Function Secret Sharing: Improvements and Extensions

Elette Boyle
IDC Herzliya, Israel
elette.boyle@idc.ac.il

Niv Gilboa
Ben Gurion University, Israel
gilboan@bgu.ac.il

Yuval Ishai
Technion and UCLA
yuvali@cs.technion.ac.il

ABSTRACT

Function Secret Sharing (FSS), introduced by Boyle et al. (Eurocrypt 2015), provides a way for additively secret-sharing a function from a given function family $\mathcal{F}$. More concretely, an m-party FSS scheme splits a function $f : \{0,1\}^n \to \mathbb{G}$, for some abelian group $\mathbb{G}$, into functions $f_1, \ldots, f_m$, described by keys $k_1, \ldots, k_m$, such that $f = f_1 + \ldots + f_m$ and every strict subset of the keys hides f. A Distributed Point Function (DPF) is a special case where $\mathcal{F}$ is the family of point functions, namely functions $f_{\alpha,\beta}$ that evaluate to β on the input α and to 0 on all other inputs.

FSS schemes are useful for applications that involve privately reading from or writing to distributed databases while minimizing the amount of communication. These include different flavors of private information retrieval (PIR), as well as a recent application of DPF for large-scale anonymous messaging.

We improve and extend previous results in several ways:

- **Simplified FSS constructions.** We introduce a tensoring operation for FSS which is used to obtain a conceptually simpler derivation of previous constructions and present our new constructions.
- **Improved 2-party DPF.** We reduce the key size of the PRG-based DPF scheme of Boyle et al. roughly by a factor of 4 and optimize its computational cost. The optimized DPF significantly improves the concrete costs of 2-server PIR and related primitives.
- **FSS for new function families.** We present an efficient PRG-based 2-party FSS scheme for the family of *decision trees*, leaking only the topology of the tree and the internal node labels. We apply this towards FSS for multi-dimensional intervals. We also present a general technique for extending FSS schemes by increasing the number of parties.
- **Verifiable FSS.** We present efficient protocols for verifying that keys $(k_1^*, \ldots, k_m^*)$, obtained from a potentially malicious user, are consistent with some $f \in \mathcal{F}$. Such a verification may be critical for applications that involve private writing or voting by many users.

CCS'16, October 24 - 28, 2016, Vienna, Austria

ISBN 978-1-4503-4139-4/16/10...$15.00

DOI: http://dx.doi.org/10.1145/2976749.2978429

Keywords: Function secret sharing, private information retrieval, secure multiparty computation, homomorphic encryption

1. INTRODUCTION

In this work we continue the study of *Function Secret Sharing (FSS),* a primitive that was recently introduced by Boyle et al. [7] and motivated by applications that involve private access to large distributed data.

Let $\mathcal{F}$ be a family of functions $f : \{0,1\}^n \to \mathbb{G}$, where $\mathbb{G}$ is an abelian group. An m-party FSS scheme for $\mathcal{F}$ provides a means for "additively secret-sharing" functions from $\mathcal{F}$. Such a scheme is defined by a pair of algorithms $(\mathsf{Gen}, \mathsf{Eval})$. Given a security parameter and a description of a function $f \in \mathcal{F}$, the algorithm Gen outputs an m-tuple of *keys* $(k_1, \ldots, k_m)$, where each key k_i defines the function $f_i(x) = \mathsf{Eval}(i, k_i, x)$. The correctness requirement is that the functions f_i add up to f, where addition is in $\mathbb{G}$; that is, for any input $x \in \{0,1\}^n$ we have that $f(x) = f_1(x) + \ldots + f_m(x)$. The security requirement is that every strict subset of the keys computationally hides f. A naive FSS scheme can be obtained by additively sharing the entire truth-table of f. The main challenge is to obtain a much more efficient solution, ideally polynomial or even linear in the description size of f.

The simplest nontrivial special case of FSS is a *Distributed Point Function* (DPF), introduced by Gilboa and Ishai [18]. A DPF is an FSS for the family of point functions, namely functions $f_{\alpha,\beta} : \{0,1\}^n \to \mathbb{G}$ for $\alpha \in \{0,1\}^n$ and $\beta \in \mathbb{G}$, where the point function $f_{\alpha,\beta}$ evaluates to β on input α and to 0 on all other inputs. Efficient constructions of 2-party DPF schemes from any pseudorandom generator (PRG), or equivalently a one-way function (OWF), were presented in [18, 7]. This was extended in [7] to more general function families, including the family of *interval functions* $f_{[a,b]}$ that evaluate to 1 on all inputs x in the interval $[a, b]$ and to 0 on all other inputs. For $m \geq 3$, the best known PRG-based DPF construction is only quadratically better than the naive solution, with key size $\approx \sqrt{N}$, where $N = 2^n$ [7]. We consider here the case $m = 2$ by default.

On the high end, polynomial-time FSS schemes for arbitrary polynomial time functions are implied by indistinguishability obfuscation [7] and by variants of fully homomorphic encryption [7, 14]. In the present work we mainly consider PRG-based FSS schemes, which have far better concrete efficiency and are powerful enough for the applications we describe next.

FSS schemes are motivated by two types of applications: ones that involve privately *reading* from a database held by m servers, and ones that involve privately *writing* (or incrementing) an array which is secret-shared among m servers. In both cases, FSS can be used to minimize the communication complexity. We illustrate two concrete application scenarios below and refer the reader to Appendix A for a more detailed exposition.

For a typical "reading" application, consider the problem of 2-server Private Information Retrieval (PIR) [10, 8]. In the basic flavor of PIR, the two servers hold a database of N strings $(x_1, \ldots, x_N)$, and a client wishes to retrieve x_α without revealing α to either of the two servers. PIR in this setting can be implemented by having the client distribute the point function $f_{\alpha,1} : [N] \to \mathbb{Z}_2$ between the servers. Concretely, the client generates a pair of keys (k_1, k_2) which define additive shares f_1, f_2 of $f_{\alpha,1}$, and sends each key to a different server. On input k_i, server i sends back the sum $\sum_{j=1}^{N} x_j f_i(j)$, where each x_j is viewed as an element in $\mathbb{Z}_2^\ell$. The client can recover x_α by taking the exclusive-or of the two ℓ-bit strings it receives.

Still relying only on a standard DPF, this can be directly generalized to private search by keywords (returning the payload associated with a private ℓ-bit keyword), private search on streaming data [27, 15, 7], and more. FSS for interval functions can be used to privately search values in a secret range. Realizing similar private search functionalities using standard PIR protocols requires the use of suitable data structures, which incur a significant additional overhead in round complexity, storage, and cost of updates [9]. In general, FSS for a function family $\mathcal{F}$ can be used to efficiently perform searches defined by predicates from $\mathcal{F}$.

For a typical "writing" application, consider the following example from [7]. Suppose that we want to collect statistics on web usage of mobile devices without compromising the privacy of individual users, and while allowing fast collection of real-time traffic data for individual web sites. A DPF provides the following solution. An array of counters is additively shared between 2 servers. A client who visits URL α can now secret-share the point function $f = f_{\alpha,1}$ over a sufficiently large group $\mathbb{G} = \mathbb{Z}_M$ and each server i updates its shared entry of each URL α_j by locally adding $f_i(\alpha_j)$ to its current share of α_j. Note that the set of URLs α_j used to index entries of the array does not need to include the actual URL α visited by the client, and in fact it can include only a selected watchlist of URLs which is unknown to the client. A different "writing" application for DPF was proposed in the context the Riposte system for anonymous messaging [11]. In this system, messages from different clients are mixed by having each client privately write the message to a random entry in a distributed array.

1.1 Our Contribution

Motivated by applications of FSS, we continue the study of efficient constructions that can be based on any PRG. We improve and extend previous results from [7] in several directions.

Simplified FSS constructions. We introduce a conceptually simple "tensoring" operation for FSS, which we use both to rederive previous constructions and obtain some of the new constructions we describe next.

Improved 2-party DPF. We reduce the key size of the PRG-based DPF scheme of Boyle et al. roughly by a factor of 4 and optimize its computational cost. In an AES-based implementation, the key size of a DPF is equivalent to roughly a single AES key per input bit. We provide further optimizations for the case of DPF with a single-bit output and for reducing the computational cost of evaluating the DPF of the entire domain (as needed, for instance, in the PIR application described above). The optimized DPF can be used to implement 2-server PIR protocols in which the communication overhead is extremely small (e.g., roughly $2.5K$ bits are sent to each server for retrieving from a database with 2^{25} records) and the computation cost on the server side is typically dominated by the cost of computing the XOR of half the data items. More concretely, the additional computational cost of expanding the DPF key for an N-record database consists of roughly $N/64$ AES operations. In the case of private keyword search, retrieving the payload associated with an 80-bit keyword requires sending less than $10K$ bits. See Table 1 for more details on the concrete efficiency of our DPF construction and Appendix B for more details on the PIR application and a comparison with alternative approaches from the literature.

FSS for new function families. We present an efficient PRG-based 2-party FSS scheme for the family of *decision trees*, leaking only the topology of the tree (i.e., the shape of the graph) and the internal node labels (i.e., which input variable labels each node). Our construction hides the labels of edges and leaves. We apply this towards PRG-based FSS for multi-dimensional intervals, e.g., capturing conjunction queries or search restricted to a geographical region. We also present a general technique for extending the expressive power of FSS schemes by increasing the number of parties. Concretely, we show how to obtain FSS schemes for the family of all *products* of pairs of functions from two given families that are realized by FSS. This can be applied towards more efficient solutions for multi-dimensional intervals, though with a larger number of parties.

Verifiable FSS. In both types of applications of FSS discussed above, badly formed FSS keys can enable a malicious client to gain an unfair advantage. The effect of malicious clients can be particularly devastating in the case of "writing" applications, where a single badly formed set of keys can corrupt the entire data. We present efficient protocols for verifying that keys $(k_1^*, \ldots, k_m^*)$ are consistent with some $f \in \mathcal{F}$. Our techniques make black-box use of the underlying FSS scheme, and avoid the cost of general-purpose secure computation techniques. Our verification protocols make a novel use of sublinear verification techniques (including special-purpose linear sketching schemes and linear PCPs) and combine them with MPC protocols that exploit correlated randomness from an untrusted client for better efficiency. These techniques may be applicable beyond the context of verifiable FSS.

Organization. Useful definitions appear in Section 2. Several FSS constructions, including the tensor product generalization, optimized DPF and evaluating a DPF on the entire domain are presented in Section 3. Definitions and protocols for verifiable FSS are the focus of Section 4. The appendix discusses applications of FSS and concrete performance for one specific application – two-server PIR. Due to lack of space, the proofs of claims in this work and additional material are postponed to the full version of the paper.

2. PRELIMINARIES

We extend the definition of function secret sharing from [7] by allowing a general specification of the allowable leakage, namely the partial information about the function that can be revealed.

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

We let $\mathsf{Leak}(\hat{f})$ capture partial information about $\hat{f}$ that can be leaked. When Leak is omitted it is understood to output D_f and R_f. (This will be sufficient for most classes considered in this work; for more general classes, one also needs to leak the size S_f.)

As in [7], we consider by default an "additive" representation of the output (i.e., an output y is split into group elements $y_1, \ldots, y_m$ that add up to y), rather than settle for an arbitrary compact output representation. The additive representation is critical for the applications we consider and is achieved by our constructions.

Definition 2.1 (FSS: Syntax). *An m-party* function secret sharing (FSS) scheme *is a pair of algorithms* $(\mathsf{Gen}, \mathsf{Eval})$ *with the following syntax:*

- $\mathsf{Gen}(1^\lambda, \hat{f})$ *is a PPT* key generation *algorithm, which on input 1^λ (security parameter) and $\hat{f} \in \{0,1\}^*$ (description of a function f) outputs an m-tuple of keys $(k_1, \ldots, k_m)$. We assume that $\hat{f}$ explicitly contains an input length 1^n, group description $\mathbb{G}$, and size parameter S (see above).*
- $\mathsf{Eval}(i, k_i, x)$ *is a polynomial-time* evaluation algorithm*, which on input $i \in [m]$ (party index), k_i (key defining $f_i : \{0,1\}^n \to \mathbb{G}$) and $x \in \{0,1\}^n$ (input for f_i) outputs a group element $y_i \in \mathbb{G}$ (the value of $f_i(x)$, the i-th share of $f(x)$).*

When m is omitted, it is understood to be 2. When $m = 2$, we sometimes index the parties by $i \in \{0,1\}$ rather than $i \in \{1,2\}$.

Definition 2.2 (FSS: Security). *Let $\mathcal{F} = (P_\mathcal{F}, E_\mathcal{F})$ be a function family and $\mathsf{Leak} : \{0,1\}^* \to \{0,1\}^*$ be a function specifying the allowable leakage. Let m (number of parties) and t (secrecy threshold) be positive integers. An* m-party t-secure FSS for $\mathcal{F}$ with leakage Leak *is a pair* $(\mathsf{Gen}, \mathsf{Eval})$ *as in Definition 2.1, satisfying the following requirements.*

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

We will also use the natural concrete security variant of (T, ϵ)-secure FSS. When Leak is omitted, it is understood to be the function $\mathsf{Leak}(\hat{f}) = (1^n, \mathbb{G})$ where 1^n and $\mathbb{G}$ are the input length and group description contained in $\hat{f}$. When t is omitted it is understood to be $m - 1$.

Definition 2.3 (Distributed Point Function). *A* point function *$f_{\alpha,\beta}$, for $\alpha \in \{0,1\}^n$ and $\beta \in \mathbb{G}$, is defined to be the function $f : \{0,1\}^n \to \mathbb{G}$ such that $f(\alpha) = \beta$ and $f(x) = 0$ for $x \neq \alpha$. A* Distributed Point Function *(DPF) is an FSS for the family of all point functions, with the default leakage (i.e., $\mathsf{Leak}(\hat{f}) = (1^n, \mathbb{G})$).*

3. NEW FSS CONSTRUCTIONS FROM ONE-WAY FUNCTIONS

In this section, we present a collection of new FSS constructions whose security relies only on one-way functions. At the core of our new results is a new procedure for combining FSS schemes together via a "tensoring" operation, to obtain FSS for a more expressive function class. A direct iterative execution of this operation with two different recursion parameters reproduces both the DPF constructions of Gilboa and Ishai [18] and the (seemingly quite different) tree-based DPF construction from [7].

Further exploring this operation, we make progress in two directions:

Improved efficiency. We demonstrate new optimizations for the case of DPFs, yielding concrete efficiency improvements over the state-of-the-art constructions (for both DPFs and FSS and for interval functions) [7], dropping the key size of an n-bit DPF from $4n(\lambda + 1)$ down to just $n(\lambda + 2)$ bits. We also provide a new procedure for efficiently performing a full domain DPF evaluation (i.e., evaluating on every element of the input domain), a task which occurs frequently within PIR-style applications.

Extended expressiveness. Then, by exploiting the generalization of the procedure, we construct FSS for the class of polynomial-sized decision trees. This enables applications such as multi-dimensional interval queries.

We also demonstrate an orthogonal means of obtaining increased FSS expressibility, achieving FSS for the *product* of two supported function classes, in exchange for requiring a larger number of parties m.

3.1 DPF Tensor Operation

Given the following three tools: (1) a DPF scheme $\mathsf{FSS}^\bullet = (\mathsf{Gen}^\bullet, \mathsf{Eval}^\bullet)$ for the class of multi-bit point functions $\mathcal{F}^\bullet$, (2) an FSS scheme $(\mathsf{Gen}^\mathcal{F}, \mathsf{Eval}^\mathcal{F})$ for an arbitrary class of functions $\mathcal{F}$ whose keys are pseudorandom (and support an additive group structure), and (3) a pseudorandom generator, we construct an FSS scheme for the *tensor* of the function family $\mathcal{F}$ with the class of single-bit point functions: that is, functions $g_{\alpha,f}(x, y)$ which evaluate to $f(y)$ on inputs (α, y), and to 0 elsewhere.

Note that if $\mathcal{F}^\bullet$ supports n_1-bit inputs and $\mathcal{F}$ supports n_2-bit inputs then the resulting function class $\mathcal{F}^\bullet \otimes \mathcal{F}$ takes $(n_1 + n_2)$-bit inputs. The key size of the resulting FSS $(\mathsf{Gen}^\otimes, \mathsf{Eval}^\otimes)$ will correspond to $\mathsf{size}_\otimes(n_1+n_2, \lambda) = \mathsf{size}_\bullet(n_1, \lambda)+ 2\mathsf{size}_\mathcal{F}(n_2, \lambda)$.

REMARK 3.1. *In the case when $\mathcal{F}$ is itself a class of (multi-bit) point functions $\mathcal{F}^\bullet$, the result of this tensor $\mathcal{F}^\bullet_{n_1} \otimes \mathcal{F}^\bullet_{n_2}$ will correspond directly to another class of (multi-bit) point functions $\mathcal{F}^\bullet_{n_1+n_2}$ with larger domain. Repeating this process iteratively by doubling the input bit-length in each step ($n_1 = n_2$) yields a construction isomorphic to that from [18], with key size $O(n^{\log_2 3})$ bits. Alternatively, repeating this process with $n_2 = 1$ at each step yields the construction from [7], with key size $4n(\lambda + 1)$ bits.*

Intuitively, the transformation works as follows. We use the DPF to generate keys for a function which outputs a random seed concatenated with the bit 1 on the special input α, and 0 everywhere else. This means (viewing the scheme with "subtractive" reconstruction, for simplicity) that when evaluating at $x = \alpha$ the parties reach independent random output seeds s_0, s_1, and disagreeing bits $t_0 = 1 - t_1$, whereas everywhere else their outputs will agree. The s_b's can then be used to generate long(er) *masks* (via a PRG) to hide information from the other party. In the tensor construction, the masks are used to hide FSS keys from the second scheme: the parties are both given *both* keys to the second FSS, but with one masked by the PRG-output of s_0 and the other masked by the PRG-output of s_1. These are the "correction words." The bit t_b tells the party which of the correction words to use. When $t_0 = t_1$ and $s_0 = s_1$, the parties will perform identical actions, and their final output will be the same. For the special input α, they will exactly remove the masks and evaluate using the revealed FSS keys. The pseudorandomness of the $\mathcal{F}$ FSS keys means the parties cannot identify which input is the special one.

Note that new keys have the form of one key from the DPF and two elements in the key space of the second FSS: that is, the resulting key size $\mathsf{size}_\otimes(n_1 + n_2, \lambda)$ is indeed $\mathsf{size}_\bullet(n_1, \lambda) + 2\mathsf{size}_\mathcal{F}(n_2, \lambda)$.

We defer a formal treatment of the tensor product operation to the full version.

3.2 Optimized DPF and PIR-like Applications

For input length n, security parameter λ, and 1-bit outputs, the best known DPF constructions [7] achieved key size $4n(\lambda+1)$ bits (the key size grows accordingly for larger outputs). We now demonstrate an optimized DPF construction stemming from the tensor approach, which drops the key size down to $n(\lambda + 2)$ bits.

THEOREM 3.2 (OPTIMIZED DPF). *Assuming a pseudorandom generator $G : \{0,1\}^\lambda \to \{0,1\}^{2(\lambda+1)}$, then the scheme $(\mathsf{Gen}^\bullet, \mathsf{Eval}^\bullet)$ in Figure 1 is a secure DPF for $f_{\alpha,\beta} : \{0,1\}^n \to \mathbb{G}$ with key size $n(\lambda + 2) + \log_2 |\mathbb{G}|$.*

We obtain savings in two different ways. First, we modify the generic tensor transformation (accordingly, the scheme of [7]) so that instead of needing *two* correction words for each level, we can suffice with one. The reason this is possible here is because the "second" FSS scheme in this instance is a single-bit-input DPF, which is simply a secret shared string of the truth table. For such FSS we do not need to enforce full control over the unmasked key values that the parties will compute in order to guarantee correct evaluation, but rather only over the *difference* between the values. This saves us one factor of 2.

Second, we are able to shrink the size of each correction word by roughly a factor of 2 (explicitly, from $2(\lambda + 1)$ bits to $(\lambda + 2)$). Recall that the goal of the correction word is to shift a (pseudo-)random string (a_1, a_2) so that it agrees with a second pseudo-random string (b_1, b_2) on one half $i \in \{0, 1\}$, and remains independent on the other half. Previous constructions achieved this via shifting by a correction word (c_1, c_2), where $c_i = a_i \oplus b_i$, and c_{1-i} was a random offset. We observe that the introduced randomness in the latter shift is unnecessary, and instead shift *both* halves by the same offset. Since a_{1-i} and b_{1-i} were (pseudo-)random and independent to begin with, conditioned on a_i, b_i, this property will be preserved with the shift $a_i \oplus b_i$. This provides us with our second saved factor of 2.

The pseudocode of our DPF construction is given in Figure 1. We provide a formal proof of security within the full version.

3.2.1 Full Domain Evaluation

Some applications of DPF require running the Eval algorithm on every element of the input domain. As an example, consider two-server Private Information Retrieval (PIR) in which two servers $\mathcal{S}_0, \mathcal{S}_1$ hold the same $N = 2^n$ group elements $x_0, \ldots, x_{N-1} \in \mathbb{G}$, for some abelian group $\mathbb{G}$, and a user wishes to retrieve x_α while hiding α.

PIR can be implemented using DPF as described in the introduction. A straightforward implementation of this idea requires each server to run $\mathsf{Eval}^\bullet$ N times independently, once for each input x_j. We show two improvements to this naïve approach of evaluating the whole input domain, leveraging the structure of our particular construction.

Consider a rooted binary tree whose leaves are the elements of the input domain and the path from the root to a leaf reflects the binary representation of the element's index. In other words, the path moves from the current node to the left child if the next bit in the representation is 0 and to the right child if the next bit in the representation is 1. In our construction $\mathsf{Eval}^\bullet(b, k_b, x)$ traverses the path from the root to a leaf x and the naïve algorithm for full domain evaluation traverses each of these paths resulting in time $O(nN)$. However, for every node in the tree there is a unique $\tau^{(i)}$ value computed by any execution of $\mathsf{Eval}^\bullet$ that traverses the node. Since the τ values and the correction words are sufficient to compute the result of $\mathsf{Eval}^\bullet$ on a single point computing full domain evaluation can be carried out by computing the τ values for each node in the tree which requires $O(N)$ time.

A second improvement is the *early termination* optimization for small output groups. The correction word $CW^{(n+1)}$ in $\mathsf{Gen}^\bullet$ is the output β masked by the expansion of two seeds. If the representation of β is short then several output values can be "packed" into $CW^{(n+1)}$. For any node V of depth ν in the tree there are $2^{n-\nu}$ leaves in its sub-tree, or $2^{n-\nu}$ input elements with a shared prefix that ends at V. If the size of $CW^{(\nu)}$ is at least $2^{n-\nu}$ times the output length then the main loop of both $\mathsf{Gen}^\bullet$ and $\mathsf{Eval}^\bullet$ can terminate at level ν instead of at level n. In this case $CW^{(\nu)}$ will be a sequence of group elements masked by the two expanded seeds. The sequence will have the output β in the location specified by the last $n - \nu$ bits of α and the unit elements of

	1-bit output			128-bit output		
Input Domain	Key length in bits	$\mathsf{Eval}^\bullet$ - #AES operations	$\mathsf{Gen}^\bullet$ - #AES operations	Key length in bits	$\mathsf{Eval}^\bullet$ - #AES operations	$\mathsf{Gen}^\bullet$ - #AES operations
$\{0,1\}^n$	$\approx (n - \log \lambda)\lambda$	$n - \log \lambda$	$2(n - \log \lambda)$	$n\lambda + 2(n + \lambda + 1)$	n	$2n$
$\{0,1\}^{16}$	1417	9	18	2320	16	32
$\{0,1\}^{25}$	2578	18	36	3481	25	50
$\{0,1\}^{40}$	4513	33	66	5416	40	80
$\{0,1\}^{80}$	9673	73	146	10576	80	160
$\{0,1\}^{160}$	19993	153	306	20896	160	320

Table 1: Performance of the optimized DPF construction as in Figure 1. In the case of single bit outputs we use the additional *early termination* optimization, which is presented in Section 3.2.1. The precise analytic expression for the key size is somewhat cumbersome, but is reflected in the exact key size for each concrete input domain. We implement the PRG operation expanding $s \in \{0,1\}^{127}$ to 256 bits by computing $\mathrm{AES}_{s||0}(0)||\mathrm{AES}_{s||0}(1)$.

Optimized Distributed Point Function $(\mathsf{Gen}^\bullet, \mathsf{Eval}^\bullet)$
Let $G : \{0,1\}^\lambda \to \{0,1\}^{2(\lambda+1)}$ a pseudorandom generator.
Let $\mathsf{Convert}_\mathbb{G} : \{0,1\}^\lambda \to \mathbb{G}$ be a map converting a random λ-bit string to a pseudorandom group element of $\mathbb{G}$.

$\mathsf{Gen}^\bullet(1^\lambda, \alpha, \beta, \mathbb{G})$:

Let $\alpha = \alpha_1, \ldots, \alpha_n \in \{0,1\}^n$ be the bit decomposition.
Sample random $s_0^{(0)} \leftarrow \{0,1\}^\lambda$ and $s_1^{(0)} \leftarrow \{0,1\}^\lambda$
Sample random $t_0^{(0)} \leftarrow \{0,1\}$ and take $t_1^{(0)} \leftarrow t_0^{(0)} \oplus 1$
for $i = 1$ to n **do**
  $s_0^L||t_0^L \;||\; s_0^R||t_0^R \leftarrow G(s_0^{(i-1)})$ and $s_1^L||t_1^L \;||\; s_1^R||t_1^R \leftarrow G(s_1^{(i-1)})$.
  if $\alpha_i = 0$ **then** $\mathsf{Keep} \leftarrow L$, $\mathsf{Lose} \leftarrow R$
  else $\mathsf{Keep} \leftarrow R$, $\mathsf{Lose} \leftarrow L$
  end if
  $s_{CW} \leftarrow s_0^{\mathsf{Lose}} \oplus s_1^{\mathsf{Lose}}$
  $t_{CW}^L \leftarrow t_0^L \oplus t_1^L \oplus \alpha_i \oplus 1$ and $t_{CW}^R \leftarrow t_0^R \oplus t_1^R \oplus \alpha_i$
  $CW^{(i)} \leftarrow s_{CW}||t_{CW}^L||t_{CW}^R$
  $s_b^{(i)} \leftarrow s_b^{\mathsf{Keep}} \oplus t_b^{(i-1)} \cdot s_{CW}$ for $b = 0, 1$
  $t_b^{(i)} \leftarrow t_b^{\mathsf{Keep}} \oplus t_b^{(i-1)} \cdot t_{CW}^{\mathsf{Keep}}$ for $b = 0, 1$
end for
$CW^{(n+1)} \leftarrow (-1)^{t_1^n}\big[\beta - \mathsf{Convert}(s_0^{(n)}) + \mathsf{Convert}(s_1^{(n)})\big]$, with addition in $\mathbb{G}$
Let $k_b = s_b^{(0)}||t_b^{(0)}||CW^{(1)}||\cdots||CW^{(n+1)}$
return (k_0, k_1)

$\mathsf{Eval}^\bullet(b, k_b, x)$:

Parse $k_b = s^{(0)}||t^{(0)}||CW^{(1)}||\cdots||CW^{(n+1)}$
for $i = 1$ to n **do**
  Parse $CW^{(i)} = s_{CW}||t_{CW}^L||t_{CW}^R$
  $\tau^{(i)} \leftarrow G(s^{(i-1)}) \oplus (t^{(i-1)} \cdot \big[s_{CW}||t_{CW}^L||s_{CW}||t_{CW}^R\big])$
  Parse $\tau^{(i)} = s^L||t^L \;||\; s^R||t^R \in \{0,1\}^{2(\lambda+1)}$
  if $x_i = 0$ **then** $s^{(i)} \leftarrow s^L, t^{(i)} \leftarrow t^L$
  else $s^{(i)} \leftarrow s^R$ and $t^{(i)} \leftarrow t^R$
  end if
end for
return $(-1)^b\big[\mathsf{Convert}(s^{(n)}) + t^{(n)} \cdot CW^{(n+1)}\big] \in \mathbb{G}$

Figure 1: Pseudocode for optimized DPF construction for $f_{\alpha,\beta} : \{0,1\}^n \to \mathbb{G}$, where $||$ denotes string concatenation. Subscripts 0 and 1 refer to party id. All s values are λ-bit strings and t values are a single bit.

$\mathbb{G}$ in every other location. The two improvements described above lead to the following theorem statement.

THEOREM 3.3 (FULL DOMAIN EVALUATION). *Let λ be a security parameter, $\mathbb{G}$ be an abelian group and $\ell = \log|\mathbb{G}|$. Let $z = \max\{\lambda, \ell\}$ and let G be a PRG $G : \{0,1\}^\lambda \to \{0,1\}^{2z+1}$. There exists a full domain evaluation protocol for $f_{\alpha,\beta} : \{0,1\}^n \to \mathbb{G}$ with key size at most $(n - \lfloor \log \frac{z}{\ell} \rfloor)(\lambda + 2)$ and at most $\lceil 2^n \frac{\ell}{z} \rceil$ PRG operations.*

3.3 FSS for Decision Trees

We now describe how the tensoring approach can be utilized to provide FSS for the broader class of *decision trees*. A decision tree is defined by a tree topology, variable labels on each node v (which can take values in some node-specific set S_v), value labels on each edge (in our case, corresponding to some element in S_v), and output labels on each leaf node.

In our construction, the key size is roughly $\lambda \cdot |V|$, where V is the set of nodes, and evaluation on a given input requires $|V|$ executions of a pseudorandom generator, and a comparable number of additions. The FSS is guaranteed to hide the secret edge value labels and leaf output labels (which we refer to as "Decisions"), but (in order to achieve this efficiency) reveals the base tree topology and node variable assignments (which we refer to as "Tree").

As a simple illustrative example, consider a decision tree representation of the OR function on n bits x_i. The tree topology includes a length-n chain of nodes (each labeled by a unique input x_i), with edges all labeled by 0, ending in a terminal output node (labeled by 0). In addition, from each internal node there is a second edge, labeled by 1, terminating in a leaf labeled by 1. In this example, the leaked information "Tree" consists of the structure of the tree and the n node labels x_i; the hidden information "Decisions" consists of the choice of condition labels 0,1 on each edge, as well as the 0,1 leaf output labels. In particular, the resulting FSS key cannot be distinguished from the analogous FSS key for the AND function, which has an identical structure but with the 0 and 1 roles reversed.

Note that FSS for decision trees could be attained directly from a linear combination of separate DPFs: for each leaf node, simply include an additional corresponding DPF. However, such an approach is wasteful; in particular, our DPF constructions have the property that a DPF key contains within it explicitly DPF keys for each of its prefixes. Because of this, we can optimize the required key size and

computation, to leverage the "shared backbone" of paths within the decision tree. Our construction instead directly applies the tensoring approach to "append" each node onto the backbone structure, one by one.

More specifically, our construction is recursive. The key generation procedure Gen starts from the leaves. In each step, we obtain an FSS key for a given tree structure Γ by sampling a key for a smaller tree Γ' with all siblings of one leaf node in Γ removed (say, children u_a of a node v). The output value we select for this newly formed leaf v in the restricted decision tree program will be the same structure $s||1 \in \{0,1\}^{\lambda+1}$ as in the tensor operation from the previous section. And, in an analogous fashion, we will construct a "correction word" associated with this node v, which contains the target values for its children, but masked by the PRG-output of the parties' respective shares of s (i.e., $G(s_b)$). The key thus consists of a correction word for every node, which is an element of $\{0,1\}^{\lambda+1}$ for each internal node, and is an element of the output group $\mathbb{G}$ for each leaf node. Security of the scheme follows the same argument as in the tensor.

The evaluation procedure Eval begins from the root node, and calls a recursive function EvalNode. When executed on a leaf, EvalNode outputs its correction word directly. When executed on an internal node, EvalNode outputs the sum of EvalNode on each of its children.

We refer the reader to the full version for formal definitions, constructions, and proof.

Constant-dimension intervals. A sample application of our FSS construction for decision trees is for constant d-dimensional interval queries: that is, functions $f(x_1, \ldots, x_d)$ which evaluate to a selected nonzero value precisely when $a_i \leq x_i \leq b_i$ for some secret interval ranges $(a_i, b_i)_{i\in[d]}$. (See, e.g., [28] for supporting a similar functionality in the context of searching on encrypted data.) For n-bit inputs of length ℓ we achieve FSS for d-dimensional intervals with key size and computation time $O(n^d)$. For small values of d, such as $d = 2$ for supporting a conjunction of intervals, this yields solutions with reasonably good concrete efficiency.

Corollary 3.4. *For $d \in \mathbb{N}$ there exists FSS for the class of d-dimensional intervals $(a_i, b_i)_{i\in[d]}$ with key size $O(\lambda \cdot n^d)$.*

The construction can be achieved as follows. First, we reduce from general d-dimensional intervals to the problem of 2^d *"special"* intervals, whose left-boundary a_i is equal to 0. This can be done by means of a linear combination of special intervals via inclusion-exclusion (and recalling that FSS schemes combine linearly [7]).

To illustrate the construction of FSS for these special d-dimensional intervals, consider the case of $d = 1$ and 2. Observe that a 1-dimensional special interval for n-bit inputs can be expressed directly as a decision list; that is, a decision tree with one long length-n path $u_1, \ldots, u_n$ with edges (u_i, u_{i+1}), and single terminal edges with appropriate 0/1 output labels departing from each node along the path. (Namely, a generalization of the OR function construction discussed earlier).

To extend to 2 dimensions, the 0/1 terminal edges from nodes u_i are each replaced by a length-n *decision list* (as above), this time labeled by the corresponding bits of the second input y. Departing from the primary path corresponds to either falling outside the x-dimension interval (in which case the final leaf will be labeled 0) or within it, in which case the leaf will be labeled based on the 1-dimensional y interval. A similar approach can be taken to extend to general d dimensions, for constant d.

We remark that revealing the topology and node labels of the utilized d-dimension decision tree (as is the case in our FSS for decision trees construction) does not adversely affect security, since this structure is identical across any choice of secret interval boundaries. Rather, the only thing that differs in the construction is the choice of edge and leaf node labels, which is precisely what is hidden by our FSS construction.

3.4 A Product Operator for FSS

In this section we present a simple technique for increasing the expressive power of FSS by increasing the number of parties. We consider here function families $\mathcal{F}$ such that for each $f \in \mathcal{F}$ the output range R_f is equipped with a ring structure. For two functions f_1, f_2 with the same input domain and output ring, we naturally define the product $f = f_1 \cdot f_2$ by $f(x) = f_1(x)f_2(x)$.

Definition 3.5 (Product of function families). *Let $\mathcal{F}_1, \mathcal{F}_2$ be function families. Define*

$$\mathcal{F}_1 \cdot \mathcal{F}_2 = \{f_1 \cdot f_2 \ : \ f_1 \in \mathcal{F}_1,\ f_2 \in \mathcal{F}_2,\ D_{f_1} = D_{f_2},\ R_{f_1} = R_{f_2}\}$$

The product operator can be used for expressing function classes that capture conjunctions. For instance, if the input x is partitioned into (x_1, x_2) and $\mathcal{F}_1, \mathcal{F}_2$ are the classes of interval functions applied to x_1 and x_2 respectively, the class $\mathcal{F}_1 \cdot \mathcal{F}_2$ is the class of all two-dimensional intervals. Compared with the solution based on decision trees, we will get better efficiency (linear in the bit-length of the input instead of quadratic) at the cost of using a larger number of parties.

The following theorem captures the basic FSS product operator. (Here we use (m,t)-FSS as an abbreviation for an m-party t-secure FSS.) In the full version we describe an alternative product operator that converts $(m, m-1)$-FSS schemes for classes $\mathcal{F}_i$, $1 \leq i \leq m-1$, into an $(m,1)$-FSS $(\mathsf{Gen}, \mathsf{Eval})$ for $\mathcal{F} = \prod_{i=1}^{m-1} \mathcal{F}_i$, where the key size for sharing $f_1 \cdot f_2 \cdots f_d$ is $m-1$ times the total key sizes required for sharing f_i.

Theorem 3.6. *Let $(\mathsf{Gen}_1, \mathsf{Eval}_1)$ be an (m_1, t)-FSS for $\mathcal{F}_1$ and $(\mathsf{Gen}_2, \mathsf{Eval}_2)$ be an (m_2, t)-FSS for $\mathcal{F}_2$. Then there exists an $(m_1 m_2, t)$-FSS $(\mathsf{Gen}, \mathsf{Eval})$ for $\mathcal{F} = \mathcal{F}_1 \cdot \mathcal{F}_2$ in which the size of the key produced by Gen on $f_1 \cdot f_2$ is the sum of the key sizes of Gen_1 on f_1 and Gen_2 on f_2.*

As a concrete instance, using a 3-party PRG-based DPF construction from [7] (which achieves a quadratic improvement over a naive solution), one can get a nontrivial PRG-based $(3,1)$-FSS that supports private searches involving a conjunction of two keywords or ranges.

4. VERIFIABLE FSS: HANDLING MALICIOUS CLIENTS

As discussed in the Introduction, FSS schemes are motivated by two types of applications: ones that involve privately *reading* from a database stored at two or more servers, and ones that involve privately *writing* into a secret-shared array. In both types of applications, badly formed FSS keys can enable a malicious client to gain an unfair advantage.

Consider, for example, an application of DPF for maintaining a secret data histogram, say for the purpose of website traffic analytics. The histogram count of each bin $x \in [N]$ is additively secret-shared between two servers over a large group $\mathbb{G} = \mathbb{Z}_p$. A valid "increment α" query for a (hidden) bin α should correspond to secret shares of the point function $f_{\alpha,1}$ that evaluates to 1 at α and 0 at all other points. However, since each server sees only one share of the function (to hide the identity of α), neither server on its own has a way of verifying that the given shares really do encode such a "proper" point function. A malicious client may just as easily encode a function that increments item α by 100, effectively casting a "heavy" vote. Even worse, it may encode a "garbage" function f^* that increments every bin by some random amount—effectively erasing all the prior existing counts. We seek efficient procedures for the servers to verify the validity of the function shared by the client before it is being processed, by using a small amount of server-to-server communication.

4.1 Modeling Verifiable FSS

We consider an FSS scheme where m the keys are generated by a potentially malicious client, but are processed by *semi-honest* servers, to which we refer from here on as *parties*. That is, the parties honestly follow the prescribed protocol, but try to infer as much information as possible about an honest client's secret function f from their view. We allow the parties to communicate over secure point-to-point channels, but try to minimize the communication to the extent possible. (Some form of communication between the parties is clearly necessary for verifying the consistency of the keys.) Finally, we make the simplifying assumption that parties can generate a common source of randomness which is unpredictable to the clients, and do not count this randomness towards the communication. In practice, this common source of randomness can be realized by having one of the parties pick a random PRF key and communicating it to all other parties. The same PRF key can be used to verify many FSS instances, as long as it is independent of the FSS keys.

We are interested in verification protocols that minimize communication between parties, do not involve the client beyond the initial key distribution, and do not involve any additional parties. This should be contrasted with a verification protocol from [11] that involves an additional party and requires communication linear in the domain size.

Similarly to the verification protocol from [11], we would like our protocols to only make a *black-box* use of the underlying FSS scheme (which in our case translates into a black-box use of a PRG). The latter requirement is meant to rule out protocols that involve a generic use of secure computation for verifying that the keys form a valid output of Gen. To the end of realizing such a black-box verification, we are willing to slightly relax the goal by settling for the validity of the keys *with respect to a given set of inputs*. That is, the verification protocol is given a subset D' of the input domain D as an additional input. The set D' represents the set of inputs on which the function f will actually be evaluated, and hence may not be known to the client. The distinction between D and D' is motivated by applications such as secure keyword search or range queries, where D' is typically a tiny subset of D. However, in other applications of FSS, including PIR and distributed histograms, it is typically the case that $D' = D$.

We allow the running time of the verification protocol, *but not its communication complexity*, to grow with the size of D'. In fact, all verification protocols we present invoke Eval on every $x \in D'$. This is typically not an efficiency bottleneck, since these evaluations are anyway necessary for the application. We expect our solutions to beat the concrete efficiency of applying practical general-purpose MPC protocols to the function defined by Gen except, perhaps, when D' is very large.

The verification protocol should have the following soundness property: If the verification protocol is successful, then the parties are essentially convinced that the function f^* effectively shared by the client is consistent with *some* $f \in \mathcal{F}$ on the domain D'. Of course, the verification protocol should not reveal to the parties any information about the function f^* beyond its validity.

We formalize the above requirements below. For simplicity we do not explicitly treat general leakage, since we will present verification protocols for FSS with standard leakage, namely where only the input domain D_f and output domain R_f are leaked. However, the definitions extend in a straightforward way to the general case.

Definition 4.1 (Verifiable FSS). *Let $\mathcal{F}$ be a class of functions. An (m,t)-*verifiable FSS *(VFSS) for $\mathcal{F}$ is a triplet of algorithms $(\mathsf{Gen}, \mathsf{Eval}, \mathsf{Ver})$ such that $(\mathsf{Gen}, \mathsf{Eval})$ are an FSS for $\mathcal{F}$ as in Definition 2.2, and there exists a negligible function negl for which the following additional requirements hold.*

- **Syntax of Ver:** *Ver is an m-party interactive protocol. In the beginning of the protocol, each party i has a local input k_i (presumably an output of Gen). In addition, all parties share the following common inputs: a security parameter 1^λ, an input domain D for (an unknown) $f \in \mathcal{F}$, output domain R for f, and a subset $D' \subseteq D$ of relevant evaluation points. (An empty D' is interpreted as $D' = D$.) We also assume that the parties have access to a common source of randomness picked independently of the inputs, and do not count this randomness towards the communication complexity. In the end of the protocol, each party outputs "Accept" or "Reject."*
- **Completeness:** *If $(k_1, \ldots, k_m)$ are valid outputs of $\mathsf{Gen}(1^\lambda, f)$ for some $f \in \mathcal{F}$ with input domain D and output domain R, then for all $D' \subseteq D$, in the execution of Ver on local inputs $(k_1, \ldots, k_m)$ and common inputs D, D', R, all parties output "Accept" with probability 1.*
- **Soundness:** *Consider the following security experiment defined by a PPT adversary $\mathcal{A}$ running on input 1^λ:*
  1. *$\mathcal{A}(1^\lambda)$ outputs a description of FSS input domain D, FSS output domain R, FSS keys $(k_1^*, \ldots, k_m^*)$, and a set $D' \subseteq D$ (represented by an explicit list of elements).*
  2. *The protocol Ver is executed on local inputs $(k_1^*, \ldots, k_m^*)$ and common inputs $1^\lambda, D, R, D'$.*
  3. *$\mathcal{A}$ wins if at least one party outputs "Accept" and moreover there is* no *function $f \in \mathcal{F}$ with input domain D^* and output domain R^* for which $f_1^*(x) + \ldots + f_m^*(x) = f(x)$ for all $x \in D'$ (where $f_i^*(x) := \mathsf{Eval}(i, k_i^*, x)$).*

The soundness requirement is that every PPT $\mathcal{A}$ can only win the above game with negligible probability in λ.

- **Secrecy:** *Following the (honest) execution of* Ver *on keys $(k_1, \ldots, k_m)$ generated by* $\mathsf{Gen}(1^\lambda, f)$ *(with an arbitrary $D' \subseteq D_f$), the joint view of any t parties should not reveal anything about f except D_f and R_f. This is formalized as in the secrecy requirement of Definition 2.2, except that the output of* Real *includes the entire view of parties in S. When t is unspecified, it is understood to be $m - 1$.*

SELECTIVE FAILURE ATTACKS. While allowing D' to be a strict subset of D is useful, it may also give rise to security vulnerabilities. First, the above soundness requirement does not rule out a correlation between the set D' of relevant evaluation points and the event of rejecting.[1] While selective failure attacks can often be problematic, we would like to argue that they are not a major concern in the context of natural applications of verifiable FSS. First, as discussed earlier, verifiable FSS is most strongly motivated by "writing" scenarios, where we typically have $D' = D$ and the client learns nothing from the event of rejection. Even if we use $D' \subset D$, in such scenarios the client does not need to be directly informed that an error has been detected, and his vote can be silently discarded. In any case, the price of being caught cheating typically outweighs the advantage of learning one bit of information about D'.

A second type of attack that may apply to the case where $D' \subset D$ is when an invalid function shared by a malicious client coincides with a valid function when restricted to D'. For instance, in a verifiable DPF protocol, a malicious client can share a function that has a nonzero output on many points (say, simultaneously voting for many candidates) in the hope that the function will have only one nonzero output on the (unknown) subset D'. Both types of attacks are irrelevant to the case $D' = D$, and they can be mitigated by incurring a penalty for being caught cheating.

4.2 Template for Verifiable FSS Protocols

We assume that the output domain $\mathbb{G}$ is of the form $\mathbb{Z}_p^\ell$, for a prime p and positive integer ℓ, and view it as the additive group of the (large) finite field $\mathbb{F}_{p^\ell}$. Our verification protocols typically achieve soundness error of $O(1/|\mathbb{F}|)$ by communicating just a constant number of field elements. To verify FSS over a small group, such as $\mathbb{G} = \mathbb{Z}_2$, one can view $\mathbb{G}$ as a subgroup of the additive group of a sufficiently large field, say $\mathbb{F} = \mathbb{F}_{2^\lambda}$, and apply a verification scheme for an FSS over $\mathbb{F}$. (This does not require any changes to Gen or Eval, only to make Ver view each output of Eval as an element of $\mathbb{F}$.)

The high level idea of our verifiable protocols is the following. Let $N = |D'|$. Consider the function family $\mathcal{F}'$ defined by restricting $\mathcal{F}$ to the evaluation points in D'. By locally applying Eval on the inputs in D', the parties obtain an additive secret sharing of a (long) vector $\mathbf{y} \in \mathbb{F}^N$ consisting of the values $(f^*(x))_{x \in D'}$. The parties need to verify that $\mathbf{y}$ is *valid*, namely it is consistent with $\mathcal{F}'$, using only a small amount of communication and without revealing information about $\mathbf{y}$. For instance, in the case of point functions the vector $\mathbf{y}$ should satisfy the requirement that it has at most one nonzero entry.

The verification that $\mathbf{y}$ is well-formed is achieved via the following combination of randomized linear sketching and special-purpose MPC. The parties use their common source of randomness to pick a linear function $L : \mathbb{F}^N \to \mathbb{F}^d$, where d is a small constant. The function L (also referred to as a "linear sketch") is picked from a carefully chosen distribution $\mathcal{L}$ that has the following properties:

1. Given $\mathbf{z} = L(\mathbf{y})$, one can decide (with negligible error probability over the choice of L) whether $\mathbf{y}$ is valid, namely it is consistent with $\mathcal{F}$;
2. This decision procedure is "MPC friendly" in the sense that there is a very efficient MPC protocol Π_{MPC} for verifying that a secret-shared vector $\mathbf{z} \in \mathbb{F}^d$ is of the right form.

Given a distribution $\mathcal{L}$ and an MPC protocol Π_{MPC} as above, the protocol Ver proceeds as follows. Party i, holding a (long) additive share $\mathbf{y}^i$ of $\mathbf{y}$, locally compresses $\mathbf{y}^i$ into $\mathbf{z}^i = L(\mathbf{y}^i)$. Then the parties run Π_{MPC} for deciding whether to accept $\mathbf{y}$ as being consistent with $\mathcal{F}'$.

To further improve the efficiency of Π_{MPC}, we let the FSS client distribute between the parties correlated randomness that is consumed by Π_{MPC}. This correlated randomness can be incorporated into the keys produced by Gen and does not require additional interaction with the client. However, it is critical that the soundness of the verification hold even if this correlated randomness is distributed by a malicious client. Our solutions for this type of "client-assisted MPC" problems can be useful beyond the context of verifiable FSS.

To fully instantiate the above template, we need to specify the distribution $\mathcal{L}$ from which L is picked, the verification predicate $\mathcal{V}$ applied to $\mathbf{z}$, and the MPC protocol Π_{MPC} for (client-assisted) computation of $\mathcal{V}$ on the shares of $\mathbf{z}$. See Figure 2 for a formal description of a verification protocol following this template. In Section 4.3 we will instantiate the sketching distribution $\mathcal{L}$ and in Section 4.4 we will instantiate the MPC protocol Π_{MPC}.

4.3 Instantiating the Sketching Scheme $(\mathcal{L}, \mathcal{V})$

In this section we propose several efficient instantiations of the sketching scheme $(\mathcal{L}, \mathcal{V})$ that apply to useful classes $\mathcal{F}$ and support very efficient MPC protocols for $\mathcal{V}$. All instantiations rely on the standard Schwartz-Zippel (SZ) lemma, bounding the number of zeros of a low-degree polynomial. Due to space constraints we defer proofs to the full version.

DPF $f_{\alpha,\beta}$ WITH $\beta \in \{0, 1\}$ AND $\mathbb{F}$ OF CHARACTERISTIC > 2. This is the most useful case for applications that involve voting or counting, where each client can increment a single counter by 1 or "abstain" by using $\beta = 0$. Here we use $\mathcal{L}_{\mathsf{sq}}$ that picks random field elements $r_1, \ldots, r_N$ and outputs the matrix $L \in \mathbb{F}^{2 \times N}$ defined by $L_{1,j} = r_j$ and $L_{2,j} = r_j^2$. That is, each column of L contains a random field element and the square of this element. (In an actual implementation, L can be generated using a short PRF key picked by one of the parties and sent to all others.) The verification predicate $\mathcal{V}_{\mathsf{sq}}$, which will be realized by Π_{MPC}, checks that the sketch $\mathbf{z} = (z_1, z_2)$ satisfies the condition $\mathcal{V}_{\mathsf{sq}}(z_1, z_2) = z_1^2 - z_2 = 0$.

[1] In fact, such correlations are inherent to any solution that only makes a black-box access of Eval, which includes all of the efficient solutions we present next. Indeed, there is no way to efficiently distinguish between, say, a random point function on λ-bit inputs and a function that has a nonzero value on two random inputs.

FSS Verification Template $\mathsf{Ver}(i, k_i^*)$

COMMON INPUTS:

- Security parameter 1^λ;
- Input length 1^n, of FSS input domain $D = \{0,1\}^n$;
- FSS output group $\mathbb{G} \subseteq \mathbb{F}$, where $\mathbb{F}$ is a finite field and $|\mathbb{F}| \geq 2^\lambda$; // Here λ can be taken to be a *statistical* security parameter.
- Subset $D' \subseteq D$ of size N. An empty D' is interpreted as $D' = D$; // Running time is linear in N.

GIVEN ALGORITHMS:

- FSS evaluation algorithm Eval;
- Matrix sampler $\mathcal{L}(\mathbb{F}, N)$, outputting a matrix $L \in \mathbb{F}^{d \times N}$ (for some constant $d \geq 1$);
- Verification predicate $\mathcal{V} : \mathbb{F}^d \to \mathbb{F}^{d'}$;
- Client-assisted MPC protocol Π_{MPC} for $\mathcal{V}$.

1: **Picking matrix:** Let $L \leftarrow \mathcal{L}(\mathbb{F}, N)$ using common randomness;
// Same L is used by all parties; L can be reused as long as it is independent of all k_i^*.
2: **Applying Eval:** $\mathbf{y}^i \leftarrow (\mathsf{Eval}(i, k_i^*, x))_{x \in D'}$; //$\mathbf{y}^i \in \mathbb{F}^N$
3: **Local compression:** Let $\mathbf{z}^i \leftarrow L \cdot \mathbf{y}^i$; //$\mathbf{z}^i \in \mathbb{F}^d$
4: **Interactive verification:** Run Π_{MPC} on input $\mathbf{z}^i$ using correlated randomness in k_i^* to evaluate $\mathcal{V}(\mathbf{z})$ and accept if and only if the output of Π_{MPC} is the all-0 vector.

Figure 2: Template for FSS verification protocol for a given $(\mathsf{Gen}, \mathsf{Eval})$.

CLAIM 4.2. *Let $\mathbb{F}$ be a finite field of characteristic $p > 2$. If $\mathbf{y} \in \mathbb{F}^N$ is neither a unit vector nor the all-0 vector, then*

$$\Pr\left[L \leftarrow \mathcal{L}_{\mathsf{sq}}(\mathbb{F}, N); (z_1, z_2) \leftarrow L \cdot \mathbf{y} \,:\, z_2 = z_1^2\right] \leq 2/|\mathbb{F}|.$$

DPF $f_{\alpha,\beta}$ WITH $\beta \in \{0,1\}$ AND GENERAL $\mathbb{F}$. We can eliminate the restriction on $\mathbb{F}$ by using a sketch of $d = 3$ field elements: $\mathcal{L}_{\mathsf{prod}}$ picks $L \in \mathbb{F}^{3 \times N}$ as a random matrix whose third row is the product of the first two. That is, $L_{1,j} = r_j$, $L_{2,j} = s_j$, and $L_{3j} = r_j s_j$ where the r_j and s_j are random and independent field elements. The verification predicate is $\mathcal{V}_{\mathsf{prod}}(z_1, z_2, z_3) = z_1 z_2 - z_3$.

CLAIM 4.3. *Let $\mathbb{F}$ be any finite field. Suppose $\mathbf{y} \in \mathbb{F}^N$ is neither a unit vector nor the all-0 vector. Then* $\Pr[L \leftarrow \mathcal{L}_{\mathsf{prod}}(\mathbb{F}, N); (z_1, z_2, z_3) \leftarrow L \cdot \mathbf{y} \,:\, z_3 = z_1 z_2] \leq 2/|\mathbb{F}|$.

DPF $f_{\alpha,\beta}$ WITH $\beta \in \{1,-1\}$ AND GENERAL $\mathbb{F}$. Our next sketching procedure applies to general fields and, like the first procedure, only requires a sketch of $d = 2$ field elements. An additional difference is that the set of possible β values is $\{1,-1\}$ instead of $\{0,1\}$. In the case of fields of characteristic 2, this is equivalent to requiring that $\beta = 1$. Over other fields, one can either view the extra possibility as a feature, e.g., for votes that involve "liking" or "disliking" a candidate, or enforce the requirement that $\beta = 1$ as described below. Here we use $\mathcal{L}_{\mathsf{inv}}$ that picks random *nonzero* field elements $r_1, \ldots, r_N$ and define $L \in \mathbb{F}^{2 \times N}$ by $L_{1,j} = r_j$ and $L_{2,j} = r_j^{-1}$. That is, each column of L contains a random field element and its inverse. The verification predicate is $\mathcal{V}_{\mathsf{inv}}(z_1, z_2) = z_1 z_2 - 1 = 0$. While generating L using $\mathcal{L}_{\mathsf{inv}}$ is computationally more expensive than $\mathcal{L}_{\mathsf{sq}}$, its cost can be amortized since the same L can be used to verify many DPF keys.

CLAIM 4.4. *Let $\mathbb{F}$ be any finite field. If $\mathbf{y} \in \mathbb{F}^N$ is neither a unit vector nor the negation of a unit vector, then*

$$\Pr\left[L \leftarrow \mathcal{L}_{\mathsf{inv}}(\mathbb{F}, N); (z_1, z_2) \leftarrow L \cdot \mathbf{y} \,:\, z_1 z_2 = 1\right] \leq N/(|\mathbb{F}|-1).$$

DPF $f_{\alpha,\beta}$ WITH $\beta = 1$. The above sketching schemes allow β to take two possible values. If we want to ensure that $\beta = 1$, it suffices to additionally check that the sum of all entries in $\mathbf{y}$ is equal to 1. Using our linear sketching framework, this can be done in both cases by adding to L an additional all-1 row and extending the verification predicate $\mathcal{V}$. For instance, for $d = 2$, we extend $\mathcal{V}(z_1, z_2)$ into $\mathcal{V}'(z_1, z_2, z_3) = (\mathcal{V}(z_1, z_2), z_3 - 1)$.

Note that the above two sketching schemes are only useful when $D' = D$. When D' is a strict subset of D, the resulting verifiable FSS would fail to be *complete* in the case $\alpha \notin \mathcal{D}'$, because in this case $\mathbf{y}$ is the all-0 vector which is rejected by the above sketching schemes.

DPF $f_{\alpha,\beta}$ WITH UNRESTRICTED $\beta \in \mathbb{F}$. The above sketching schemes natively support useful restrictions of β, namely either $\beta \in \{0,1\}$, $\beta \in \{1,-1\}$, or $\beta = 1$. However, in some applications, such as "writing" applications in which a client is free to overwrite the entire contents of a single entry of an array, it is useful to support a DPF $f_{\alpha,\beta}$ where β can be an arbitrary field element. To this end, we augment the general template in Figure 2 by allowing the verification predicate $\mathcal{V}$ to be non-deterministic. That is, $\mathcal{V}$ may depend an additional input $w \in \mathbb{F}$ that is secret-shared by the client as part of Gen. Given an honestly generated w, the verification succeeds, and even a maliciously generated w cannot increase the probability of accepting an invalid $\mathbf{y}$.

To verify an arbitrary point function $f_{\alpha,\beta}$, we augment the previous verification predicates as follows:

- $\mathcal{V}'_{\mathsf{sq}}(z_1, z_2, w) = z_1^2 - z_2 w$, where an honest client uses $w = \beta$.
- $\mathcal{V}'_{\mathsf{prod}}(z_1, z_2, z_3, w) = z_1 z_2 - z_3 w$, where an honest client uses $w = \beta$.
- $\mathcal{V}'_{\mathsf{inv}}(z_1, z_2, w) = z_1 z_2 - w$, where an honest client uses $w = \beta^2$.

Completeness is easy to verify. To argue soundness, one can modify the previous case analysis to show that for any $\mathbf{y}$ with at least two nozero entries and for any fixed $w \in \mathbb{F}$, the polynomial $\mathcal{V}'$ still contains a monomial with a nonzero coefficient.

FSS FOR INTERVALS. In the full version, we present an extension of the above techniques to the class of *interval* functions $f_{[a,b]}$ that evaluate to 1 on all $x \in [a, b]$ (where x is interpreted as an integer in $[0, 2^n - 1]$) and evaluates to 0 on all other inputs.

4.4 Instantiating the MPC Protocol Π_{MPC}

The previous sketching schemes reduce the verification that a long vector $\mathbf{y}$ is "well formed" (i.e., belongs to some

set $B \subseteq \mathbb{F}^N$ defined by $\mathcal{F}$ and D'), to computing a simple, low-degree predicate $\mathcal{V}$ on a short vector $\mathbf{z}$ given an additive sharing $\mathbf{z}^1, \ldots, \mathbf{z}^m$ of $\mathbf{z}$.

Given the simple nature of the predicates $\mathcal{V}$ we use, the parties could compute $\mathcal{V}$ on their own. For instance, if there is an honest majority of parties ($t < m/2$) they could use a "BGW-style" protocol [4], or if there is no honest majority they could use a "GMW-style" protocol [19] or an arithmetic variant of this protocol [17, 23]. However, in the latter case, and in particular in the 2-party case, such protocols make use of public-key cryptography and involve a considerable computation and communication overhead. While this overhead can be amortized to some extent over multiple instances (e.g., using OT extension techniques [2, 21]), we can obtain better asymptotic and concrete efficiency by using the help of correlated randomness provided by the client as part of its key generation.

We present two different MPC techniques that apply to different scenarios. The first applies to the basic scenario of verifying predicates $\mathcal{V}$ for the simple sketching schemes described above. It relies on Beaver's circuit randomization technique [1] and its soundness exploits the fact that if $\mathbf{y}$ is invalid, then the output of $\mathcal{V}$ is not only nonzero with high probability but it also has a lot of *entropy* that cannot be eliminated even if the client provides badly formed correlated randomness. The second technique is based on so-called *linear PCPs* and applies to a more specialized verification scenario in which such entropy is not present.

Using shared products. Originating from Beaver's circuit randomization technique [1], a common technique for speeding up MPC protocols is by employing correlated randomness provided by a trusted dealer (or securely implemented using input-independent preprocessing). In the case of semi-honest parties, Beaver's technique is very efficient: it requires the dealer to send 3 field elements to each party for each multiplication gate, and requires each party to communicate to all other parties just a single field element for every input or multiplication gate.

Naturally, when the dealer is malicious, the security guarantees of the protocol break down. (The goal of protecting such protocols against a malicious dealer is orthogonal to the goal of protecting them against malicious parties; see, e.g., [5, 12] for efficient solutions to the latter.) In [16] it was observed that in natural protocols of this type, the effect of a malicious dealer corresponds precisely to an *additive attack* on the circuit computed by the protocol, namely an attack that can "blindly" add a field element to every internal wire in the arithmetic circuit computed by the protocol. To protect against this type of attacks, the solution proposed in [16] is to protect the computation against additive attack by using a special type of fault-tolerant circuit called "AMD circuit." While this approach can be used protect against a malicious dealer with a constant overhead, this constant is quite large and the resulting protocols are fairly complex.

Our main observation is that for the purposes of securely verifying $\mathcal{V}$, the additive attacks induced by badly formed correlated randomness are harmless, because the soundness of the sketching scheme holds even in the presence of such attacks. As noted above, the high level reason for this is that the attack cannot reduce the entropy of $\mathcal{V}(\mathbf{z})$ for a sketch $\mathbf{z}$ computed from a badly formed $\mathbf{y}$.

More concretely, the predicates $\mathcal{V}$ defined above only require either one or two multiplications, where each multiplication of additively shared secrets a and b is implemented using Beaver's technique as follows:

- Inputs: Additive shares $[a] = ([a]_1, \ldots, [a]_m)$ and $[b] = ([b]_1, \ldots, [b]_m)$ of secrets $a, b \in \mathbb{F}$.
- Outputs: Additive shares $[c] = ([c]_1, \ldots, [c]_m)$ of $c = ab$.
- Correlated randomness: Random additive shares $[a'], [b']$ of random and independent secrets $a', b' \in \mathbb{F}$, and random additive shares $[c']$ of $c' = a'b'$.
  // This correlated randomness is included in the keys output by Gen.
- Communication: Party i locally computes $[\Delta a]_i = [a]_i - [a']_i$ and $[\Delta b]_i = [b]_i - [b']_i$ and sends $[\Delta a]_i$ and $[\Delta b]_i$ to all other parties.
- Computing output: Party i computes $\Delta a = \sum_{j=1}^{m} [\Delta a]_j$ and $\Delta b = \sum_{j=1}^{m} [\Delta b]_j$, and outputs $[c]_i = \Delta a \Delta b + \Delta b[a]_i + \Delta a[b]_i + [c']_i$.

To evaluate $\mathcal{V}$ on the shared sketch $[\mathbf{z}]$, we use the above procedure for evaluating each multiplication, where additions are implemented non-interactively, and the output is reconstructed by simply exchanging shares of the output.

We now briefly analyze the security of the FSS verification protocol obtained by combining the above client-aided MPC protocol with the sketching schemes proposed above. First, since we assume the parties to be semi-honest, the secrecy property follows from the semi-honest security of the MPC protocol (i.e., the only information learned by the parites is that $\mathcal{V}(\mathbf{z}) = 0$, which is always the case for an honest client). Consider the case of correlated randomness generated by a malicious client. Since all predicates $\mathcal{V}$ we consider include only a single level of multiplications, the effect of such bad randomness is limited to adding some $\chi \in \mathbb{F}$ to each *output*. Indeed, since every possible choice of $[a']$ and $[b']$ is valid, an inconsistency can always be viewed as an error in the choice of c', which is only *added* to the *output*. (If the computation involves two multiplications followed by an addition, which is needed for some of the predicates $\mathcal{V}$ we propose, the two errors χ_1 and χ_2 are added.)

The crucial point is that the additive error χ introduced by bad randomness is independent of the randomness of $\mathcal{L}$. Hence, it suffices to observe that the soundness of the sketching schemes $(\mathcal{L}, \mathcal{V})$ we propose holds also if the constant χ is added to the output of $\mathcal{V}$. This follows from the fact that soundness is argued via the Schwartz-Zippel Lemma applied to polynomials whose degree is greater than 1, for which adding a constant does not change the degree.

Using linear PCPs for specially structured β. The soundness of the above MPC protocol crucially depends on the fact that the client cannot predict the inputs for the protocol in case $\mathbf{y}$ is invalid. In the full version, we describe a different technique for the case where the inputs are fully known to the client. Concretely, the technique is used to make the client prove that the value β of a point function $f_{\alpha,\beta}$ has a special structure. A particularly useful instance is where β is of the form $(\gamma, \gamma^2, \ldots, \gamma^d) \in \mathbb{F}^d$ for a finite field $\mathbb{F}$. A β vector of this form can be used to decode a set of messages that match the same search criterion in "reading" applications or are written to the same bin in "writing" applications [15, 11]. The idea is to make the client help the parties verify the validity of β by secret-sharing a suitable

proof between the parties. The proof can be viewed as a special case of the so-called Hadamard-PCP that has been previously used in the context of sublinear-communication arguments for NP [22, 6], exploiting the simple algebraic structure of the statement for better efficiency.

4.5 Putting the Pieces Together

We conclude by summarizing the type of FSS verification enabled by combining the above sketching schemes and MPC protocols.

THEOREM 4.5 (VERIFIABLE FSS). *For each function family $\mathcal{F}_i$ specified below, the following holds. For every m-party FSS scheme $\Pi = (\mathsf{Gen}, \mathsf{Eval})$ for $\mathcal{F}_i$, there is an m-party* verifiable *FSS scheme $\Pi' = (\mathsf{Gen}', \mathsf{Eval}, \mathsf{Ver})$ for $\mathcal{F}_i$ such that keys generated by Gen' include keys of Gen and a constant number of additional field elements, and in Ver parties invoke Eval once on each $x \in D'$, perform $O(|D'|)$ additional field operations, and communicate a constant number of field elements with each other.*

The function families $\mathcal{F}_i$ are:

- *All point functions $f_{\alpha,\beta}$;*
- *All point functions $f_{\alpha,\beta}$ with $\beta \in \{0, 1\}$ (alternatively, with $\beta = 1$ or $\beta \in \{1, -1\}$ if $D' = D$);*
- *All interval functions $f_{[a,b]}$ and their negations $-f_{[a,b]}$.*
- *All point functions $f_{\alpha,\beta}$ for β of the special form $(\gamma, \gamma^2, \ldots, \gamma^d) \in \mathbb{F}^d$; here the keys produced by Gen include $O(d)$ additional field elements.*

ACKNOWLEDGEMENTS. Research done in part while visiting the Simons Institute for the Theory of Computing, supported by the Simons Foundation and by the DIMACS / Simons Collaboration in Cryptography through NSF grant #CNS-1523467. Supported by ERC starting grant 259426.

First author additionally supported by ISF grant 1709/14 and ERC starting grant 307952. Second author additionally supported by ISF grant 1638/15, a grant by the BGU Cyber Center, the Israeli Ministry Of Science and Technology Cyber Program and by the European Union's Horizon 2020 ICT program (Mikelangelo project). Third author additionally supported by ISF grant 1709/14, BSF grant 2012378, a DARPA/ARL SAFEWARE award, NSF Frontier Award 1413955, NSF grants 1228984, 1136174, 1118096, and 1065276. This material is based upon work supported by the Defense Advanced Research Projects Agency through the ARL under Contract W911NF-15-C-0205. The views expressed are those of the author and do not reflect the official policy or position of the Department of Defense, the National Science Foundation, or the U.S. Government.

5. REFERENCES

[1] D. Beaver. Efficient multiparty protocols using circuit randomization. In *CRYPTO '91*, pages 420–432, 1991.

[2] D. Beaver. Correlated pseudorandomness and the complexity of private computations. In *STOC 1996*, pages 479–488, 1996.

[3] A. Beimel, Y. Ishai, E. Kushilevitz, and I. Orlov. Share conversion and private information retrieval. In *CCC 2012*, pages 258–268, 2012.

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

[5] R. Bendlin, I. Damgård, C. Orlandi, and S. Zakarias. Semi-homomorphic encryption and multiparty computation. In *Eurocrypt 2011*, pages 169–188, 2011.

[6] N. Bitansky, A. Chiesa, Y. Ishai, R. Ostrovsky, and O. Paneth. Succinct non-interactive arguments via linear interactive proofs. In *TCC*, pages 315–333, 2013.

[7] E. Boyle, N. Gilboa, and Y. Ishai. Function secret sharing. In *EUROCRYPT 2015*, pages 337–367, 2015.

[8] B. Chor and N. Gilboa. Computationally private information retrieval (extended abstract). In *STOC 1997*, pages 304–313, 1997.

[9] B. Chor, N. Gilboa, and M. Naor. Private information retrieval by keywords. *IACR Cryptology ePrint Archive*, 1998:3, 1998.

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

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

[12] I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. In *CRYPTO 2012*, pages 643–662, 2012.

[13] G. Di Crescenzo, T. Malkin, and R. Ostrovsky. Single database private information retrieval implies oblivious transfer. In *EUROCRYPT 2000*, pages 122–138, 2000.

[14] Y. Dodis, S. Halevi, R. D. Rothblum, and D. Wichs. Spooky encryption and its applications. In *CRYPTO 2016*, pages 93–122, 2016.

[15] M. Finiasz and K. Ramchandran. Private stream search at the same communication cost as a regular search: Role of LDPC codes. In *ISIT 2012*, pages 2556–2560, 2012.

[16] D. Genkin, Y. Ishai, M. Prabhakaran, A. Sahai, and E. Tromer. Circuits resilient to additive attacks with applications to secure computation. In *STOC 2014*, pages 495–504, 2014.

[17] N. Gilboa. Two party RSA key generation. In *CRYPTO '99*, pages 116–129, 1999.

[18] N. Gilboa and Y. Ishai. Distributed point functions and their applications. In *Advances in Cryptology - EUROCRYPT*, pages 640–658, 2014.

[19] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *STOC*, pages 218–229, 1987.

[20] T. Gupta, N. Crooks, W. Mulhern, S. Setty, L. Alvisi, and M. Walfish. Scalable and private media consumption with popcorn. In *NSDI 2016*, 2016.

[21] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank. Extending oblivious transfers efficiently. In *CRYPTO 2003*, pages 145–161, 2003.

[22] Y. Ishai, E. Kushilevitz, and R. Ostrovsky. Efficient arguments without short PCPs. In *22nd CCC*, pages 278–291, 2007.

[23] Y. Ishai, M. Prabhakaran, and A. Sahai. Secure arithmetic computation with no honest majority. In *TCC 2009*, pages 294–314. Springer, 2009.

[24] E. Kushilevitz and R. Ostrovsky. Replication is NOT needed: SINGLE database, computationally-private information retrieval. In *38th FOCS*, pages 364–373, 1997.

[25] C. A. Melchor, J. Barrier, L. Fousse, and M. Killijian. XPIR : Private information retrieval for everyone. *PoPETs*, 2016(2):155–174, 2016.

[26] R. Ostrovsky and V. Shoup. Private information storage (extended abstract). In *STOC 1997*, pages 294–303, 1997.

[27] R. Ostrovsky and W. Skeith III. Private searching on streaming data. In *CRYPTO 2005*, pages 223–240, 2005.

[28] E. Shi, J. Bethencourt, H. T. Chan, D. X. Song, and A. Perrig. Multi-dimensional range query over encrypted data. In *2007 IEEE Symposium on Security and Privacy*, pages 350–364, 2007.

APPENDIX

A. APPLICATIONS OF FSS

In this section we describe some representative applications of FSS beyond those described in the Introduction. For simplicity we restrict attention to 2-party FSS.

PRIVATE KEYWORD SEARCH. Suppose that each of two servers holds a database of keywords with corresponding payloads $X = \{(x_1, p_1), \ldots, (x_N, p_N)\}$ where $x_j \in \{0,1\}^n$ and $p_j \in \{0,1\}^m$. We further assume that no payload p_j is the all-0 string. The client would like to privately test whether some secret keyword x is in the database, and obtain the corresponding payload if it is. To this end, the client uses a DPF to split the point function $f_{x,1} : \{0,1\}^n \to \mathbb{Z}_2$ into f_1, f_2, sending each key to the corresponding server. Server i sends back the sum $\sum_{j=1}^{N} p_j f_i(x_j)$, where each p_j is viewed as an element in $\mathbb{Z}_2^m$. Let $a_1, a_2 \in \mathbb{Z}_2^m$ be the answers received from the two servers. The client lets $p = a_1 + a_2$, and outputs "no match" if $p = 0$ or p otherwise.

GENERALIZED KEYWORD SEARCH. In the case of a generalized keyword search, where the search predicate $f : \{0,1\}^n \to \{0,1\}$ is taken from a class $\mathcal{F}$, the client can count the number of entries j satisfying $f(x_j) = 1$ by viewing the range of f as the group $\mathbb{G} = \mathbb{Z}_{N+1}$. Splitting f into $f_1 + f_2$, server i returns $\sum_{j=1}^{N} f_i(x_j)$, and the client can recover the exact number of matches by adding the two answers in $\mathbb{G}$. A bounded number of matching payloads can be retrieved by using sketching or coding techniques [27, 15].

PRIVATE UPDATES. Consider the following application scenario for a writing analogue of PIR [26]. A user owns N files $(x_1, \ldots, x_N)$ where $x_j \in \{0,1\}^m$. For backup purposes, the files are secret-shared between two cloud servers, namely every x_j is split into $x_{j,1}$ and $x_{j,2}$ such that $x_{j,1} \oplus x_{j,2} = x_j$. The user would like to update file x_α to a new version x'_α without revealing any information about the update (including the identity α of the file that has been updated) to any individual server. To this end, the client lets $\beta = x_j \oplus x'_j$ and uses a DPF to split the point function $f_{\alpha,\beta} : [N] \to \mathbb{Z}_2^m$ into f_1, f_2, sending each key to the corresponding server. Each server i updates its shares of the N files by letting $x'_{j,i} \leftarrow x_{j,i} \oplus f_i(j)$, for $1 \le j \le N$.

B. CONCRETE EFFICIENCY OF PIR

Since the introduction of Private Information Retrieval (PIR), there have been three different models for PIR. The first, presented in the original PIR work of Chor et al. [10], assumes that the database is replicated in $k \ge 2$ non-colluding servers and requires that the protocol be information theoretically secure. The second, proposed by Chor and Gilboa [8], assumes two non-colluding servers but relaxes the security requirement to hold against computationally bounded adversaries. The third model, introduced in [24], assumes that the database is held by a single server and security is again computational.

In the application of DPF to PIR, as described in the Introduction, the query size is the DPF key, which is roughly of size $\lambda \cdot (n - \log \lambda)$ for a database of size $N = 2^n$ records (improving over the previous best DPF-based solution from [7] by roughly a factor of 4). Using Theorem 3.3, the DPF key requires approximately $N/64$ AES operations. In addition to this cost, each server should take the inner product of a (pseudo-)random N-bit vector with the database, effectively amounting to computing the XOR of roughly half the records in the database. The latter cost is common to all multi-server PIR protocols that have 1-bit answers.[2]

Chor et al. [10] present a simple 2-server information-theoretic PIR protocol with N-bit queries and 1-bit answers.This protocol provides good concrete efficiency in applications that involve a small number of records (e.g., see [20]). The optimized DPF protocol in this paper leads to better communication complexity for databases that have 400 items or more. Information-theoretic PIR protocols with $k \ge 3$ servers achieve better communication complexity than two-server protocols. However, employing an additional server may be costly, and moreover, even the best published schemes for practical parameters (e.g., the 3-server protocol in [3] with communication complexity $14N^{1/4}$) have higher communication than our construction for large enough databases, e.g. four billion items for [3]. In terms of computation, all multi-server PIR protocols require each server to expand a received key to a string of N elements and then compute the inner product of the database with the expanded key. In the two-server solution of [10] the expanded string is explicitly sent to each server; other information-theoretic PIR protocols require several field or ring operations per database bit to perform this expansion. We note that all of the alternative approaches do not natively generalize to more general types of searches, such as keyword search or range queries, without a significant overhead introduced by the use of data structures.

Single-server PIR with sublinear communication is known to imply public-key encryption [13]. Indeed all known single-server PIR protocols require (additively homomorphic) *public-key* encryption of the whole database to evaluate a PIR query (cf. [25] and references therein). This is typically several orders of magnitude slower than the solution we present.

[2]To simplify the exposition we restrict our attention to protocols that have 1-bit answers, since these seem to be the most attractive from a concrete efficiency point of view.