of the p parties; output space* R*; and a decoder function* Dec : $S_1 \times \cdots \times S_p \to R$ *taking parties' shares to an output.*

We define the p-party additive output decoder *for an Abelian group* $\mathbb{G}$ *to be the tuple* $\mathsf{DEC} = ((\mathbb{G}, \cdots, \mathbb{G}), \mathbb{G}, \mathsf{Dec}^+)$*, where* $\mathsf{Dec}^+(g_1, \ldots, g_p) = \sum_{i=1}^{p} g_i$ *computes the sum of elements w.r.t. the group operator of* $\mathbb{G}$*.*

Remark 1 (Modeling Function Families). We model a function family $\mathcal{F}$ as an infinite collection of bit strings f ("functions"), together with efficient procedures $\mathsf{IdentifyDomain}$ and $\mathsf{Evaluate}$, such that the procedure $D_f \leftarrow \mathsf{IdentifyDomain}(1^\lambda, f)$ interprets from the string f its corresponding input domain space, and $y \leftarrow \mathsf{Evaluate}(f, x)$, for any input $x \in D_f$, defines the "output" of f at x. By convention, we assume the description of f includes also the input length and output length of f. We refer the reader to e.g. [36] for a complete formal description of this model.
For simplicity of notation, in this work we will refer to the domain D_f of f without making explicit reference to the corresponding call to $\mathsf{IdentifyDomain}$, and will denote an evaluation $\mathsf{Evaluate}(f, x)$ by shorthand notation "$f(x)$."

Definition 2 (Function Secret Sharing). *For* $p \in \mathbb{N}, T \subsetneq [p]$*, a p-party, T-secure* function secret sharing (FSS) scheme *with respect to share output decoder* $\mathsf{DEC} = (S_1, \ldots, S_p, R, \mathsf{Dec})$ *and function class* $\mathcal{F}$ *is a pair of PPT algorithms* $(\mathsf{Gen}, \mathsf{Eval})$ *with the following syntax:*

– $\mathsf{Gen}(1^\lambda, f)$*: On input the security parameter* 1^λ *and function description* $f \in \mathcal{F}$*, the key generation algorithm outputs p keys,* $(k_1, \ldots, k_p)$*.*
– $\mathsf{Eval}(i, k_i, x)$*: On input a party index i, key* k_i *(which we assume to encode the input and output domains* D, R *of the shared function) and input string* $x \in D$*, the evaluation algorithm outputs a value* $y_i \in S_i$*, corresponding to this party's share of* $f(x)$*.*

satisfying the following correctness and secrecy requirements:

– **Correctness:** *For all* $f \in \mathcal{F}$*,* $x \in D_f$*,*

$$\Pr\big[(k_1, \ldots, k_p) \leftarrow \mathsf{Gen}(1^\lambda, f) \\ : \mathsf{Dec}\big(\mathsf{Eval}(1, k_1, x), \ldots, \mathsf{Eval}(p, k_p, x)\big) = f(x)\big] = 1.$$

– **Security:** *Consider the following indistinguishability challenge experiment for corrupted parties* $T \subset [p]$*:*

1: The adversary outputs $(f_0, f_1, \mathsf{state}) \leftarrow \mathcal{A}(1^\lambda)$*, where* $f_0, f_1 \in \mathcal{F}$ *with* $D_{f_0} = D_{f_1}$*.*
2: The challenger samples $b \leftarrow \{0,1\}$ *and* $(k_1, \ldots, k_p) \leftarrow \mathsf{Gen}(1^\lambda, f_b)$*.*
3: The adversary outputs a guess $b' \leftarrow \mathcal{A}((k_i)_{i \in T}, \mathsf{state})$*, given the keys for corrupted T.*

Denote by $\mathsf{Adv}(1^\lambda, \mathcal{A}) := \Pr[b = b'] - 1/2$ *as the advantage of* $\mathcal{A}$ *in guessing b in the above experiment, where probability is taken over the randomness of the challenger and of* $\mathcal{A}$*. We say the scheme* $(\mathsf{Gen}, \mathsf{Eval})$ *is T*-secure *if there exists a negligible function* ν *such that for all non-uniform PPT adversaries* $\mathcal{A}$*, it holds that* $\mathsf{Adv}(1^\lambda, \mathcal{A}) \leq \nu(\lambda)$*.*

Unless otherwise specified, we naturally interpret the output domain of the function f as an Abelian group $\mathbb{G}$ *(in particular,* $\{0,1\}^n$ *is interpreted as an Abelian group with respect to the xor group operator* $\oplus$*), and* DEC *is the corresponding* additive *output decoder as specified in Definition 1.*

Remark 2. A few remarks about our definition.

1. (Adversary Structure). We say an FSS scheme is *t-secure* for threshold $t < p$ if it is T-secure for all $T \subset [p]$ of size $|T| \leq t$. By default, when not otherwise specified, "secure FSS" will refer to $(p-1)$-security, in which any strict subset of parties may be corrupted.
2. (Variable Output Domains). For simplicity, we take the convention that all functions within a class $\mathcal{F}$ share the same output domain (i.e., $f : D_f \to R$ for shared R). We may also extend in a straightforward way to the setting in which each function f has a possibly different output domain R_f. The corresponding security will be required to hold with respect to pairs of functions $f_0, f_1 \in \mathcal{F}$ with both matching domains ($D_{f_0} = D_{f_1}$) *and* ranges ($R_{f_0} = R_{f_1}$).
3. (Simulation-Based Security). Our game-based security definition mirrors that of semantic security, where the shares of corrupted parties play the role of an "encryption" of f. As with semantic security, our game-based indistinguishability security definition can equivalently be expressed as a *simulation-based* definition, where one must be able to simulate the distribution of corrupted parties' shares without knowledge of the shared function f (*cf.* [32, 28]).

Output Decoding Schemes. The FSS definition above is presented with respect to an arbitrary choice of output decoding function Dec. Based on the structure of the chosen decoding process, the corresponding FSS scheme will have very different properties. For example, more complex decoding procedures Dec open the possibility of achieving FSS for more general classes of functions $\mathcal{F}$, 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.

Arbitrary reconstruction. Consider, for example, the FSS notion as defined, but with *no restriction* on the reconstruction procedure for parties' output shares. Such wide freedom will render the notion non-meaningful, as it gives rise to trivial constructions. Indeed, for any efficient function family $\mathcal{F}$, one can 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 Dec will first reconstruct the description of f, and then compute and output the value $f(x)$.
This construction satisfies correctness and security as specified 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 (resulting from executing Eval) 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.[5] Yet this construction (and thus definition) is unsatisfying: although the output shares now hide f, their size is massive—for every output, comparable to a copy of f itself.

Succinct, function-private output shares. We thus 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 shown in Section 4.2, achieving such an FSS scheme for general functions implies a form of communication-efficient secure multi-party computation that is currently only achievable using advanced cryptographic machinery (i.e., fully homomorphic encryption or reusable garbled circuits). However, there is one final property that enables an important class of applications, but which is not yet guaranteed: a notion of *share compressibility*.

Let us explore this property. Recall that one of the exciting application regimes of distributed point functions (DPF) [26] was enabling communication-efficient secure (2-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 DPF 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 [37]! In extending the DPF notion to more general FSS primitives, we wish to preserve and extend this class of applications. 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.

[5] 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.

Our setting: Linear share decoding. In this work, we focus purely on the setting of FSS where the output decoder is a *linear function* of parties' shares: specifically, the additive output decoder as in Definition 1. 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 we show in Section 4.1, the linear reconstruction in conjunction with basic key security directly implies *function privacy*.
We hence restrict our attention to this setting, and unless otherwise specified will implicitly take an "FSS scheme" to be one with a linear reconstruction procedure DEC defined above.

2.1 Preliminaries

In this work, we make use of several cryptographic tools. For formal definitions of the notions of *computational indistinguishability*, *pseudorandom generators*, and *pseudorandom functions*, we refer the reader to [28]. For *fully homomorphic encryption* definitions and constructions, see, e.g., [23, 25, 10]. And, for program obfuscation, see *virtual black-box* [3], *indistinguishability obfuscation* (*i$\mathcal{O}$*) [3, 22], and *probabilistic i$\mathcal{O}$* [13].

3 New Constructions

In the following section, we present several new constructions of FSS schemes for various function families.
We begin in Section 3.1 by showing two new constructions for the family of *point functions*. The first is a two-key construction that significantly reduces the key size and computational complexity compared to all previous constructions. The second is the first p-key construction, secure against coalitions of up to $p-1$ key holders, with key size a square root of what a trivial construction achieves.
In Section 3.2, we go beyond the family of point functions in several ways. We identify general low-level transformations that modify an existing FSS scheme into one for a modified function class. We combine some of these general transformations, in addition to existing FSS schemes, to yield constructions for more expressive function families. In addition, we extend the previous results for point functions to include the family of interval functions with minimal overhead.
In Section 3.3, we show that FSS for *general efficient functionalities* is implied by certain forms of program obfuscation (namely, virtual black-box or sub-exponentially secure indistinguishability obfuscation).

3.1 Point Functions

Definition 3. *For $a, b \in \{0,1\}^n$, the* point function $P_{a,b} : \{0,1\}^n \rightarrow \{0,1\}^m$ *is defined by $P_{a,b}(a) = b$ and $P_{a,b}(a') = 0^m$ for all $a' \neq a$.*

We begin by describing a construction for the class of two-party point functions $P_{a,b}(x) : \{0,1\}^n \to \{0,1\}^m$. The scheme we show, $(\mathsf{Gen}^\bullet, \mathsf{Eval}^\bullet)$, reduces the key size and the computational complexity compared to the construction of distributed point functions in [26], from $O(\lambda n^{\log 3})$ to $O(\lambda n)$, making use of a pseudorandom generator with seed length λ. $(\mathsf{Gen}^\bullet, \mathsf{Eval}^\bullet)$ are given by Algorithms 1 and 2.

At a high level, the scheme works as follows. Each party's key, k_0 and k_1, defines a binary tree of depth n with a pseudo-random string at each node (the strings are the $S||T$'s defined in lines 9 and 10 of Algorithm 2). The binary trees defined by k_0 and k_1 are identical except for the path from the root to the target point $a = a_1, \ldots, a_n$. On this path, the strings in the two trees are chosen pseudo-randomly and independently of each other.

$\mathsf{Eval}^\bullet(\beta, k_\beta, x)$ traverses a path in the tree that k_β defines from the root to $x = x_1, \ldots, x_n$, computing the strings along the path. At each node with string $S_0^\beta[i]||S_1^\beta[i]||T_0^\beta[i]||T_1^\beta[i]$, $\mathsf{Eval}^\bullet$ computes the corresponding strings for its x_ith child (left or right) by expanding either the left or right seed $S_{x_i}^\beta[i]$ using the pseudo-random generator $G(S_{x_i}^\beta[i])$, and adding in "correction" strings cs, ct (from the key k_β) to the corresponding "s" and "t" portions of the expanded output, as dictated by the bit $T_{x_i}^\beta[i]$.

The function of $\mathsf{Gen}^\bullet(1^\lambda, a, b)$ is to ensure the correct creation of the two trees. Specifically, it ensures that at the exact point that a prefix of x diverges from the path to a, $\mathsf{Eval}^\bullet(0, k_0, x)$ and $\mathsf{Eval}^\bullet(1, k_1, x)$ compute the *same* strings S, T. (Then, for any path continuing from this point, the values will always remain equal). For prefixes that diverge at the root (i.e., $a_1 \neq x_1$), each key includes the same string since lines 2, 3 sets $S_{\neg a_1}^1[1] = S_{\neg a_1}^0[1]$ and $T_{\neg a_1}^1[1] = T_{\neg a_1}^0[1]$ (superscript here is party id). Any other location of diverging prefixes is resolved by setting the correct strings cs, ct in lines 6-9 of Algorithm 1.

$\mathsf{Gen}^\bullet$ has a negligble probability of failure (expressed by setting $w \leftarrow 0$), which is a result of generating equal random values for $S_{a_n}^0[n] = S_{a_n}^1[n]$. It is always possible to run $\mathsf{Gen}^\bullet$ again if it fails. In Algorithm 5 we show how to obtain a scheme without any error.

Intuitively, security holds for $(\mathsf{Gen}^\bullet, \mathsf{Eval}^\bullet)$ because all information related to the point function $f_{a,b}$ is encoded in the strings cs, ct, *masked* by pseudorandom strings whose seeds appear only in the other party's key. Note that the original values S, T in lines 2,3 are completely independent of the point function.

Due to space limitations, we refer the reader to the full version of this work for a complete proof of correctness and security of $(\mathsf{Gen}^\bullet, \mathsf{Eval}^\bullet)$.

Notation 1 *We use the following notational conventions in Algorithms 1 and 2. Superscripts denote the party id, and are used for strings appearing in the tree defined by this party's key. Square brackets denote the depth of a node in the tree, ranging from* 1 *to* n. *One or two binary-valued subscripts are used to distinguish between strings that are associated with a specific node in the tree (e.g., to be used when continuing to the left or right from this node). For example* $S_\alpha^\beta[i]$ *is in the tree defined by party* β*'s key* k_β *at depth* i*, and is one of two strings (the other is* $S_{\neg\alpha}^\beta[i]$*) at a specific node in the tree.*

Algorithm 1 $\mathsf{Gen}^{\bullet}(1^\lambda, a, b)$

```
1: Let $G : \{0,1\}^\lambda \longrightarrow \{0,1\}^{\max\{2\lambda+2,m\}}$ be a PRG.
2: Choose three random seeds $S^0_{a_1}[1], S^1_{a_1}[1], S^0_{\neg a_1}[1] \in \{0,1\}^\lambda$ and set $S^1_{\neg a_1}[1] \leftarrow S^0_{\neg a_1}[1]$.
3: Choose four random bits $T^\beta_\alpha[1]$, for $\alpha, \beta \in \{0,1\}$, subject to $T^0_{a_1}[1] \neq T^1_{a_1}[1]$ and $T^0_{\neg a_1}[1] = T^1_{\neg a_1}[1]$.
4: for $i = 1$ to $n-1$ do
5:     Let $G(S^\beta_{a_i}[i]) = s^\beta_0 || s^\beta_1 || t^\beta_0 || t^\beta_1$, where $s^\beta_\alpha \in \{0,1\}^\lambda$, $t^\beta_\alpha \in \{0,1\}$ for $\alpha, \beta \in \{0,1\}$.
6:     Randomly choose $cs_{0,a_{i+1}}, cs_{1,a_{i+1}} \in \{0,1\}^\lambda$.
7:     Randomly choose $cs_{0,\neg a_{i+1}}, cs_{1,\neg a_{i+1}} \in \{0,1\}^\lambda$ subject to $\bigoplus_{\beta=0}^{1}(cs_{\beta,\neg a_{i+1}} \oplus s^\beta_{\neg a_{i+1}}) = 0$.
8:     Randomly choose $ct_{0,a_{i+1}}, ct_{1,a_{i+1}} \in \{0,1\}$ subject to $\bigoplus_{\beta=0}^{1}(ct_{\beta,a_{i+1}} \oplus t^\beta_{a_{i+1}}) = 1$.
9:     Randomly choose $ct_{0,\neg a_{i+1}}, ct_{1,\neg a_{i+1}} \in \{0,1\}$ subject to $\bigoplus_{\beta=0}^{1}(ct_{\beta,\neg a_{i+1}} \oplus t^\beta_{\neg a_{i+1}}) = 0$.
10:    Set $CW_\beta[i] \leftarrow cs_{\beta,0} || cs_{\beta,1} || ct_{\beta,0} || ct_{\beta,1}$ for $\beta = 0, 1$.
11:    Set $S^\beta_\alpha[i+1] \leftarrow s^\beta_\alpha \oplus cs_{\tau,\alpha}$ for $\tau = T^\beta_{a_i}[i]$ and $\alpha, \beta \in \{0,1\}$.
12:    Set $T^\beta_\alpha[i+1] \leftarrow t^\beta_\alpha \oplus ct_{\tau,\alpha}$ for $\tau = T^\beta_{a_i}[i]$ and $\alpha, \beta \in \{0,1\}$.
13: end for
14: if $G(S^0_{a_n}[n]) \neq G(S^1_{a_n}[n])$ then
15:    Set $w \leftarrow (G(S^0_{a_n}[n]) + G(S^1_{a_n}[n]))^{-1} \cdot b$ with arithmetic over $\mathbb{F}_{2^m}$.
16: else
17:    Set $w \leftarrow 0$.
18: end if
19: Set $k_\beta \leftarrow ((S^\beta_0[1], S^\beta_1[1], T^\beta_0[1], T^\beta_1[1]), (CW_0[1], CW_1[1], \ldots, CW_0[n-1], CW_1[n-1]), w)$.
20: Return $(k_0, k_1)$.
```

A p-party protocol. For some applications, one may wish to share a function f among *several parties*. In this setting, there is an additional challenge in maintaining security against collusions of corrupted parties. Note that for any family of functions $\mathcal{F} : \{0,1\}^n \to \{0,1\}^m$, we can trivially support secret sharing of $\mathcal{F}$ across p parties with security against coalitions of up to $p-1$ keys, with key size $2^n \cdot m$. Indeed, this amounts to simply secret sharing the entire evaluation table of the function f among parties as a string: $\mathsf{Gen}(1^\lambda, f)$ chooses p random strings $k_1, \ldots, k_p \in \{0,1\}^{2^n \cdot m}$ such that $\bigoplus_{i=1}^{p} k_i[x] = f(x)$ for all $x \in \{0,1\}^n$.
We now present a scheme $(\mathsf{Gen}^{p_0}, \mathsf{Eval}^{p_0})$ sharing a DPF $P_{a,b} : \{0,1\}^n \to \{0,1\}^m$, secure against any coalition of at most $p-1$ key holders, and with key length $O(2^{n/2} \cdot 2^{p/2} \cdot m)$. For a constant number of parties $p \in O(1)$, this corresponds to a square root of the key length in the trivial solution.
At a high level, the scheme $(\mathsf{Gen}^{p_0}, \mathsf{Eval}^{p_0})$ works as follows. Consider the 2^n-entry evaluation table of the secret function $f_{a,b}$ as a $2^{n/2} \times 2^{n/2}$ grid[6], where rows and columns are indexed by the first and second $n/2$

[6] The dimensions of the table in the algorithm are slightly different, which results in reducing the key size by a factor of $2^{p/2}$.

Algorithm 2 $\mathsf{Eval}^{\bullet}(\beta, k_\beta, x)$

```
1: Let $G : \{0,1\}^\lambda \longrightarrow \{0,1\}^{\max\{2\lambda+2,m\}}$ be a PRG.
2: Let the binary representation of $x$ be $x = x_1, \ldots, x_n$.
3: Parse $k_\beta$ as $k_\beta = ((S_0^\beta[1], S_1^\beta[1], T_0^\beta[1], T_1^\beta[1]), (CW_0[1], CW_1[1], \ldots, CW_0[n-1], CW_1[n-1]), w)$.
4: Set $S \leftarrow S_{x_1}^\beta[1]$.
5: Set $T \leftarrow T_{x_1}^\beta[1]$.
6: for $i = 2$ to $n$ do
7:     Parse $G(S)$ as $G(S) = s_0||s_1||t_0||t_1$.
8:     Parse $CW_T[i-1]$ as $CW_T[i-1] = cs_{T,0}||cs_{T,1}||ct_{T,0}||ct_{T,1}$.
9:     Set $S \leftarrow s_{x_i} \oplus cs_{T,x_i}$.
10:    Set $T \leftarrow t_{x_i} \oplus ct_{T,x_i}$.
11: end for
12: Return $G(S) \cdot w$ with arithmetic over $\mathbb{F}_{2^m}$.
```

bits of the input. The algorithm Gen^{p_0} generates the following values: For each row $\gamma' \in \{0,1\}^{n/2}$ in this table, it samples 2^{p-1} random λ-bit strings $s_{\gamma',1}, \ldots, s_{\gamma',2^{p-1}} \in \{0,1\}^\lambda$ to be used as seeds for a pseudorandom generator (PRG) G. In addition, it generates 2^{p-1} total (not per row) "correction words" $cw_1, \ldots, cw_{2^{p-1}} \in (\{0,1\}^m)^{2^{n/2}}$, as a function of the strings $s_{\gamma',\ell}$ and the secret function $P_{a,b}$. Each party i receives as its key the collection of all 2^{p-1} correction words and some subset of the PRG seeds. The algorithm Eval^{p_0}, given a party's key and input x, parses $x = (\gamma', \delta') \in \{0,1\}^{n/2} \times \{0,1\}^{n/2}$, takes its set of PRG seeds corresponding to the row γ', expands each via G to a vector $(\{0,1\}^m)^{2^{n/2}}$ which matches the form of a row in the function evaluation table, takes the exclusive-or of all the expanded vectors together with the corresponding subset of correction words (i.e. the subset of $\{cw_j : j \in [2^{p-1}]\}$ for which its key contained the jth row-γ' seed $s_{\gamma',j}$), and outputs the (δ')th component of this row vector. Collectively, this description corresponds to Step 6 of Algorithm 4.

The subset of seeds, and the generation of the correction words is chosen by Gen^{p_0} so as to ensure the following properties:

1. For each row γ' *not* equal to the special row γ, and for each of the 2^{p-1} PRG seeds $s_{\gamma',j}$ corresponding to this row, it will hold that the number of parties holding $s_{\gamma',j}$ in their key is *even*. Thus, during the evaluation phase, all contributions from $G(s_{\gamma',j})$ and from its corresponding jth correction word cw_j will cancel out, leaving the desired 0 evaluation.
2. For the special row γ, each $s_{\gamma,j}$ will appear in an *odd* number of parties' keys. This means there will be exactly one copy of each $G(s_{\gamma,j})$ and each cw_j remaining in the combined evaluation xor from all parties. Further, for each party i, there is at least one seed $s_{\gamma,j}$ (in our construction, exactly one) for which party i is the *only* party given $s_{\gamma,j}$. This will be important for security, as $G(s_{\gamma,j})$ for the uncorrupted party will serve as a mask to hide information on $P_{a,b}$ in the correction words.

3. Given any $p-1$ keys, Case (1) and (2) are indistinguishable.
4. The correction words $cw_j, j \in [2^{p-1}]$ are chosen randomly subject to the constraint $\bigoplus_{j=1}^{2^{p-1}}(cw_j \oplus G(s_{\gamma,j})) = e_\delta \cdot b$, where e_δ denotes the unit vector whose δth component is equal to 1. From Property (2), this constraint exactly yields the required correctness guarantee. And, since the cw_j are random up to this condition, then even given any $(2^{p-1}-1)$ of the seeds $s_{\gamma,j}$ (but with one missing), the distribution of these seeds together with all the cw_j's is computationally indistinguishable from random.

We now proceed to describe the scheme with these properties.
Given natural numbers p and q, it is readily apparent that for exactly q^{p-1} of the sequences of length p over the set $\{0,\dots,q-1\}$ the sum of the p elements modulo q is 0 and for exactly q^{p-1} of these sequences the sum of all the elements modulo q is 1. (One way to deduce this statement is that given any choice of the first $p-1$ elements in $\{0,\dots,q-1\}$ there is a single choice for the last element that makes the sum of the whole sequence 1 and a single choice that makes the sum 0). For the special case of $q=2$ we introduce the following useful notation.

Notation 2 *Given $p \in \mathbb{N}$, let E_p and O_p denote subsets of binary arrays of size $p \times 2^{p-1}$. Let E_p denote the set of all arrays such that the columns of each array are all the p-bit strings with an even number of 1 bits and let O_p denote the set of all arrays such that the columns of each array are all the p-bit strings with an odd number of 1 bits. We use $A \in_R E_p$ (or $A \in_R O_p$) to denote that A is randomly sampled from E_p (O_p). We use $e_a \cdot b$ to denote a vector of length $2^{|a|}$ with b in location a and 0 in all other locations.*

We present the p-party FSS scheme for point functions $(\mathsf{Gen}^{p_0}, \mathsf{Eval}^{p_0})$ in Algorithms 3 and 4.

Algorithm 3 $\mathsf{Gen}^{p_0}(1^\lambda, a, b)$

```
1: Let $G : \{0,1\}^\lambda \longrightarrow \{0,1\}^{m\mu}$ be a PRG ($\mu$ is defined in line 2).
2: Let $\mu \leftarrow \lceil 2^{n/2} \cdot 2^{(p-1)/2} \rceil$ and let $\nu \leftarrow \lceil 2^n/\mu \rceil$.
3: Regard $a$ as a pair $a = (\gamma, \delta)$, $\gamma \in [\nu], \delta \in [\mu]$.
4: Choose $\nu$ arrays $A_1, \dots, A_\nu$, s.t. $A_\gamma \in_R O_p$ and $A_{\gamma'} \in_R E_p$ for all $\gamma' \neq \gamma$.
5: Choose randomly and independently $\nu \cdot 2^{p-1}$ seeds $s_{1,1}, \dots, s_{\nu,2^{p-1}} \in \{0,1\}^\lambda$.
6: Choose $2^{p-1}$ random strings $cw_1, \dots, cw_{2^{p-1}} \in \{0,1\}^{m\mu}$ s.t. $\bigoplus_{j=1}^{2^{p-1}}(cw_j \oplus G(s_{\gamma,j})) = e_\delta \cdot b$.
7: Set $\sigma_{i,\gamma'} \leftarrow (s_{\gamma',1} \cdot A_{\gamma'}[i,1]) || \dots || (s_{\gamma',2^{p-1}} \cdot A_{\gamma'}[i, 2^{p-1}])$ for all $1 \le i \le p$, $1 \le \gamma' \le \nu$.
8: Set $\sigma_i = \sigma_{i,1} || \dots || \sigma_{i,\nu}$ for $1 \le i \le p$.
9: Let $k_i = (\sigma_i || cw_1 || \dots || cw_{2^{p-1}})$ for $1 \le i \le p$.
10: Return $(k_1, \dots, k_p)$.
```

Algorithm 4 $\mathsf{Eval}^{p_0}(i, k_i, x)$

```
1: Let $G : \{0,1\}^\lambda \longrightarrow \{0,1\}^{m\mu}$ be a PRG ($\mu$ is defined in line 2).
2: Let $\mu \leftarrow \lceil 2^{n/2} \cdot 2^{(p-1)/2} \rceil$ and let $\nu \leftarrow \lceil 2^n/\mu \rceil$.
3: Regard $x$ as a pair $x = (\gamma', \delta')$, $\gamma' \in [\nu], \delta' \in [\mu]$.
4: Parse $k_i$ as $k_i = (\sigma_i, cw_1, \ldots, cw_{2^{p-1}})$.
5: Parse $\sigma_i$ as $\sigma_i = s_{1,1}||\ldots||s_{1,2^{p-1}}||\ldots||s_{\nu,2^{p-1}}$.
6: Let $y_i \leftarrow \bigoplus_{1 \le j \le 2^{p-1}, s_{\gamma',j} \neq 0} (cw_j \oplus G(s_{\gamma',j}))$.
7: Return $y_i[\delta']$.
```

We informally argue that $(\mathsf{Gen}^{p_0}, \mathsf{Eval}^{p_0})$ is an FSS scheme for point functions. The scheme is correct because of the following. If $\mathsf{Gen}^{p_0}(k_i, x)$ outputs $(k_1, \ldots, k_p)$ then $\bigoplus_{i=1}^p \mathsf{Eval}^{p_0}(i, k_i, x) = \bigoplus_{i=1}^p y_i[\delta']$. If $\gamma' \neq \gamma$ then $A_{\gamma'} \in E_p$ and hence each of the terms $cw_j \oplus G(s_{\gamma',j})$ appears an even number of times in $\bigoplus_{i=1}^p y_i$, therefore canceling out and ensuring that $\bigoplus_{i=1}^p y = 0$. However, if $\gamma' = \gamma$ then $\bigoplus_{i=1}^p y_i = \sum_{j=1}^{2^{p-1}} cw_j \oplus G(s_{\gamma',j})$. By the definition of the correction words $cw_1, \ldots, cw_{2^{p-1}}$ we have that $\bigoplus_{i=1}^p y_i[\delta'] = 0$ if $\delta' \neq \delta$ while $\bigoplus_{i=1}^p y_i[\delta'] = b$ if $\delta' = \delta$, i.e. if $x = a$.
The scheme $(\mathsf{Gen}^{p_0}, \mathsf{Eval}^{p_0})$ is secret because each subset of at most $p-1$ keys k_i includes $p-1$ strings $\sigma_i = \sigma_{i,1}, \ldots, \sigma_{i,\nu}$. The distribution of seeds in $\sigma_{i,\gamma'}$ reflects the distribution of 1 bits in the i-th row of $A_{\gamma'}$. However, any $p-1$ rows of $A_{\gamma'}$ are distributed identically, regardless of whether $A_{\gamma'}$ is sampled randomly from E_p or it is sampled randomly from O_p. Therefore, the view of the strings σ_i does not give any information on γ. In addition, $cw_1, \ldots, cw_{2^{p-1}}$ are masked by $\bigoplus_{j=1}^{2^{p-1}} G(s_{\gamma,j})$ and there is at least one seed $s_{\gamma,j}$ which is not included in any of the keys in the subset. Therefore, all the correction words together cannot be distinguished from random strings of the appropriate length.
The length of a key k_i that Gen^{p_0} outputs is a sum of the length of σ_i, which is $\nu\lambda \cdot 2^{p-1}$ and the length of the correction words, which is $\mu m \cdot 2^{p-1}$. The key size is therefore $O(2^{n/2} 2^{(p-1)/2}(\lambda + m))$.

3.2 Supporting New Function Classes

In Sections 3.2, 3.2, and 3.2, we (1) present general transformations for obtaining FSS for new function classes from existing ones, (2) provide an extension of the improved DPF construction from the previous section to support the more general class of interval functions with minimal increase in key size, and (3) extend further to the case of *many* parties, where security is required to hold against coalitions of parties.

General Transformations We begin by describing a number of general transformations to convert one or more existing function secret sharing schemes into a new FSS scheme supporting a modified class of functions. The important metrics to maintain are the key size and computation time of the modified scheme, as a function of the original(s). Slightly

abusing notation, we denote by $\mathsf{size}(\mathcal{F})$ and $\mathsf{time}(\mathcal{F})$ the corresponding values for the key size and computation time for the FSS scheme for $\mathcal{F}$ (where the FSS scheme being referred to is clear from context).
Due to space constraints, we provide here only a brief summary of the relevant closure properties, and defer their corresponding constructions and proofs to the full version of the paper.

1. **Including the Zero Function:** $\mathcal{F} \to \mathcal{F} \cup \{0\}$.
For any FSS scheme for class $\mathcal{F}$, there exists a FSS scheme for the class $\mathcal{F}$ together with the all 0s function, $0(x) = 0\ \forall x$. It holds that $\mathsf{size}(\mathcal{F} \cup \{0\}) = \mathsf{size}(\mathcal{F})$, $\mathsf{time}(\mathcal{F} \cup \{0\}) = \mathsf{time}(\mathcal{F})$.
2. **Pre-composition with Arbitrary Function:** $(\mathcal{F}, g) \to \mathcal{F} \circ g$.
For any FSS scheme for function class $\mathcal{F} = \{f : \mathbb{G}_1 \to \mathbb{G}\}$, and arbitrary fixed public function $g : \mathbb{G}_2 \to \mathbb{G}_1$, there exists an FSS scheme for class $\mathcal{F} \circ g := \{f \circ g : \mathbb{G}_2 \to \mathbb{G} | f \in \mathcal{F}\}$, (where functions in $\mathcal{F} \circ g$ are described as the pair (f, g)). The resulting key size is equal to $|g| + \mathsf{size}(\mathcal{F})$, and the computation time is $|g| + \mathsf{time}(\mathcal{F})$.
This transformation extends to the case where the choice of function g may be made *dependent* on the secret function f, as long as the corresponding distribution of g is computationally indistinguishable from one independent of f. For example, g may consist of an encryption of some portion of f; indeed, such an approach can be used to bootstrap an FSS scheme for NC^1 to one supporting all $P/poly$, making use of fully homomorphic encryption (see Section 4.3).
3. **Post-composition with Linear Function:** $(\mathcal{F}, L) \to L \circ \mathcal{F}$.
For any FSS for function class $\mathcal{F} = \{f : \mathbb{G}_1 \to \mathbb{G}\}$ and for any fixed linear function $L : \mathbb{G} \to \mathbb{G}_0$, there exists a FSS scheme for class $L \circ \mathcal{F} := \{L \circ f | f \in \mathcal{F}\}$ of functions from $\mathbb{G} \to \mathbb{G}_0$ (where functions $(L \circ f) \in L \circ \mathcal{F}$ are described by the pair (L, f)). The resulting scheme satisfies $\mathsf{size}(L \circ \mathcal{F}) = \mathsf{size}(\mathcal{F}) + |L|$ and $\mathsf{time}(L \circ \mathcal{F}) = \mathsf{time}(\mathcal{F}) + |L|$.
4. **Linear Combination of FSSes:** $(\mathcal{F}, \mathcal{G}) \to \mathcal{F} + \mathcal{G}$.
Given FSS schemes for families $\mathcal{F}, \mathcal{G}$ taking $\mathbb{G}_1 \to \mathbb{G}$, there exists an FSS scheme for class $\mathcal{F} + \mathcal{G} := \{f \oplus g | f \in \mathcal{F}, g \in \mathcal{G}\}$, with key size equal to $\mathsf{size}(\mathcal{F} + \mathcal{G}) = \mathsf{size}(\mathcal{F}) + \mathsf{size}(\mathcal{G})$ and evaluation time $\mathsf{time}(\mathcal{F} + \mathcal{G}) = \mathsf{time}(\mathcal{F}) + \mathsf{time}(\mathcal{G})$.
5. **Union of Function Families:** $(\mathcal{F}_1, \mathcal{F}_2) \to \mathcal{F}_1 \cup \mathcal{F}_2$.
Given FSS schemes for families $\mathcal{F}, \mathcal{G}$, there exists an FSS scheme for the class $\mathcal{F} \cup \mathcal{G}$, with key size and time complexities as in Transformation 4 (combining with Transformation 1).
6. **FSS for Small Function Classes:** Arbitrary $\mathcal{F}$, with $\mathsf{time}(\mathcal{F}) \sim |\mathcal{F}|$, but short keys. For *any* class of functions $\mathcal{F}$ with some canonical indexing, and a DPF (i.e., FSS for class of point functions) with domain size $|\mathcal{F}|$, there exists an FSS scheme for $\mathcal{F}$ with computation time $O\left(|\mathcal{F}| \cdot \mathsf{time}(DPF) \cdot \max_{f \in \mathcal{F}} |f|\right)$ and key size $\mathsf{size}(DPF)$.

We describe useful function classes supported via combinations of the above transformations.

1. NC^0 functions.
For each constant depth $d \in \mathbb{N}$ and input/output bit-lengths n, m, by Transformation 6, we obtain an FSS scheme supporting the class $\mathcal{C}_d$ of

depth-d boolean circuits with input $\{0,1\}^n$, output $\{0,1\}^m$, and fan-in 2. The important observation is that we may secret share the entire circuit C by independently sharing m separate 1-bit-output sub-circuits (which each has $O(n^{2^d})$ possibilities) instead of separately treating all possible $m^{O(n^{2^d})}$ values for all of C.
Plugging in the state-of-the-art DPF instantiations (as given in Section 3.1), the resulting (server-side) runtime of the scheme is $\mathsf{time}(\mathcal{C}_d) \in O(\lambda n^{2^d} m)$, and the key size is $O(\lambda m \log n)$, where λ is the seed length for the underlying pseudorandom generator, and the hidden constants include a factor of 2^d.

2. Constant-conjunction search queries.
As a consequence of Transformation 6, together with the best known DPF instantiations (given in Section 3.1) with key size $O(\lambda n)$ for domain size 2^n and PRG seed length λ, we obtain an FSS scheme for the class $Match_\ell$ of data-matching functions, for a constant number of data entries ℓ, where each of which may take one of polynomially many $|\mathbb{G}_1| \in n^{O(1)}$ possible values. That is, for canonical nonzero element $g \in \mathbb{G}$,

$$Match_\ell = \Big\{ f_{S,v} : \mathbb{G}_1^n \to \mathbb{G} \Big\}_{\substack{S \subset [n], \\ |S| \leq \ell, \\ v \in \mathbb{G}_1^\ell}} \text{ s.t. } f_{S,v}(x) = \begin{cases} g & \text{if } x_i = v_i \ \forall i \in S \\ 0 & \text{else} \end{cases}.$$

Indeed, the class $Match_\ell$ contains $\binom{n}{\ell}|\mathbb{G}_1|^\ell \in O(n^\ell |\mathbb{G}_1|^\ell)$ different functions. Thus, for $N := (n|\mathbb{G}_1|)^\ell$, we obtain a FSS scheme supporting $Match_\ell$ with evaluation time $O(\lambda N \log N)$ and key size $O(\lambda \log N)$. For the case of $|\mathbb{G}_1| \in O(1)$, these correspond to runtime $O(\lambda n^\ell \ell \log n)$ and key size $O(\lambda \ell \log n)$.

3. Interval functions: Black-box from DPF.
The class of interval functions consists of those functions $f_{a,b}$ which output a fixed element $g \in \mathbb{G}$ precisely for inputs x that lie within the interval $a < x < b$, and $0 \in \mathbb{G}$ otherwise.

$$\mathcal{F}_n^{int} = \Big\{ f_{(a,b)} : \{0,1\}^n \to \mathbb{G} \Big\}_{\substack{0 \leq a \\ \leq b < 2^n}}, \text{ where } f_{(a,b)}(x) = \begin{cases} g & a < x < b \\ 0 & \text{else} \end{cases}.$$

Lemma 1. *Based on any DPF (i.e., FSS scheme for the class of multi-bit point functions) with key size s, there exists an FSS scheme for family $\mathcal{F}_n^{int}$, with key sizes $O(sn)$.*

Intuitively, we express the condition $x < a$ as the disjunction of (up to) n mutually inconsistent exact *prefix-matching* conditions, such that an element x is less than a precisely if it contains exactly one the prefixes. (Viewing the target value a as a path down a binary tree, this amounts to the sequence of (up to) n prefixes that agree with a up to some level i, but then continue to 0 at level $i+1$ whereas a continues to 1). We thus attain the desired FSS as a linear combination of n DPFs, each acting on a *prefix* of the input x (using Transformations 2 and 4).

Two-key FSS for Comparison and Interval Functions We show efficient constructions of FSS for the family $\mathcal{F}_n^<$ of all comparison functions from $\{0,1\}^n$ to some finite group $\mathbb{G}$. The class of comparison functions consists of those functions $f_{a,g}$ which output a fixed element $g \in \mathbb{G}$ for inputs x that lie within the interval $0 \leq x < a$, and $0 \in \mathbb{G}$ otherwise.

$$\mathcal{F}_n^< = \left\{ f_{a,g} : \{0,1\}^n \to \mathbb{G} \right\}_{0 \leq a < 2^n}, \text{ where } f_{a,g}(x) = \begin{cases} g & x < a \\ 0 & \text{else} \end{cases}.$$

Note that (by Transformation 4 above), supporting comparison functions also directly yields FSS for interval functions, with a factor of 2 overhead. We describe a two-key construction which is a natural extension of the two-party DPF construction in Algorithms 1 and 2. The key size of this construction is larger by an additive factor of $n \log |\mathbb{G}|$ compared to the key size of the DPF construction.
The scheme for comparison functions has a similar structure to the scheme for DPF. Again, each of the keys k_0, k_1 generated by $\mathsf{Gen}^<(1^\lambda, a, g)$ represents a binary tree of depth n, and $\mathsf{Eval}^<(\beta, k_\beta, x)$ traverses the tree defined by k_β to the leaf $x = x_1, \ldots, x_n$.
However, there are several key differences between the scheme for comparison functions and the DPF scheme. First, the objects in each node of the tree are *group elements*, generalizing the approach in the DPF scheme. In addition, similarly to the DPF scheme, when the path to x diverges from the path to a, if $x \geq a$ then the sum of the two group elements generated by $\mathsf{Eval}^<(0, k_0, x)$ and $\mathsf{Eval}^<(1, k_1, x)$ is 0 for any node from the point of divergence to the leaf. However, if $x < a$ then the sum of the two group elements in every node is g. Finally, the current Gen algorithm returns correct keys with probability 1.

Notation 3 *Let $\mathbb{G}$ be an abelian group with group operation $+$ (while $\oplus$ denotes the exclusive-or of bits), let $0 \in \mathbb{G}$ denote the identity element, let $g \in \mathbb{G}$ and let $-g$ denote the inverse of g in the group. Let $e_a \cdot g$ denote a sequence of $2^{|a|}$ elements in $\mathbb{G}$ such that the element at location a is g and all other elements in the sequence are the identity element. We assume that the length of e_a is determined by the domain of a.*

Notation 4 *Let $\mathbb{G}$ be a group, let $g \in \mathbb{G}$ and let $b \in -1, 0, 1$. We denote by $g \cdot b$ a group element that is the identity unit 0 if $b = 0$, is equal to g if $b = 1$ and is equal to $-g$ if $b = -1$. Let $c_a \cdot g$ be a sequence in of $2^{|a|}$ elements with g in every location a' such that $a' < a$ and 0 in every other location. We assume that the length of $c_a \cdot g$ is determined by the domain of a.*

Notation 5 *Let $E_{p,q}$ ($O_{p,q}$) be the set of all $p \times q^{p-1}$ arrays over the set $\{0, \ldots, q-1\}$ such that the sum of elements in every column is 0 modulo q (1 modulo q) and every column appears exactly once in the array.*

Algorithm 5 $\mathsf{Gen}^{<}(1^\lambda, a, g)$

```
1: Let $G : \{0,1\}^\lambda \longrightarrow \{0,1\}^{2\lambda+2\log|\mathbb{G}|+2}$ be a PRG.
2: Choose three random seeds $S^0_{a_1}[1], S^1_{a_1}[1], S^0_{\neg a_1}[1] \in \{0,1\}^\lambda$ and set $S^1_{\neg a_1}[1] \leftarrow S^0_{\neg a_1}[1]$.
3: Choose random bits $T^\beta_\alpha[1]$, $\alpha, \beta \in \{0,1\}$, subject to $T^0_{a_1}[1] \neq T^1_{a_1}[1]$ and $T^0_{\neg a_1}[1] = T^1_{\neg a_1}[1]$.
4: Choose random elements $V^\beta_\alpha[1] \in \mathbb{G}$, $\alpha, \beta \in \{0,1\}$, subject to $V^0_{a_1}[1] + (-V^1_{a_1}[1]) = 0$ and $V^0_{\neg a_1}[1] + (-V^1_{\neg a_1}[1]) = g \cdot a_1$.
5: for $i = 1$ to $n-1$ do
6:     Let $G(S^\beta_{a_i}[i]) = s^\beta_0 || s^\beta_1 || t^\beta_0 || t^\beta_1 || v^\beta_0 || v^\beta_1$, where $s^\beta_\alpha \in \{0,1\}^\lambda$, $t^\beta_\alpha \in \{0,1\}$ and $v^\beta_\alpha \in \mathbb{G}$ for $\alpha, \beta = 0, 1$.
7:     Randomly choose $cs_{0,a_{i+1}}, cs_{1,a_{i+1}} \in \{0,1\}^\lambda$.
8:     Randomly choose $cs_{0,\neg a_{i+1}}, cs_{1,\neg a_{i+1}} \in \{0,1\}^\lambda$ s.t. $\bigoplus_{\beta=0}^{1}(cs_{\beta,\neg a_{i+1}} \oplus s^\beta_{\neg a_{i+1}}) = 0$.
9:     Randomly choose $ct_{0,a_{i+1}}, ct_{1,a_{i+1}} \in \{0,1\}$ s.t. $\bigoplus_{\beta=0}^{1}(ct_{\beta,a_{i+1}} \oplus t^\beta_{a_{i+1}}) = 1$.
10:    Randomly choose $ct_{0,\neg a_{i+1}}, ct_{1,\neg a_{i+1}} \in \{0,1\}$ s.t. $\bigoplus_{\beta=0}^{1}(ct_{\beta,\neg a_{i+1}} \oplus t^\beta_{\neg a_{i+1}}) = 0$.
11:    Randomly choose $cv_{0,a_{i+1}}, cv_{1,a_{i+1}} \in \mathbb{G}$ s.t. $\sum_{\beta=0}^{1}(cv_{\tau,a_{i+1}} + v^\beta_{a_{i+1}}) \cdot (-1)^\beta = 0$, for $\tau = T^\beta_{a_i}[i]$.
12:    Randomly choose $cv_{0,\neg a_{i+1}}, cv_{1,\neg a_{i+1}} \in \mathbb{G}$ s.t. $\sum_{\beta=0}^{1}(cv_{\tau,\neg a_{i+1}} + v^\beta_{\neg a_{i+1}}) \cdot (-1)^\beta = g \cdot a_{i+1}$, for $\tau = T^\beta_{\neg a_i}[i]$.
13:    Set $CW_\beta[i] \leftarrow cs_{\beta,0} || cs_{\beta,1} || ct_{\beta,0} || ct_{\beta,1} || cv_{\beta,0} || cv_{\beta,1}$ for $\beta = 0, 1$.
14:    Set $S^\beta_\alpha[i+1] \leftarrow s^\beta_\alpha \oplus cs_{\tau,\alpha}$ for $\tau = T^\beta_{a_i}[i]$ and $\alpha, \beta \in \{0,1\}$.
15:    Set $T^\beta_\alpha[i+1] \leftarrow t^\beta_\alpha \oplus ct_{\tau,\alpha}$ for $\tau = T^\beta_{a_i}[i]$ and $\alpha, \beta \in \{0,1\}$.
16: end for
17: Set $k_\beta \leftarrow ((S^\beta_0[1], S^\beta_1[1], T^\beta_0[1], T^\beta_1[1], V^\beta_0[1], V^\beta_1[1]), (CW_0[1], CW_1[1], \ldots, CW_0[n-1], CW_1[n-1]))$.
18: Return $(k_0, k_1)$.
```

Algorithm 6 $\mathsf{Eval}^{<}(\beta, k_\beta, x)$

```
1: Let $G : \{0,1\}^\lambda \longrightarrow \{0,1\}^{2\lambda+2\log|\mathbb{G}|+2}$ be a PRG.
2: Let the binary representation of $x$ be $x = x_1, \ldots, x_n$.
3: Let $k_\beta = ((S^\beta_0[1], S^\beta_1[1], T^\beta_0[1], T^\beta_1[1], V^\beta_0[1], V^\beta_1[1]), (CW_0[1], CW_1[1], \ldots, CW_0[n-1], CW_1[n-1]))$.
4: Set $S^\beta \leftarrow S^\beta_{x_1}[1]$.
5: Set $T^\beta \leftarrow T^\beta_{x_1}[1]$.
6: Set $V^\beta \leftarrow V^\beta_{x_1}[1]$.
7: for $i = 2$ to $n$ do
8:     Parse $G(S^\beta)$ as $G(S^\beta) = s_0 || s_1 || t_0 || t_1 || v_0 || v_1$.
9:     Let $CW_{T^\beta}[i-1] = cs_{T^\beta,0} || cs_{T^\beta,1} || ct_{T^\beta,0} || ct_{T^\beta,1} || cv_{T^\beta,0} || cv_{T^\beta,1}$.
10:    Set $S^\beta \leftarrow s_{x_i} \oplus cs_{T^\beta,x_i}$.
11:    Set $T^\beta \leftarrow t_{x_i} \oplus ct_{T^\beta,x_i}$.
12:    Set $V^\beta \leftarrow V^\beta + (v_{x_i} + cv_{T^\beta,x_i})$.
13: end for
14: Return $V^\beta \cdot (-1)^\beta$.
```

We prove the correctness and security of $(\mathsf{Gen}^<, \mathsf{Eval}^<)$ via the following sequence of claims. Due to space limitations, we omit proofs of these claims, and refer the reader to the full version of this paper.

Lemma 2. *For every $n \in \mathbb{N}$, every $a, x \in \{0,1\}^n$, every finite abelian group $\mathbb{G}$, every $g \in \mathbb{G}$ and every $i, 1 \leq i \leq n$,*

1. *If $(x_1, \dots, x_i) = (a_1, \dots, a_i)$ then for $\beta = 0, 1$, the values S^β and T^β that $\mathsf{Eval}^<(\beta, k_\beta, x)$ computes are equal to the values $S^\beta_{a_i}[i]$ and $T^\beta_{a_i}[i]$ (respectively) that $\mathsf{Gen}^<(1^\lambda, a, g)$ computes; in addition, $T^0 \oplus T^1 = 1$.*
2. *If $(x_1, \dots, x_i) \neq (a_1, \dots, a_i)$ then $S^0 = S^1$ and $T^0 = T^1$.*

Building atop Lemma 2, we arrive at the desired correctness guarantee:

Proposition 1 (Correctness). *For every $n \in \mathbb{N}$, every $a, x \in \{0,1\}^n$, every finite abelian group $\mathbb{G}$ and every $g \in \mathbb{G}$, if $(k_0, k_1) \leftarrow \mathsf{Gen}^<(1^\lambda, a, g)$ then $\mathsf{Eval}^<(0, k_0, x) \oplus \mathsf{Eval}^<(1, k_1, x) = f^<_{a,g}(x)$.*

Theorem 6. *For every $n \in \mathbb{N}$, $a \in \{0,1\}^n$, every security parameter $\lambda \in \mathbb{N}$ and every finite abelian group $\mathbb{G}$, $(\mathsf{Gen}^<, \mathsf{Eval}^<)$ is a two-key FSS scheme for the family of comparison functions from $\{0,1\}^n$ to $\mathbb{G}$, with key size $O(n(\lambda + \log |\mathbb{G}|))$.*

We remark that, via a simple transformation, the constructed FSS for comparison functions also directly yields an FSS scheme for point functions over a general abelian group $\mathbb{G}$.

Corollary 1. *For every $n \in \mathbb{N}$, every security parameter $\lambda \in \mathbb{N}$ and every finite abelian group $\mathbb{G}$ there exists a two-key scheme for the family of point functions from $\{0,1\}^n$ to $\mathbb{G}$, without errors and with key size $O(n(\lambda + \log |\mathbb{G}|))$.*

Proof. A point function is a linear combination of two comparison functions. Specifically, $P_{a,g}(x) = f^<_{a+1}(x) + (-f^<_a(x))$, where $-f^<_a(x)$ is the inverse of $f^<_a(x)$ in $\mathbb{G}$. The corollary follows from Theorem 6 and the linear combination of FSS schemes in Section 3.2.

Extending to the Many-Party Setting We construct a scheme for the family of comparison functions from $\{0,1\}^n$ to an abelian group $\mathbb{G}$ that is secure against coalitions of all but one of the keys. The scheme, defined in Algorithms 7 and 8, has a similar structure to Algorithms 3 and 4.
There are several differences between the current scheme and the DPF scheme. The scheme for comparison functions is over $\mathbb{G}$ and the choice of arrays $A_{\gamma'}$ is from the sets $E_{p,q}$ and $O_{p,q}$, for $q = |\mathbb{G}|$, instead of choosing the arrays from E_p or O_p. The correction words, $cw_1, \dots, cw_\nu$, are chosen in a different way in line 6 of Algorithm 7 and additional group elements, $v_1, \dots, v_\nu$, are used in line 7 of Algorithm 7 and line 6 of Algorithm 8. The reason for the differences in $cw_1, \dots, cw_\nu$ and $v_1, \dots, v_\nu$ is that $f^<_{a,g}(x) = g$ for any $x < a$, while $P_{a,b}(x) = 0$ for any $x < a$.

Theorem 7. *For every security parameter $\lambda \in \mathbb{N}$, every $n, p \in \mathbb{N}$, every abelian group $\mathbb{G}$, $|\mathbb{G}| = q$, every $a, x \in \{0,1\}^n$ and every $g \in \mathbb{G}$, the pair of algorithms $(\mathsf{Gen}^p, \mathsf{Eval}^p)$ is an FSS scheme for the family of all comparison functions from $\{0,1\}^n$ to $\mathbb{G}$, such that* Gen *outputs p keys $(k_1, \ldots, k_p)$, the scheme is secure against any coalition of at most $p-1$ keys and the key size is $O(2^{n/2} \cdot q^{(p-1)/2} \log q)$.*

Corollary 2. *For any abelian group $\mathbb{G} = \mathbb{G}_1 \times \ldots \times \mathbb{G}_r$, such that $|\mathbb{G}_i| = q_i$ for $i = 1, \ldots, r$, there exists an FSS scheme for the family of comparison functions from $\{0,1\}^n$ to $\mathbb{G}$ that generates p keys and is secure against coalitions of up to $p-1$ keys with key size $O(2^{n/2} \cdot q^{(p-1)/2} \sum_{i=1}^{p} \log q_i)$. This result is obtained by running $(\mathsf{Gen}^p, \mathsf{Eval}^p)$ separately on each component $\mathbb{G}_i$.*

Algorithm 7 $\mathsf{Gen}^p(1^\lambda, a, g)$

```
1: Let $G : \{0,1\}^\lambda \longrightarrow \mathbb{G}^\mu$ be a PRG ($\mu$ is defined in line 2).
2: Let $\mu \leftarrow \lceil 2^{n/2} \cdot q^{(p-1)/2} \rceil$ and let $\nu \leftarrow \lceil 2^n/\mu \rceil$.
3: Regard $a$ as a pair $a = (\gamma, \delta)$, $\gamma \in \{0,1\}^\nu, \delta \in \{0,1\}^\mu$.
4: Choose $\nu$ random arrays $A_1, \ldots, A_\nu$, s.t. $A_\gamma \in O_{p,q}$ and $A_{\gamma'} \in E_{p,q}$ for all $\gamma' \neq \gamma$.
5: Choose $\nu \cdot q^{p-1}$ random seeds $s_{1,1}, \ldots, s_{\nu,q^{p-1}} \in \{0,1\}^\lambda$.
6: Randomly choose $cw_1, \ldots, cw_{q^{p-1}} \in \mathbb{G}^\mu$ s.t. $\sum_{j=1}^{q^{p-1}} (cw_j + G(s_{\gamma,j})) = c_\delta$.
7: Select $v_1, \ldots, v_p \in \mathbb{G}^\nu$ randomly s.t. $\sum_{i=1}^{p} v_i = c_\gamma \cdot g$.
8: If $A_{\gamma'}[i,j] \neq 0$ set $\sigma_{i,\gamma',j} \leftarrow (s_{\gamma',j}, A_{\gamma'}[i,j])$, otherwise $\sigma_{i,\gamma',j} \leftarrow (0,0)$, for all
   $1 \leq i \leq p, 1 \leq \gamma' \leq \nu, 1 \leq j \leq q^{p-1}$.
9: Set $\sigma_{i,\gamma'} \leftarrow (\sigma_{i,\gamma',1}|| \ldots ||\sigma_{i,\gamma',q^{p-1}})$, for all $1 \leq i \leq p, 1 \leq \gamma' \leq \nu$.
10: Set $\sigma_i = \sigma_{i,1}|| \ldots ||\sigma_{i,\nu}$ for $1 \leq i \leq p$.
11: Let $k_i = (\sigma_i, v_i, cw_1, \ldots, cw_{q^{p-1}})$ for $1 \leq i \leq p$.
12: Return $(k_1, \ldots, k_p)$.
```

Algorithm 8 $\mathsf{Eval}^p(i, k_i, x)$

```
1: Let $G : \{0,1\}^\lambda \longrightarrow \mathbb{G}^\mu$ be a PRG ($\mu$ is defined in line 2).
2: Let $\mu \leftarrow \lceil 2^{n/2} \cdot q^{(p-1)/2} \rceil$ and let $\nu \leftarrow \lceil 2^n/\mu \rceil$.
3: Regard $x$ as a pair $x = (\gamma', \delta')$, $\gamma' \in \{0,1\}^\nu, \delta' \in \{0,1\}^\mu$.
4: Parse $k_i$ as $k_i = (\sigma_i, v_i, cw_1, \ldots, cw_{q^{p-1}})$.
5: Parse $\sigma_i$ as $\sigma_i = (s_{1,1}, A_1[i,1])|| \ldots ||(s_{1,q^{p-1}}, A_1[i, q^{p-1}])|| \ldots ||(s_{\nu,q^{p-1}}, A_\nu[\nu, q^{p-1}])$.
6: Let $y_i \leftarrow v_i[\gamma'] + \sum_{\substack{1 \leq j \leq q^{p-1} \\ A_{\gamma'}[i,j] \neq 0}} A_{\gamma'}[i,j] \cdot (cw_j + G(s_{\gamma',j}))$.
7: Return $y_i[\delta']$.
```

Proposition 2 (Correctness). *For every security parameter $\lambda \in \mathbb{N}$, every $n, p \in \mathbb{N}$, every abelian group $\mathbb{G}$, every $a, x \in \{0,1\}^n$ and every $g \in \mathbb{G}$, if $(k_1, \ldots, k_p) \leftarrow \mathsf{Gen}^p(1^\lambda, a, g)$ then $\sum_{i=1}^{p} \mathsf{Eval}^p(i, k_i, x) = f^{<}_{a,g}(x)$.*

3.3 General FSS from Obfuscation

In this section, we provide general positive constructions of FSS based on program obfuscation. We first obtain FSS schemes for *P/poly* given access to a program obfuscator that satisfies a *virtual black-box (VBB)* notion of security [3]. We then build on top of recent advances in *indistinguishability obfuscation ($i\mathcal{O}$)* [3, 22] to demonstrate a similar conclusion from $i\mathcal{O}$ with sub-exponential hardness.
In particular, building atop recent candidate obfuscation constructions, these provide us with heuristic constructions of FSS for any efficiently computable function class of choice. Further, it yields provably secure solutions within idealized models, for which secure constructions of VBB obfuscation have been constructed [9, 2], e.g. in the generic multilinear map model, or in settings with secure hardware.
For purposes of space limits, we describe only the high-level intuition and defer complete constructions and proofs of security to the full version.

General FSS from Virtual Black Box (VBB) Obfuscation

Proposition 3. *Assume the existence of an ideal virtual black-box obfuscation oracle for P/poly, and the existence of one-way functions. Then there exists an FSS scheme supporting P/poly.*

Intuitively, the FSS construction works by obfuscating (1) a pseudorandom function (PRF) F_s for one party, and (2) $(C - F_s)$ for the desired circuit C for the second party. The VBB property enables a party's key to be simulated given black-box access to the underlying program, which can in turn be simulated (by the security of the PRF) by a truly random sequence of outputs.

General FSS From Sub-Exponential $i\mathcal{O}$ Our construction relies on a recent work of Canetti *et al.* [13] which demonstrates that sub-exponential $i\mathcal{O}$ implies a notion of *probabilistic $i\mathcal{O}$ ($pi\mathcal{O}$)*. Loosely, $pi\mathcal{O}$ converts a randomized program into a deterministic obfuscated program, and provides the guarantee that it is hard to distinguish obfuscations of two (randomized) circuits whose output distributions at each input are computationally indistinguishable, possibly in the presence of auxiliary input. We refer the reader to [13] for a full definition.

Theorem 8. *Assume the existence of sub-exponentially secure indistinguishability obfuscation and sub-exponentially secure one-way functions. Then there exists an FSS scheme supporting P/poly.*

The construction makes use of a $pi\mathcal{O}$-obfuscated (randomized) program P that takes as input x, samples a random value R, and outputs *encryptions* of the values R and $f(x) - R$ for the secret function f, under two different hardcoded public keys (i.e., $\mathsf{Enc}_{\mathsf{pk}_A}(R)$ and $\mathsf{Enc}_{\mathsf{pk}_B}(f(x) - R)$), as described in Figure 1. Recall that while this program P is randomized, its $pi\mathcal{O}$-obfuscation $\tilde{P}$ is a *deterministic* circuit. A party's FSS key for $f \in \mathcal{F}$ will consist of this obfuscated program $\tilde{P}$, together with *one* of the secret keys sk_A or sk_B. To evaluate his FSS share on an input x, the party runs $\tilde{P}(x)$, and decrypts his corresponding output. We remark that (sub-exponentially) secure public-key encryption (PKE) is implied by (sub-exponentially) secure indistinguishability obfuscation together with (sub-exponentially) secure one-way functions [43].

```
Program $\mathsf{FSS}_{f,\mathsf{pk}_A,\mathsf{pk}_B}$
Hardcoded: $f \in \mathcal{F}$, public keys $\mathsf{pk}_A, \mathsf{pk}_B$.
Input: $x \in \{0,1\}^n$. Randomness: $R, r_A, r_B$.
  1. Encrypt $R$ under $\mathsf{pk}_A$, as $\hat{y}_A \leftarrow \mathsf{Enc}_{\mathsf{pk}_A}(R; r_A)$.
  2. Encrypt $f(x) - R$ under $\mathsf{pk}_B$, as $\hat{y}_B \leftarrow \mathsf{Enc}_{\mathsf{pk}_B}(f(x) - R; r_B)$.
  3. Output $(\hat{y}_A, \hat{y}_B)$.
```

Fig. 1: Real program obfuscated in $\mathsf{Gen}(1^\lambda, f)$.

Correctness of the scheme follows by the correctness of the encryption and the *pi*$\mathcal{O}$: since the original program P outputs value pairs $(\hat{y}_A, \hat{y}_B)$ for which $\mathsf{Dec}_{\mathsf{sk}_A}(\hat{y}_A) + \mathsf{Dec}_{\mathsf{sk}_B}(\hat{y}_B) = f(x)$, the same property (which is efficiently testable given auxiliary input $\mathsf{sk}_A, \mathsf{sk}_B$) must hold for the outputs of $\tilde{P}$. By the security of the PKE, a party learns nothing from the second encrypted output, and thus his own decrypted shares (either R or $f(x) - R$) appear indistinguishable from random values. This is formalized in the proof by replacing the obfuscated program $\tilde{P}$ with an obfuscation of a fake program which instead outputs $\mathsf{Enc}_{\mathsf{pk}_A}(R)$ and $\mathsf{Enc}_{\mathsf{pk}_B}(R')$ for a second *independent* random value R'.

4 Relation to Other Primitives

In this section, we explore the relation between FSS and other cryptographic primitives. We first demonstrate in Section 4.1 that once the supported function class $\mathcal{F}$ becomes reasonably rich, each share of function $f \in \mathcal{F}$ must be a *pseudorandom function.* This holds in particular for the special case of point functions. We next provide evidence in Section 4.2 that achieving FSS for certain function classes (beginning as low as AC^0) is likely to require cryptographic tools heavier than one-way functions or even oblivious transfer. This is done by showing that such FSS schemes imply low-communication general secure computation protocols that rely on *reusable* preprocessing. Such protocols are currently only achievable using stronger cryptographic primitives, namely somewhat-homomorphic encryption or reusable garbled circuits. Finally, in Section 4.3 we show that, assuming fully homomorphic encryption (FHE) with decryption in NC^1 (as is the case for nearly all existing constructions, e.g. [8, 25, 10]), FSS for general functions is implied by the existence of FSS for NC^1.

4.1 Key Functions are Pseudorandom Functions

Parties' keys in the FSS each define their own *function*, taking inputs x to output shares $\mathsf{Eval}(b, k_b, x)$. This function serves as one piece of

the secret function being shared. A natural question is: what can we say about these functions? Can they have any sort of structure? Or, does the security property of the FSS together with the linearity of the output decoding procedure directly enforce a particular structure on the output share functions themselves?
We show that, in fact, if the supported class $\mathcal{F}$ is sufficiently rich, in the sense that it "efficiently spans" the whole function space, then it must be that the parties' output share functions $\mathsf{Eval}(b, k_b, x)$ themselves are *pseudorandom functions (PRFs).* We formalize this condition on $\mathcal{F}$ as "poly-spanning."

Definition 4. *A family of functions $\mathcal{F} = \{f : \mathbb{G}_n \to \mathbb{G}_m\}$ is said to be* poly-spanning *if for each polynomial $p(n)$ there exists a polynomial $q(n)$ and efficient procedure $P : (\{0,1\}^n \times \{0,1\}^m)^{p(n)} \to \mathcal{F}^{q(n)}$ mapping $p(n)$ pairs of input-output assignments to a collection of $q(n)$ functions from $\mathcal{F}$, with $P\left((x_i, y_i)_{i\in[p(n)]}\right) = (f_j)_{j\in[q(n)]}$ such that the function $f' := \sum_{j\in[q(n)]} f_j$ satisfies $f'(x_i) = y_i$ for every $i \in [p(n)]$.*

Remark 3 (Examples of poly-spanning function families).

- **Multi-bit Point Functions.** The class of functions $\{f_{x^*,y^*}\}$ over $x^* \in \{0,1\}^n, y^* \in \{0,1\}^m$ where $f_{x^*,y^*}(x) = y^*$ if $x = x^*$ and 0 otherwise. Indeed, the desired procedure P is simply given by $P\left((x_i, y_i)_{i\in[p(n)]}\right) = (f_{x_i,y_i})_{i\in[p(n)]}$.
- **Comparison Functions.** The class of comparison functions $\mathcal{F}_n^{\leq}$. Indeed, the desired procedure P is given as follows:

```
1: Initialize $S \leftarrow \emptyset$.
2: Sort inputs $x_1, \ldots, x_{p(n)} \in [2^n]$ as $x'_1 \leq \ldots \leq x'_{p(n)}$. Denote their
   outputs as $y'_i$.
3: for $i = p(n)$ to 1 do
4:     if $y'_i \neq y'_{i+1}$ (where $y'_{p(n)+1} := 0$) then
5:         Include the new function $f^{\leq}_{x'_i}$ (to flip the output of the
   sum): $S \leftarrow S \cup \{f^{\leq}_{x'_i}\}$
6:     end if
7: end for
8: return $S$
```

We now introduce notation for the output share function that we study.

Definition 5. *Let $(\mathsf{Gen}, \mathsf{Eval})$ be an FSS scheme w.r.t. function class $\mathcal{F}$. Then for each $f \in \mathcal{F}$ and $b \in \{0,1\}$, we denote by $\mathsf{OutputShare}_{f,b}$ the function family $\{\mathsf{Eval}(b, k_b, \cdot)\}_{k_b}$ defined by sampling and evaluation procedures:*

- *Sample: Outputs a key k_b, where $(k_0, k_1) \leftarrow \mathsf{Gen}(1^\lambda, f)$.*
- *Evaluate: On input x, computes $\mathsf{Eval}(b, k_b, x)$.*

Theorem 9. *Let $(\mathsf{Gen}, \mathsf{Eval})$ be a FSS scheme (as per Definition 2) w.r.t. a* poly-spanning *function class $\mathcal{F}$. Then for every $f \in \mathcal{F}$ and every $b \in \{0,1\}$, the function family $\mathsf{OutputShare}_{f,b}$ as given in Definition 5 is a PRF family (against nonuniform adversaries).*

Proof. Intuitively, we first show that oracle access to a randomly sampled party key function $\mathsf{OutputShare}_{f,b}$ (over the randomness of Gen) must be computationally indistinguishable from oracle access to the distribution $(\mathsf{OutputShare}_{f,b}+\sum_{i\in S} f_i)$ for any fixed polynomial-size subset S of functions $f_i \in \mathcal{F}$ in the supported function class. Then, we show that if $\mathcal{F}$ is poly-spanning, then for any possible PRF distinguishing adversary $\mathcal{A}$, we can fool this $\mathcal{A}$, guaranteeing that he *cannot* succeed in distinguishing from a random function, with an appropriate carefully tailored choice of functions $\{f_i\}_{i\in S} \subset \mathcal{F}$.
We defer the full proof of Theorem 9 to the full version of this paper.

4.2 Barriers Toward FSS for Expressive Function Classes

We now turn to exploring likely *barriers* in constructing FSS for certain function classes based on lightweight cryptographic tools. Our results in this section take the following form: Assume there exists FSS for a class of functions containing $\mathcal{F} \circ \mathsf{Dec}$, where $\mathcal{F}$ is some function class and Dec corresponds to the complexity of decryption of a symmetric-key encryption scheme. Then there exists a particular form of highly communication-efficient secure computation for functions in $\mathcal{F}$, which is currently only known to exist based on *$\mathcal{F}$-homomorphic encryption*[7] or *reusable garbled circuits for $\mathcal{F}$*. In particular:

– At the high end, FSS for $P/poly$ implies a form of secure computation whose only known constructions rely on *fully homomorphic* encryption or *reusable* garbled circuits for $P/poly$. We conclude that FSS for $P/poly$ is likely to require heavy cryptographic machinery.
– At the low end, FSS for AC^0 in combination with any symmetric-key encryption scheme with decryption in AC^0 together imply a form of secure computation only currently known to exist based on existence of AC^0-homomorphic encryption or reusable garbled circuits for AC^0.
  In particular, symmetric-key encryption with decryption in AC^0 is implied by sub-exponential hardness of Learning Parity with Noise (LPN) [7]. However, despite significant efforts in the cryptographic community, it is unknown even how to build from this assumption collision resistant hashing, much less stronger primitives like homomorphic encryption that imply them [35]. Indeed, all proposed constructions to date of homomorphic encryption and reusable garbled circuits (even for the restricted class AC^0), such as those from [11, 10, 33], rely on Learning With Errors (LWE) [42] or similar lattice-based assumptions; a construction under weaker or significantly different assumptions such as LPN would be considered a major result. We conclude that FSS for AC^0 is unlikely to be achieved based on sub-exponential LPN (or any weaker) assumption alone.
  We contrast this conclusion with our construction of FSS for various strict subclasses of AC^0 in Section 3.2 based on *one-way functions.*

[7] That is, semantically secure encryption supporting *compact* homomorphic evaluation of the function class $\mathcal{F}$.

We now formalize the above discussion. Concretely, we demonstrate that FSS for a function class $\mathcal{F} \circ \mathsf{Dec}$ (formally defined below) yields a construction of exceedingly communication-efficient (semi-honest) secure multiparty computation (MPC) in the preprocessing model, for the function class $\mathcal{F}$. That is, given an offline setup phase independent of parties' inputs, the parties A, B can reuse this setup to achieve secure evaluation of a fixed $f \in \mathcal{F}$ on arbitrarily many input pairs $(x_1^A, x_1^B), (x_2^A, x_2^B), \ldots$ in the online phase with communication that depends *only on the size of the inputs and outputs* of f, and not on the size of f itself. To date, the only other known approaches to achieving MPC with this efficiency feature (even when allowing reusable preprocessing) rely on strong cryptographic tools: either fully *homomorphic encryption for* $\mathcal{F}$ (as in [23, 1]) or *reusable* garbled circuits for $\mathcal{F}$ (as in [33].[8])
Intuitively, the FSS enables communication efficiency as follows. Suppose we wish to achieve secure computation of a function $f \in \mathcal{F}$. In the offline phase, the parties A, B will each receive[9] a secret key $\mathsf{sk}_A, \mathsf{sk}_B$ for the symmetric key encryption scheme, and FSS keys of a function $\hat{f}_{\mathsf{sk}} \in \mathcal{F} \circ \mathsf{Dec}$ that depends on both sk_A and sk_B. This function $\hat{f}_{\mathsf{sk}}$ will take as input a pair of ciphertexts $(\hat{x}^A, \hat{x}^B)$, decrypts each with respect to the corresponding hardcoded secret key sk_A or sk_B, and then evaluates the function f on the resulting values. In the online phase, for each desired input pair (x_i^A, x_i^B), the parties exchange *encryptions* of their private inputs under their respective secret keys. They then use their FSS keys to compute output shares of $\hat{f}_{\mathsf{sk}}$ evaluated on input this pair of ciphertexts $(\hat{x}_i^A, \hat{x}_i^B)$. Finally, the computed output shares are exchanged, and the value of $\hat{f}_{\mathsf{sk}}(\hat{x}_i^A, \hat{x}_i^B)$ is reconstructed. By the correctness of the FSS scheme and the choice of $\hat{f}_{\mathsf{sk}}$, this will exactly allow the parties to compute the desired value $f(x_i^A, f_i^B)$. And by the security of the FSS and the encryption scheme, no additional information on the inputs will be revealed.

We now formalize these intuitions.

Remark 4 (MPC Security). Recall that MPC security is defined with respect to the real/ideal world paradigm. Very loosely, for every PPT adversary $\mathcal{A}$ in a real-world execution of the protocol, there exists a PPT simulator in the ideal-world execution (receiving only the function output(s)) who can consistently simulate the experiment output. We refer the reader to e.g. [46, 30] for a formal definition.

Definition 6 (Communication-Efficient Online MPC for $\mathcal{F}$). *It is said that* communication-efficient online MPC for the function class $\mathcal{F}$

[8] Loosely, the offline phase will result in one party receiving a reusable garbled circuit of f and the second will receive the information to generate garbled input labels; the offline phase will only require communication on order the size of the garbled input and output labels, and not the size of f itself.

[9] For simplicity, we treat the offline setup phase as correlated randomness generated and given to the two parties by some trusted source; in practice, this can be implemented by running a standard MPC protocol between the two parties to securely generate these values.

exists if for any $f \in \mathcal{F}$, there exists a distribution of correlated randomness (D_A, D_B), polynomial p, and a two-party protocol Π in the correlated randomness model such that, for any $\ell \in \mathbb{N}$, and any sequence of (possibly adaptively chosen) inputs $(x_1^A, x_1^B), \ldots, (x_\ell^A, x_\ell^B)$, the protocol Π achieves secure evaluation of f on the input pairs in the semi-honest model, with (online) communication complexity $O\big(\sum_{i=1}^{\ell} \big(|x_i^A| + |x_i^B| + |f(x_i^A, x_i^B)|\big) \cdot p(\lambda)\big)$, where λ is the security parameter. In particular, the online communication complexity is independent of the size of the description of f.

Definition 7. *For a given symmetric encryption scheme* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *and function class $\mathcal{F}$, we define the function class* $\mathcal{F} \circ \mathsf{Dec} := \{f \circ (\mathsf{Dec}_{\mathsf{sk}_A} \times \mathsf{Dec}_{\mathsf{sk}_B}) : f \in \mathcal{F}, \mathsf{sk}_A, \mathsf{sk}_B \in \mathit{Supp}(\mathsf{Gen}(1^k))\}$.

Theorem 10. *Assume the existence of symmetric-key encryption with decryption* Dec*, and FSS for* $\mathcal{F} \circ \mathsf{Dec}$ *(as in Definition 7). Then there exists communication-efficient online MPC for the class $\mathcal{F}$, as in Definition 6.*

Due to space limitations, we defer the proof of Theorem 10 to the full version of this paper.

Remark 5. We note that the proof of Theorem 10 does not rely directly on the linearity of the output decoding procedure of the FSS scheme. Rather, the same result holds identically for any output decoding function that still guarantees *function privacy* (to preserve security of the MPC) and *succinctness* (to maintain communication efficiency in the online phase).

We now address the implications of Theorem 10 to two specific function classes $F \circ \mathsf{Dec}$.

Corollary 3 (FSS for $P/poly$**).** *Assuming FSS for $P/poly$, there exists communication-efficient online MPC for all $P/poly$.*

Proof. By Theorem 9, FSS for $P/poly$ implies the existence of pseudorandom functions, which thus implies secure symmetric-key encryption with decryption in $P/poly$. The corollary hence follows directly from Theorem 10.

Corollary 4 (FSS for AC^0**).** *Assuming FSS for AC^0 and sub-exponential hardness of LPN, there exists communication-efficient online MPC for AC^0.*

Proof. Follows from Theorem 10 and [7].

4.3 Bootstrapping with Fully Homomorphic Encryption

We show that FSS schemes enjoy a convenient bootstrapping property, when paired with fully homomorphic encryption (FHE). Namely, assuming the existence of FHE with decryption in NC^1 (as is the case for essentially all existing constructions, e.g. [8, 25, 10]), then any FSS scheme supporting the class NC^1 directly implies an FSS for the class of *all* circuits, where the FSS key size grows with the size of the circuit being secret shared.[10]

Proposition 4. *Assuming the existence of fully homomorphic encryption with perfect correctness and decryption in NC^1, and FSS for NC^1, then there exists a secure FSS scheme for $P/poly$.*

Proof. Intuitively, the new FSS construction will work by sampling FSS keys in the underlying NC^1-supported scheme for the FHE *decryption* function $\mathsf{Dec}_{\mathsf{sk}}$ for random, secret sk, and additionally providing an *encryption* $\hat{C}$ of a description of the desired circuit $C \in P/poly$. To evaluate, the parties first homomorphically evaluate C on their input x using $\hat{C}$, and then use this evaluated ciphertext as the input to the FSS for $\mathsf{Dec}_{\mathsf{sk}}$. We defer the full proof of Proposition 4 to the full version of this paper.

Acknowledgement. We thank Nir Bitansky and Vinod Vaikuntanathan for helpful discussions and for pointing out the relevance of [13].

References

1. Gilad Asharov, Abhishek Jain, Adriana Lopez-Alt, Eran Tromer, Vinod Vaikuntanathan, Daniel Wichs: Multiparty Computation with Low Communication, Computation and Interaction via Threshold FHE. EUROCRYPT 2012: 483-501.
2. Boaz Barak, Sanjam Garg, Yael Tauman Kalai, Omer Paneth, Amit Sahai: Protecting Obfuscation against Algebraic Attacks. EUROCRYPT 2014: 221-238.
3. Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, Ke Yang: On the (im)possibility of obfuscating programs. J. ACM 59(2): 6 (2012).
4. O. Barkol, Y. Ishai and E. Weinreb, "On Locally Decodable Codes, Self-Correctable Codes, and t-Private PIR", *Algorithmica*, Volume 58, Number 4, pp. 831-859, 2010.
5. R. Beigel, L. Fortnow, and W. I. Gasarch, "A tight lower bound for restricted PIR protocols", *Computational Complexity* 15(1): 82-91, 2006.
6. A. Beimel, Y. Ishai, E. Kushilevitz and I. Orlov, "Share Conversion and Private Information Retrieval", *IEEE Conference on Computational Complexity 2012*, pp. 258-268, 2012.
7. Andrej Bogdanov, Chin Ho Lee: "On the depth complexity of homomorphic encryption schemes," Electronic Colloquium on Computational Complexity (ECCC) 2012/157, 2012.
8. Zvika Brakerski, Craig Gentry, Vinod Vaikuntanathan: (Leveled) fully homomorphic encryption without bootstrapping. ITCS 2012: pp. 309-325.

[10] Note that this growth in key size is necessary, as the size of the two parties' keys together with the complexity of the Eval' must match or exceed the circuit description size; thus circuits of arbitrary polynomial size cannot be supported by a fixed polynomial size key and Eval algorithm.

9. Zvika Brakerski, Guy N. Rothblum: Virtual Black-Box Obfuscation for All Circuits via Generic Graded Encoding. TCC 2014: pp. 1-25.
10. Zvika Brakerski, Vinod Vaikuntanathan: Lattice-based FHE as secure as PKE. ITCS 2014: pp. 1-12.
11. Z. Brakerski and V. Vaikuntanathan, "Efficient Fully Homomorphic Encryption from (Standard) LWE", *FOCS 2011*, pp. 97-106, 2011.
12. C. Cachin, S. Micali and Markus Stadler, "Computationally Private Information Retrieval with Polylogarithmic Communication", EUROCRYPT, pp. 402-414, 1999.
13. Ran Canetti, Huijia Lin, Stefano Tessaro, Vinod Vaikuntanathan: "Obfuscation of Probabilistic Circuits and Applications", *Cryptology ePrint Archive, Report 2014/882*, 2014.
14. B. Chor and N. Gilboa, "Computationally Private Information Retrieval", (STOC'97), pp. 304-313, 1997.
15. B. Chor, O. Goldreich, E. Kushilevitz and M. Sudan, "Private Information Retrieval", *Journal of the ACM* (JACM), Volume 45 Issue 6, pp. 965-981, 1998.
16. Alfredo De Santis, Yvo Desmedt, Yair Frankel, Moti Yung: How to share a function securely. STOC 1994: 522-533.
17. Y. Desmedt, "Society and Group Oriented Cryptography: A New Concept", (CRYPTO'87), pp. 120-127, 1987.
18. Y. Desmedt and Y. Frankel, "Threshold Cryptosystems", (CRYPTO'89), pp. 307-315, 1989.
19. Zeev Dvir, Sivakanth Gopi: 2-Server PIR with sub-polynomial communication. Electronic Colloquium on Computational Complexity (ECCC) 21: 94 (2014)
20. G. Di Crescenzo, T. Malkin, and R. Ostrovsky, "Single Database Private Information Retrieval Implies Oblivious Transfer", *EUROCRYPT 2000*, pp. 122-138, 2000.
21. K. Efremenko, "3-query locally decodable codes of subexponential length", (STOC'09), pp. 39-44, 2009.
22. Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, Brent Waters: Candidate Indistinguishability Obfuscation and Functional Encryption for all Circuits. FOCS 2013: 40-49.
23. C. Gentry "Fully Homomorphic Encryption Using Ideal Lattices", (STOC 2009), pp. 169-178, 2009.
24. C. Gentry and Z. Ramzan, "Single-Database Private Information Retrieval with Constant Communication Rate", *ICALP 2005*, pp. 803-815, 2005.
25. Craig Gentry, Amit Sahai, Brent Waters: Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based. CRYPTO (1) 2013: 75-92.
26. N. Gilboa and Y. Ishai. "Distributed point functions and their applications." In EUROCRYPT, pages 640-658, 2014.
27. O. Goldreich, "A Note on Computational Indistinguishability", *Inf. Process. Lett.*, volume 34, number 6, pp. 277-281. 1990
28. O. Goldreich, "Foundations of Cryptography: Basic Tools", Cambridge University Press, 2000.
29. O. Goldreich, S. Goldwasser and S. Micali, "How to construct random functions", *Journal of the ACM* (JACM) 33.4 pp. 792-807, 1986.

30. Oded Goldreich, Silvio Micali, Avi Wigderson: How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority. STOC 1987: 218-229.
31. Oded Goldreich, Rafail Ostrovsky: Software Protection and Simulation on Oblivious RAMs. J. ACM 43(3): 431-473 (1996)
32. Shafi Goldwasser, Silvio Micali: "Probabilistic Encryption", *J. Comput. Syst. Sci.* 28(2), pp. 270-299, 1984.
33. Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, Nickolai Zeldovich: Reusable garbled circuits and succinct functional encryption. STOC 2013: 555-564.
34. J. Hastad, R. Impagliazzo, L. Levin and M. Luby, "A Pseudorandom Generator from any One-way Function", SIAM J. Comput. 28(4), pp. 1364-1396, 1999.
35. Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky: Sufficient Conditions for Collision-Resistant Hashing. TCC 2005: 445-456
36. Yuval Ishai, Anat Paskin: Evaluating Branching Programs on Encrypted Data. TCC 2007: 575-594.
37. Bala Kalyanasundaram, Georg Schnitger: "The Probabilistic Communication Complexity of Set Intersection", SIAM J. Discrete Math. 5(4), pp. 545-557, 1992.
38. E. Kushilevitz and R. Ostrovsky, "Replication is NOT Needed: SINGLE Database, Computationally-Private Information Retrieval", (FOCS'97), pp. 364-373, 1997.
39. H. Lipmaa, "An Oblivious Transfer Protocol with Log-Squared Communication", (ISC'05), pp. 314-328, 2005.
40. R. Ostrovsky and V. Shoup, "Private information storage", STOC '97, pp. 294-303. ACM, 1997.
41. R. Ostrovsky and W. E. Skeith III, "Private Searching on Streaming Data", *J. Cryptology* 20(4): 397-430, 2007.
42. Oded Regev: "On lattices, learning with errors, random linear codes, and cryptography," (STOC'05), pp. 84-93, 2005.
43. Amit Sahai, Brent Waters: How to use indistinguishability obfuscation: deniable encryption, and more. STOC 2014: 475-484.
44. Shamir, A., "How to Share a Secret", *CACM,* Volume 22, Number 11, pp. 612-613, 1979.
45. S. Wehner and R. de Wolf, "Improved Lower Bounds for Locally Decodable Codes and Private Information Retrieval", *ICALP 2005*, pp. 1424-1436.
46. Andrew Chi-Chih Yao: Protocols for Secure Computations (Extended Abstract). FOCS 1982: 160-164.
47. S. Yekhanin, "Towards 3-query locally decodable codes of subexponential length", (STOC'07), pp. 266-274, 2007.