Function Secret Sharing: Improvements and Extensions*

Elette Boyle† Niv Gilboa‡ Yuval Ishai§

July 24, 2018

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 obtaining more expressive 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.

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

*This is a full version of [10].

†IDC Herzliya, Israel, eboyle@alum.mit.edu.

‡Ben Gurion University, Israel gilboan@bgu.ac.il.

§Technion, Israel, and UCLA, USA. yuvali@cs.technion.ac.il.

1 Introduction

In this work we continue the study of *Function Secret Sharing (FSS),* a primitive that was recently introduced by Boyle et al. [8] 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 [24]. 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 [24, 8]. This was extended in [8] 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, namely the key size grows linearly with $\sqrt{N}$ where $N = 2^n$ [8]. We consider here the case $m = 2$ by default.

On the high end, efficient FSS schemes for arbitrary polynomial time functions can be based on the Learning with Errors (LWE) assumption by using threshold or key-homomorphic variants of fully homomorphic encryption [8, 18]. (Alternatively, they can be based on indistinguishability obfuscation and one-way functions [8].) In the mid-range, FSS schemes for log-space or NC^1 functions (with inverse polynomial error) are implied by the Decisional Diffie-Hellman assumption [9]. 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 more details and additional examples.

For a typical "reading" application, consider the problem of 2-server Private Information Retrieval (PIR) [14, 12]. In the basic flavor of 2-server PIR, the two servers hold a database of N strings $(x_1, \ldots, x_N)$, where $x_i \in \{0,1\}^\ell$, 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. See Appendix B for a survey of alternative approaches to PIR.

Still relying only on a standard DPF, the above application to PIR can be directly generalized to private search by keywords (returning the payload associated with a private n-bit keyword, see Appendix A), private search on streaming data [35, 21, 8], 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 [13]. In general, FSS for a function family $\mathcal{F}$ can be directly used to perform private searches defined by predicates from $\mathcal{F}$. By additionally using data structures and coding techniques, data structures and coding techniques, the power of FSS for simple function families $\mathcal{F}$ can be significantly boosted. For instance, the recent "Splinter" system [38] efficiently supports a rich class of private search queries by building only on FSS schemes for point functions and interval functions.

For a typical "writing" application, consider the following example from [8]. 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 $\mathbb{G} = \mathbb{Z}_M$ into $f = f_1 + f_2$. 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 [15]. 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 [8] 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 from [8] 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 on 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 reading and 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 the client to send less than $10K$ bits to each server, and each server to send back a string of the same length as the payload. The server computation in this case is dominated by 73 AES invocations per keyword. 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 verification protocols make black-box use of the underlying FSS scheme, and avoid the cost of general-purpose secure computation techniques. The 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.

A verification protocol for DPF was previously proposed in the context of the Riposte system for anonymous messaging [15]. Compared to our verification protocols, the protocol from [15] requires an additional party, its communication complexity is higher, and it only applies to a special (and relatively inefficient) DPF implementation.

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 appendices contain further discussion on applications of FSS and the concrete efficiency of PIR, as well as some proofs.

2 Preliminaries

We extend the definition of function secret sharing from [8] by allowing a general specification of the allowable leakage, namely the partial information about the function that can be revealed. We also give a more explicit treatment of how functions and groups are represented.

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

Representing groups and their elements. We restrict the attention to Abelian groups $\mathbb{G}$ of the form $\mathbb{Z}_{u_1} \times \cdots \times \mathbb{Z}_{u_\ell}$, for positive integers $u_1, \ldots, u_\ell$, and represent such a group by the sequence $(u_1, \ldots, u_\ell)$. A group element $y \in \mathbb{G}$ is naturally described by a sequence of ℓ non-negative integers.

Modeling leakage. We capture the allowable leakage by a function $\mathsf{Leak} : \{0,1\}^* \to \{0,1\}^*$, where $\mathsf{Leak}(\hat{f})$ is interpreted as the partial information about $\hat{f}$ that can be leaked. When Leak is omitted it is understood to output the input domain D_f and the output domain R_f. This will be sufficient for most classes considered in this work; for more general classes, one also needs to leak the size $S_{\hat{f}}$.

Output representation. As in [8], 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). However,

in the 2-party case it will be sometimes convenient to use a *subtractive* FSS, where an output y is represented by a pair of group elements (y_1, y_2) such that $y_1 - y_2 = y$. In the multi-party case it is also useful to consider a further generalization to linear representations captured by arbitrary linear secret sharing schemes, such as Shamir's scheme, but we do not pursue this generalization here.

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 $\mathsf{Sim}(1^\lambda, \mathsf{Leak}(\hat{f}_\lambda))$.

When Leak is omitted, it is understood to be the function $\mathsf{Leak}(\hat{f}) = (1^n, S_{\hat{f}}, \mathbb{G})$ where 1^n, $S_{\hat{f}}$, and $\mathbb{G}$ are the input length, size, and group description contained in $\hat{f}$. When t is omitted it is understood to be $m-1$. Finally, for $m = 2$ we define a *subtractive FSS* in the same way as above, except that in the correctness requirement the predicate $\mathsf{Eval}(1, k_1, x) + \mathsf{Eval}(2, k_2, x) = f(x)$ is replaced by $\mathsf{Eval}(1, k_1, x) - \mathsf{Eval}(2, k_2, x) = f(x)$.

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

To illustrate our representation conventions, a point function $f_{\alpha,\beta}$ for $\alpha \in \{0,1\}^{50}$ and $\beta \in \mathbb{G} = \mathbb{Z}_{100}^3$ is described by (a binary encoding of) $\hat{f} = (50, (100, 100, 100), \alpha, \beta)$.

Indistinguishability vs. simulation. The security requirement in the FSS definition from [8] (see Appendix D) appears to be weaker than the above because it refers to indistinguishability instead of efficient simulation. However, the two flavors are equivalent for every function family $\mathcal{F}$ and leakage function Leak for which Leak can be efficiently inverted; that is, given $\mathsf{Leak}(\hat{f})$

one can efficiently find $\hat{f}'$ such that $\mathsf{Leak}(\hat{f}') = \mathsf{Leak}(\hat{f})$. Such an inversion algorithm exists for all instances of $\mathcal{F}$ and Leak considered in this work.

A CONCRETE SECURITY VARIANT. For the purpose of describing and analyzing some of our FSS constructions, it will be convenient to consider a finite family $\mathcal{F}$ of functions $f : D_f \to R_f$ sharing the same (fixed) input domain and output domain, as well as a fixed value of the security parameter λ. We say that such a finite FSS scheme is (T, ϵ)-*secure* if the computational indistinguishability requirement in Definition 2.2 is replaced by (T, ϵ)-indistinguishability, namely any size-T circuit has at most an ϵ advantage in distinguishing between Real and Ideal. When considering an infinite collection of such finite $\mathcal{F}$, parameterized by the input length n and security parameter λ, we require that Eval and Sim be each implemented by a (uniform) PPT algorithm, which is given 1^n and 1^λ as inputs.

Remark 2.4 (Function Secret Sharing vs. Homomorphic Secret Sharing)**.** FSS can be thought of as a dual of the notion of Homomorphic Secret Sharing (HSS) defined in [9], where the roles of the function and the input are reversed. Whereas FSS considers the goal of secret-sharing a function f (represented by a program) in a way that enables compact evaluation on any given input x via local computation on the shares of f, HSS considers the goal of secret-sharing an input x in a way that enables compact evaluation of any given function f via local computation on the shares of x. While any FSS scheme can be viewed as an HSS scheme for a suitable class of programs and vice versa, the notion of FSS is more liberal in that it allows the share size to grow with the size of the program computing f. Our results for decision trees make essential use of this relaxation. Furthermore, the FSS view is more natural for the function classes and applications considered in this work. We refer the reader to [9, 18] for constructions of HSS and FSS schemes for broader classes of programs under stronger assumptions. In particular, multi-party HSS and FSS schemes for circuits (resp., 2-party schemes for branching programs) can be based on the LWE (resp., DDH) assumption.

3 New FSS Constructions From One-Way Functions

In this section, we present a collection of new FSS constructions that can be based on any pseudorandom generator (PRG), or equivalently a one-way function. 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 [24] and the (seemingly quite different) tree-based DPF construction of Boyle et al. [8].

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 from [8] (for both DPF and FSS for interval functions), 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 a generalization of the tensoring operation, we construct an efficient FSS scheme for decision trees (leaking the tree topology and node labels). 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 $\mathcal{F}^\bullet$ (supporting output length at least $(\lambda + 1)$ bits), (2) an FSS scheme $(\mathsf{Gen}^{\mathcal{F}}, \mathsf{Eval}^{\mathcal{F}})$ for an arbitrary class of functions $\mathcal{F}$ whose keys are pseudorandom bit-strings, 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, the class of functions

$$\mathcal{F}^\bullet \otimes \mathcal{F} := \{g_{\alpha,f} : f_{\alpha,1} \in \mathcal{F}^\bullet, f \in \mathcal{F}\}, \text{ where}$$

$$g_{\alpha,f}(x,y) := f_{\alpha,1}(x) \cdot f(y) = \begin{cases} f(y) & \text{if } x = \alpha \\ 0 & \text{otherwise} \end{cases}.$$

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. As we will later see, 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 input 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 [24], with key size $O(n^{\log_2 3})$ bits. Alternatively, repeating this process with $n_2 = 1$ at each step yields the construction from [8], with key size $4n(\lambda + 1)$ bits.

Intuitively, the transformation works as follows. We use the DPF to generate keys for a function which on the special input α outputs $s||1$, a random seed concatenated with the bit 1, 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 (whose sum is s), 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 thus 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 $\mathsf{size}_\bullet(n_1, \lambda) + 2\mathsf{size}_{\mathcal{F}}(n_2, \lambda)$.

Theorem 3.2. *There exists a polynomial $p(n)$ and constant $c > 1$ such that, if the following tools exist:*

1. **Distributed Point Function:** *$(T, \epsilon_{\mathsf{DPF}})$-secure FSS scheme $(\mathsf{Gen}^\bullet, \mathsf{Eval}^\bullet)$ for multi-bit point function family $\mathcal{F}^\bullet = \{f_{\alpha,\beta}\}_{\alpha,\beta} : \{0,1\}^{n_1} \to \{0,1\}^{\lambda+1}$, with key size $\mathsf{size}_\bullet(n_1, \lambda)$*
2. **FSS with Pseudorandom Keys:**[1] *FSS scheme $(\mathsf{Gen}^{\mathcal{F}}, \mathsf{Eval}^{\mathcal{F}})$ for arbitrary function family $\mathcal{F} : \{0,1\}^{n_2} \to \mathbb{G}$, with key size $\mathsf{size}_{\mathcal{F}}(n_2, \lambda)$, satisfying:*
   - *Key group structure: The key space $\mathcal{K}$ is endowed with an additive group structure.*
   - *$(T, \epsilon_{\mathsf{FSS}})$-key-pseudorandomness: $\forall f \in \mathcal{F}, b \in \{0,1\}$, for every adversary $\mathcal{A}$ running in time no greater than T, the advantage of $\mathcal{A}$ in distinguishing between the distributions $\{k_b : (k_0, k_1) \leftarrow \mathsf{Gen}(1^\lambda, f)\}$ and $\{u : u \leftarrow \mathcal{K}\}$ is bounded by ϵ_{FSS}.*

[1] Note that key-pseudorandomness implies standard FSS security.

3. **Pseudorandom generator:** $(T, \epsilon_{\mathsf{PRG}})$*-secure* $PRG : \{0,1\}^\lambda \to \mathcal{K}$.

then there exists a (T', ϵ')*-secure* FSS $(\mathsf{Gen}^\otimes, \mathsf{Eval}^\otimes)$ *with key size* $\mathsf{size}_\otimes(n_1+n_2, \lambda) = \mathsf{size}_\bullet(n_1, \lambda) + 2\mathsf{size}_\mathcal{F}(n_2, \lambda)$ *for the family of functions* $\mathcal{G} = \mathcal{F}^\bullet \otimes \mathcal{F} := \{g_{\alpha,f} : \{0,1\}^{n_1} \times \{0,1\}^{n_2} \to \mathbb{G} \mid \alpha \in \{0,1\}^{n_1}, f \in \mathcal{F}\}$, *specified by*

$$g_{\alpha,f}(x_1, x_2) := f^\bullet_{\alpha,1}(x_1) \cdot f(x_2) = \begin{cases} f(x_2) & \text{if } x_1 = \alpha \\ 0 & \text{else} \end{cases},$$

for $T' = T - p(n_1 + n_2)$ *and* $\epsilon' = \epsilon_{\mathsf{DPF}} + 2 \cdot \epsilon_{\mathsf{FSS}} + 2 \cdot \epsilon_{\mathsf{PRG}}$.

We provide a full proof of Theorem 3.2 within the Appendix.

3.2 Optimized DPF

For input length n, security parameter λ, and 1-bit outputs, the best previous DPF construction [8] achieved key size $4n(\lambda + 1)$ bits. We now present an optimized DPF construction stemming from the tensor approach, which drops the key size down to $n(\lambda + 2)$ bits.

We obtain savings in two different ways. First, we modify the generic tensor transformation (accordingly, the scheme of [8]) 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.

3.2.1 An informal description

The above overview describes our optimized DPF as a refinement of the construction obtained via the generic tensor transformation. We now give an alternative and self-contained description of the construction, which provides intuition for the more formal description that will follow. For simplicity, consider first the case of a DPF with a single-bit output $\beta = 1$.

At a high level, each of the two keys defines a GGM-style binary tree [25] with 2^n leaves, where the leaves are labeled by inputs $x \in \{0,1\}^n$. We will refer to a path from the root to a leaf labeled by x as the *evaluation path* of x, and to the evaluation path of the special input α as the *special evaluation path*. Each node v in a tree will be labeled by a string of length $\lambda + 1$, consisting of a *control bit* t and a λ-bit *seed* s, where the label of each node is fully determined by the label of its parent. The function $\mathsf{Eval}^\bullet$ will compute the labels of all nodes on the evaluation path to the input x, using the root label as the key, and output the control bit of the leaf.

We would like to maintain the invariant that for each node outside the special path, the two labels (on the two trees) are identical, and for each node on the special path the two control bits are different and the two seeds are indistinguishable from being random and independent. Note that since the label of a node is determined by that of its parent, if this invariant is met for a node outside the special path then it is automatically maintained by its children. Also, we

can easily meet the invariant for the root (which is always on the special path) by just explicitly including the labels in the keys. The challenge is to ensure that the invariant is maintained also when leaving the special path.

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

1. In the 2-party case, additive secret sharing satisfies the following weak homomorphism: If G is a PRG, then $G([s]) = (G(s_0), G(s_1))$ extends shares of the 0-string $s = 0$ into shares of a longer 0-string $S = 0$, and shares of a random seed s into shares of a longer (pseudo-)random string S, where S is pseudo-random even given one share of s.
2. Additive secret sharing is additively homomorphic: given shares $[s], [t]$ of a string s and a bit t, and a public correction word CW, one can locally compute shares of $[s \oplus t \cdot CW]$. We view this as a *conditional correction* of the secret s by CW conditioned on $t = 1$.

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

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

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

3.2.2 The construction

We proceed with a formal description of the optimized DPF construction, whose pseudocode is given in Figure 1.[2]

Theorem 3.3 (Optimized DPF)**.** *Suppose $G : \{0,1\}^\lambda \to \{0,1\}^{2(\lambda+1)}$ is a pseudorandom generator. Then the scheme $(\mathsf{Gen}^\bullet, \mathsf{Eval}^\bullet)$ from Figure 1 is a DPF for the family of point functions $f_{\alpha,\beta} : \{0,1\}^n \to \mathbb{G}$ with key size $n \cdot (\lambda + 2) + \lambda + \lceil \log_2 |\mathbb{G}| \rceil$ bits. The number of PRG invocations in Gen is at most $2(n + \lceil \frac{\log |\mathbb{G}|}{\lambda+2} \rceil)$ and the number of PRG invocations in Eval is at most $n + \lceil \frac{\log |\mathbb{G}|}{\lambda+2} \rceil$.*

[2] A minor difference with respect to the conference version [10] is that the values of $t_0^{(0)}, t_1^{(0)}$ are fixed deterministically (Step 3), whereas in [10] they are selected at random; this does not affect security and results in reducing the key size by one bit.

Optimized Distributed Point Function $(\mathsf{Gen}^\bullet, \mathsf{Eval}^\bullet)$

Let $G : \{0,1\}^\lambda \to \{0,1\}^{2(\lambda+1)}$ be 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 $\mathbb{G}$. (See Figure 3.)

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

Let $\alpha = \alpha_1, \ldots, \alpha_n \in \{0,1\}^n$ be the bit decomposition of α
Sample random $s_0^{(0)} \leftarrow \{0,1\}^\lambda$ and $s_1^{(0)} \leftarrow \{0,1\}^\lambda$
Let $t_0^{(0)} = 0$ and $t_1^{(0)} = 1$
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
$CW^{(n+1)} \leftarrow (-1)^{t_1^n} \cdot \left[\beta - \mathsf{Convert}(s_0^{(n)}) + \mathsf{Convert}(s_1^{(n)})\right] \in \mathbb{G}$
Let $k_b = s_b^{(0)} || CW^{(1)} || \cdots || CW^{(n+1)}$
return (k_0, k_1)

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

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

Figure 1: Pseudocode for optimized DPF construction for the class $f_{\alpha,\beta} : \{0,1\}^n \to \mathbb{G}$. The symbol $||$ 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.

Remark 3.4 (Early termination optimization)**.** For the case of small output group $\mathbb{G}$ (e.g., $\mathbb{G} = \mathbb{Z}_2$), we can further improve the complexity of $(\mathsf{Gen}^\bullet, \mathsf{Eval}^\bullet)$ via an "early termination" optimization. For $\nu := \log_2(\lambda/\log_2|\mathbb{G}|)$, this optimization reduces the key size by $\nu(\lambda+2)$ bits and the number of calls to the psuedorandom generator by ν. See Section 3.2.3 for details.

Proof of Theorem 3.3. Correctness is argued similar to the tensor product case.

Security: We prove that each party's key k_b is pseudorandom. This will be done via a sequence of hybrids, where in each step we replace another correction word $CW^{(i)}$ within the key from being honestly generated to being random.

The high-level argument for security will go as follows. Each party $b \in \{0,1\}$ begins with a random seed $s_b^{(0)}$ that is completely unknown to the other party. In each level of key generation (for $i = 1$ to n), the parties apply a PRG to their seed $s_b^{(i-1)}$ to generate 4 items: namely, 2 seeds s_b^L, s_b^R and 2 bits t_b^L, t_b^R. This process will *always* be performed on a seed which appears completely random and unknown given the view of the other party; because of this, the security of the PRG guarantees that the 4 resulting values appear similarly random and unknown given the view of the other party. The ith level correction word $CW^{(i)}$ will "use up" the secret randomness of 3 of these 4 pieces: the two bits t_b^L, t_b^R, and the seed s_b^{Lose} for $\mathsf{Lose} \in \{L, R\}$ corresponding to the direction *exiting* the "special path" α: i.e. $\mathsf{Lose} = L$ if $\alpha = 1$ and $\mathsf{Lose} = R$ if $\alpha = 0$. However, given this $CW^{(i)}$, the remaining seed s_b^{Keep} for $\mathsf{Keep} \neq \mathsf{Lose}$ still appears random to the other party. The argument then continued in similar fashion to the next level, beginning with seeds s_b^{Keep}.

For each $j \in \{0, 1, \ldots, n+1\}$, we will consider a distribution Hyb_j defined roughly as follows:

1. $s_b^{(0)} \leftarrow \{0,1\}^\lambda$ chosen at random (honestly), and $t_b^{(0)} = b$.
2. $CW^{(1)}, \ldots, CW^{(j)} \leftarrow \{0,1\}^{\lambda+1}$ chosen at random.
3. For $i \leq j$, $s_b^{(i)}||t_b^{(i)}$ computed honestly, as a function of $s_b^{(0)}||t_b^{(0)}$ and $CW^{(1)}, \ldots, CW^{(j)}$.
4. For j, the other party's seed $s_{1-b}^{(j)} \leftarrow \{0,1\}^\lambda$ is chosen at random, and $t_{1-b}^{(j)} = 1 - t_b^{(j)}$.
5. For $i > j$: the remaining values $s_b^{(i)}||t_b^{(i)}, s_{1-b}^{(i)}||t_{1-b}^{(i)}, CW^{(i)}$ all computed honestly, as a function of the previously chosen values.
6. The output of the experiment is $k_b := s_b^{(0)}||CW^{(1)}||\cdots||CW^{(n+1)}$.

Formally, Hyb_j is fully described in Figure 2. Note that when $j = 0$, this experiment corresponds to the honest key distribution, whereas when $j = n+1$ this yields a completely random key k_b. We claim that each pair of adjacent hybrids $j-1$ and j will be indistinguishable based on the security of the pseudorandom generator.

The proof of Theorem 3.3 follows from the following four claims:

Claim 3.5. *For every $b \in \{0,1\}, \alpha \in \{0,1\}^n, \beta \in \mathbb{G}$, it holds that*

$$\{k_b \leftarrow \mathsf{Hyb}_0(1^\lambda, b, \alpha, \beta)\} \equiv \{k_b : (k_0, k_1) \leftarrow \mathsf{Gen}^\bullet(1^\lambda, f_{\alpha,\beta})\}.$$

Claim 3.6. *For every $b \in \{0,1\}, \alpha \in \{0,1\}^n, \beta \in \mathbb{G}$, it holds that*

$$\{k_b \leftarrow \mathsf{Hyb}_{n+1}(1^\lambda, b, \alpha, \beta)\} \equiv \{k_b \leftarrow U\}.$$

Note that Claims 3.5 and 3.6 follow directly by construction of Hyb_j.

Claim 3.7. *There exists a polynomial p' such that for any $(T, \epsilon_{\mathsf{PRG}})$-secure pseudorandom generator G, then for every $j \in [n]$, every $b \in \{0,1\}, \alpha \in \{0,1\}^n, \beta \in \mathbb{G}$, and every nonuniform adversary $\mathcal{A}$ running in time $T' \leq T - p'(\lambda)$, it holds that*

$$\Big|\Pr[k_b \leftarrow \mathsf{Hyb}_{j-1}(1^\lambda, b, \alpha, \beta); c \leftarrow \mathcal{A}(1^\lambda, k_b) : c = 1] - \Pr[k_b \leftarrow \mathsf{Hyb}_j(1^\lambda, b, \alpha, \beta); c \leftarrow \mathcal{A}(1^\lambda, k_b) : c = 1]\Big| < \epsilon_{\mathsf{PRG}}.$$

$\mathsf{Hyb}_j(1^\lambda, b, \alpha, \beta)$:

Let $\alpha = \alpha_1, \ldots, \alpha_n \in \{0,1\}^n$ be the bit decomposition of α
Sample $s_0^{(0)}, s_1^{(0)} \leftarrow \{0,1\}^\lambda$, and let $t_0^{(0)} = 0$ and $t_1^{(0)} = 1$.

for $i = 1$ to n **do**
    Compute $s_b^L || t_b^L || s_b^R || t_b^R = G(s_b^{(i-1)})$.
    if $\alpha_i = 0$ **then** Set $\mathsf{Keep} \leftarrow L$, $\mathsf{Lose} \leftarrow R$. **else**, Set $\mathsf{Keep} \leftarrow R$, $\mathsf{Lose} \leftarrow L$

    if $i < j$ **then**
        Sample $CW^{(i)} \leftarrow \{0,1\}^{\lambda+2}$.
    else
        if $i = j$ **then** Sample $s_{1-b}^{(j-1)} \leftarrow \{0,1\}^\lambda$ and let $t_{1-b}^{(j-1)} = 1 - t_b^{(j-1)}$. **end if**
        Expand $s_{1-b}^L || t_{1-b}^L || s_{1-b}^R || t_{1-b}^R = G(s_{1-b}^{(i-1)})$.
        $CW^{(i)} = \mathsf{ComputeCW}(i, \alpha_i, [s_0^L || t_0^L || s_0^R || t_0^R], [s_1^L || t_1^L || s_1^R || t_1^R])$.
        $(s_{1-b}^{(i)}, t_{1-b}^{(i)}) = \mathsf{ComputeNextST}(1-b, i, t_{1-b}^{(i-1)}, s_{1-b}^{\mathsf{Keep}} || t_{1-b}^{\mathsf{Keep}}, CW^{(i)})$
    end if

    $(s_b^{(i)}, t_b^{(i)}) = \mathsf{ComputeNextST}(b, i, t_b^{(i-1)}, s_b^{\mathsf{Keep}} || t_b^{\mathsf{Keep}}, CW^{(i)})$
end for

if $j = n + 1$ **then**
    $CW^{(n+1)} \leftarrow \mathbb{G}$
else
    $CW^{(n+1)} \leftarrow (-1)^{t_1^n} \cdot \left[\beta - \mathsf{Convert}(s_0^{(n)}) + \mathsf{Convert}(s_1^{(n)})\right] \in \mathbb{G}$
end if
Let $k_b = s_b^{(0)} || CW^{(1)} || \cdots || CW^{(n+1)}$
return k_b

$\mathsf{ComputeCW}(i, \alpha_i, [s_0^L || t_0^L || s_0^R || t_0^R], [s_1^L || t_1^L || s_1^R || t_1^R])$:

if $\alpha_i = 0$ **then** $\mathsf{Keep} \leftarrow L$, $\mathsf{Lose} \leftarrow R$. **else**, $\mathsf{Keep} \leftarrow R$, $\mathsf{Lose} \leftarrow L$
$s_{CW} \leftarrow s_0^{\mathsf{Lose}} \oplus s_1^{\mathsf{Lose}}$
$t_{CW}^L \leftarrow t_0^L \oplus t_1^L \oplus \alpha_i \oplus 1$ and $t_{CW}^R \leftarrow t_0^R \oplus t_1^R \oplus \alpha_i$
return $CW^{(i)} \leftarrow s_{CW} || t_{CW}^L || t_{CW}^R$

$\mathsf{ComputeNextST}(x, i, t_x^{(i-1)}, s_x^{\mathsf{Keep}} || t_x^{\mathsf{Keep}}, CW^{(i)})$:

Parse $CW^{(i)} = s_{CW} || t_{CW}^L || t_{CW}^R$.
$s_x^{(i)} \leftarrow s_x^{\mathsf{Keep}} \oplus t_x^{(i-1)} \cdot s_{CW}$
$t_x^{(i)} \leftarrow t_x^{\mathsf{Keep}} \oplus t_x^{(i-1)} \cdot t_{CW}^{\mathsf{Keep}}$
return $(s_x^{(i)}, t_x^{(i)})$.

Figure 2: Hybrid distribution j, in which the first j correction words are sampled completely at random, and the remaining correction words are computed honestly.

Proof. Fix an arbitrary $j \in [n], b \in \{0,1\}, \alpha \in \{0,1\}^n, \beta \in \mathbb{G}$. Given a Hyb-distinguishing adversary $\mathcal{A}$ with advantage ϵ for these values, we construct a corresponding PRG adversary $\mathcal{B}$. Recall that in the PRG challenge for G, the adversary $\mathcal{B}$ is given a value r that is either computed by sampling a seed $s \leftarrow \{0,1\}^\lambda$ and computing $r = G(s)$, or is sampled truly at random $r \leftarrow \{0,1\}^{2\lambda+2}$.

PRG adversary $\mathcal{B}(1^\lambda, (j, b, \alpha, \beta), r)$:

Let $\alpha = \alpha_1, \ldots, \alpha_n \in \{0,1\}^n$ be the bit decomposition of α
Sample $s_b^{(0)} \leftarrow \{0,1\}^\lambda$, and let $t_b^{(0)} = b$.

for $i = 1$ to $(j-1)$ **do**
    $CW^{(i)} \leftarrow \{0,1\}^{\lambda+2}$.
    Expand $s_b^L||t_b^L\big||s_b^R||t_b^R = G(s_b^{(i-1)})$.
    if $\alpha_i = 0$ **then** Set $\mathsf{Keep} \leftarrow L$, $\mathsf{Lose} \leftarrow R$. **else**, Set $\mathsf{Keep} \leftarrow R$, $\mathsf{Lose} \leftarrow L$
    $(s_b^{(i)}, t_b^{(i)}) = \mathsf{ComputeNextST}(b, i, t_b^{(i-1)}, s_b^{\mathsf{Keep}}, t_b^{\mathsf{Keep}}, CW^{(i)})$.
    Take $t_{1-b}^{(i)} = 1 - t_b^{(i)}$.
end for

Expand $s_b^L||t_b^L\big||s_b^R||t_b^R = G(s_b^{(i-1)})$ and set $s_{1-b}^L||t_{1-b}^L\big||s_{1-b}^R||t_{1-b}^R = r$ (the PRG challenge).
$CW^{(j)} = \mathsf{ComputeCW}(j, \alpha_j, [s_0^L||t_0^L\big||s_0^R||t_0^R], [s_1^L||t_1^L\big||s_1^R||t_1^R])$.
if $\alpha_j = 0$ **then** Set $\mathsf{Keep} \leftarrow L$, $\mathsf{Lose} \leftarrow R$. **else**, Set $\mathsf{Keep} \leftarrow R$, $\mathsf{Lose} \leftarrow L$
Compute $(s_x^{(j)}, t_x^{(j)}) = \mathsf{ComputeNextST}(x, j, t_x^{(j-1)}, s_x^{\mathsf{Keep}}||t_x^{\mathsf{Keep}}, CW^{(j)})$, for both $x \in \{0,1\}$.

Compute $(CW^{(j+1)}|| \cdots ||CW^{(n+1)}) =$
    $\mathsf{RemainingKey}(\alpha, j, CW^{(1)}|| \cdots ||CW^{(j)}, t_0^{(j)}, t_1^{(j)}, [s_0^L||t_0^L\big||s_0^R||t_0^R], [s_1^L||t_1^L\big||s_1^R||t_1^R])$.
return $k_b = s_b^{(0)}||CW^{(1)}|| \cdots ||CW^{(n+1)}$.

$\mathsf{RemainingKey}(\alpha, j, CW^{(1)}|| \cdots ||CW^{(j)}, t_0^{(j)}, t_1^{(j)}, [s_0^L||t_0^L\big||s_0^R||t_0^R], [s_1^L||t_1^L\big||s_1^R||t_1^R])$:

for $i = (j+1)$ to n **do**
    Expand $s_x^L||t_x^L\big||s_x^R||t_x^R = G(s_x^{(i-1)})$ for both $x \in \{0,1\}$.
    if $\alpha_i = 0$ **then** Set $\mathsf{Keep} \leftarrow L$, $\mathsf{Lose} \leftarrow R$. **else**, Set $\mathsf{Keep} \leftarrow R$, $\mathsf{Lose} \leftarrow L$
    $CW^{(i)} = \mathsf{ComputeCW}(i, \alpha_i, [s_0^L||t_0^L\big||s_0^R||t_0^R], [s_1^L||t_1^L\big||s_1^R||t_1^R])$.
    Compute $(s_x^{(i)}, t_x^{(i)}) = \mathsf{ComputeNextST}(x, i, t_x^{(i-1)}, s_x^{\mathsf{Keep}}||t_x^{\mathsf{Keep}}, CW^{(i)})$, for both $x \in \{0,1\}$.
end for

$CW^{(n+1)} \leftarrow (-1)^{t_1^n} \cdot \left[\beta - \mathsf{Convert}(s_0^{(n)}) + \mathsf{Convert}(s_1^{(n)})\right] \in \mathbb{G}$
return $(CW^{(j)}||CW^{(j+1)}|| \cdots ||CW^{(n+1)})$

Now, consider $\mathcal{B}$'s success in the PRG challenge as a function of $\mathcal{A}$'s success in distinguishing Hyb_{j-1} from Hyb_j. If r is computed *pseudorandomly*, then it is clear the generated k_b is distributed as $\mathsf{Hyb}_{j-1}(1^\lambda, b, \alpha, \beta)$.

It remains to show that if r was sampled at random then the generated k_b is distributed as $\mathsf{Hyb}_j(1^\lambda, b, \alpha, \beta)$. That is, if r is random, then the corresponding computed values of $s_{1-b}^{(j)}$ and $CW^{(j)}$ are distributed *randomly* conditioned on the values of $s_b^{(0)}||t_b^{(0)}||CW^{(1)}|| \cdots ||CW^{(j-1)}$, and the value of $t_{1-b}^{(j)}$ is given by $1 - t_b^{(j)}$. Note that all remaining values (for "level" $i > j$) are computed as a function of the values up to "level" j.

First, consider $CW^{(j)}$, computed in three parts:

- $s_{CW} = s_b^{\mathsf{Lose}} \oplus s_{1-b}^{\mathsf{Lose}}$.
- $t_{CW}^L = t_b^L \oplus t_{1-b}^L \oplus \alpha_j \oplus 1$.

- $t^L_{CW} = t^L_b \oplus t^L_{1-b} \oplus \alpha_j$.

In the case that r is random, then $s^{\mathsf{Lose}}_{1-b}, t^L_{1-b}$, and t^R_{1-b} (no matter the value of $\mathsf{Lose} \in \{L, R\}$) are each perfect one-time pads, and so $CW^{(j)} = s_{CW}||t^L_{CW}||t^R_{CW}$ is indeed distributed uniformly.

Now, condition on $CW^{(j)}$ as well, and consider the value of $s^{(j)}_{1-b}$. Depending on the value of $t^{(j-1)}_{1-b}$, $s^{(j)}_{1-b}$ is selected either as s^{Keep}_{1-b} or $s^{\mathsf{Keep}}_{1-b} \oplus s_{CW}$. However, s^{Keep}_{1-b} is distributed uniformly conditioned on the view thus far, and so in either case the resulting value is again distributed uniformly.

Finally, consider the value of $t^{(j)}_{1-b}$. Note that both $t^{(j)}_b$ and $t^{(j)}_{1-b}$ are computed as per $\mathsf{ComputeNextST}$, as a function of $t^{(j-1)}_1$ and $t^{(j-1)}_{1-b}$, respectively (and $t^{(j-1)}_{1-b}$ was set to $1-t^{(j-1)}_b$). In particular,

$$\begin{aligned} t^{(j)}_b \oplus t^{(j)}_{1-b} &= (t^{\mathsf{Keep}}_b \oplus t^{(i-1)}_b \cdot t^{\mathsf{Keep}}_{CW}) \oplus (t^{\mathsf{Keep}}_{1-b} \oplus t^{(i-1)}_{1-b} \cdot t^{\mathsf{Keep}}_{CW}) \\ &= t^{\mathsf{Keep}}_b \oplus t^{\mathsf{Keep}}_{1-b} \oplus (t^{(i-1)}_b \oplus t^{(i-1)}_{1-b}) \cdot t^{\mathsf{Keep}}_{CW} \\ &= t^{\mathsf{Keep}}_b \oplus t^{\mathsf{Keep}}_{1-b} \oplus 1 \cdot (t^{\mathsf{Keep}}_0 \oplus t^{\mathsf{Keep}}_1 \oplus 1) \\ &= 1 \end{aligned}$$

Combining these pieces, we have that in the case of a random PRG challenge r, the resulting distribution of k_b as generated by $\mathcal{B}$ is precisely distributed as is $\mathsf{Hyb}_j(1^\lambda, b, \alpha, \beta)$. Thus, the advantage of $\mathcal{B}$ in the PRG challenge experiment is equivalent to the advantage ϵ of $\mathcal{A}$ in distinguishing $\mathsf{Hyb}_{j-1}(1^\lambda, b, \alpha, \beta)$ from $\mathsf{Hyb}_j(1^\lambda, b, \alpha, \beta)$. The runtime of $\mathcal{B}$ is equal to the runtime of $\mathcal{A}$ plus a fixed polynomial $p'(\lambda)$. Thus for any $T' \leq T - p'(\lambda)$, it must be that the distinguishing advantage ϵ of $\mathcal{A}$ is bounded by ϵ_{PRG}. □

Claim 3.8. *There exists a polynomial p' such that for any $(T, \epsilon_{\mathsf{Convert}})$-secure pseudorandom $\mathsf{Convert} : \{0,1\}^\lambda \to \mathbb{G}$, then for every $b \in \{0,1\}, \alpha \in \{0,1\}^n, \beta \in \mathbb{G}$, and every nonuniform adversary $\mathcal{A}$ running in time $T' \leq T - p'(\lambda)$, it holds that*

$$\Big| \Pr[k_b \leftarrow \mathsf{Hyb}_n(1^\lambda, b, \alpha, \beta); c \leftarrow \mathcal{A}(1^\lambda, k_b) : c = 1] - \Pr[k_b \leftarrow \mathsf{Hyb}_{n+1}(1^\lambda, b, \alpha, \beta); c \leftarrow \mathcal{A}(1^\lambda, k_b) : c = 1] \Big| < \epsilon_{\mathsf{Convert}}.$$

Proof. Fix an arbitrary $b \in \{0,1\}, \alpha \in \{0,1\}^n, \beta \in \mathbb{G}$. In a similar fashion to the previous claim, an adversary $\mathcal{A}$ who distinguishes between the corresponding distributions Hyb_n and Hyb_{n+1} with advantage ϵ directly yields a corresponding adversary $\mathcal{B}$ for the pseudo-randomness of $\mathsf{Convert}$ with the same advantage, and only polynomial additional runtime $p'(\lambda)$. Namely, $\mathcal{B}$ samples $s^{(n)}_b \leftarrow \{0,1\}^\lambda$ and all values $CW^{(1)}, \ldots, CW^{(n)} \leftarrow \{0,1\}^{\lambda+2}$ at random, and then embeds the $\mathsf{Convert}$ challenge by setting $CW^{(n+1)} = (-1)^{t^n_1} \cdot [\beta + (-1)^{1-b} \cdot \mathsf{Convert}(s^{(n)}_b) + (-1)^b \cdot r]$. In the case that r is generated pseudo-randomly as the output of $\mathsf{Convert}(s^{(n)}_{1-b})$ for random $s^{(n)}_{1-b}$, this is precisely the distribution generated by Hyb_n. In the case that r is truly random, then it directly acts as a one-time pad on the remaining terms and thus $CW^{(n+1)}$ is distributed uniformly, precisely as per Hyb_{n+1}. The claim follows. □

This concludes the proof of Theorem 3.3. □

Remark 3.9 (Public vs. local information)**.** Note that in the keys generated by $\mathsf{Gen}^\bullet$ in Figure 1, most of the information consists of correction words which are common to both keys and do not need to be kept secret, and only the initial seeds $s^{(0)}_0$ and $s^{(0)}_1$ need to be kept secret. This may be helpful for further reducing the cost of communicating or storing the DPF keys.

$\mathsf{Convert}_{\mathbb{G}}(s)$:

Let $u \leftarrow |\mathbb{G}|$.
if $u = 2^m$ for an integer m **then**
    if $m \leq \lambda$ **then**
        Return the group element represented by the first m bits of s.
    else
        Return the group element represented by the first m bits of $G(s)$ for a PRG
          $G : \{0,1\}^\lambda \to \{0,1\}^{m+\lambda}$.
    end if
else
    Return the group element corresponding to $G(s)$ mod u.
end if

Figure 3: Pseudocode for converting a string $s \in \{0,1\}^\lambda$ to an element in a group $\mathbb{G}$. If s is random then $\mathsf{Convert}_{\mathbb{G}}(s)$ is pseudo-random.

Input Domain	Output Domain	Key length in bits	$\mathsf{Eval}^\bullet$ PRG operations	$\mathsf{Eval}^\bullet$ AES operations	$\mathsf{Gen}^\bullet$ PRG operations	$\mathsf{Gen}^\bullet$ AES operations
$\{0,1\}^n$	$\mathbb{G}$	$\nu(\lambda+2)+2\lambda$	ν	ν	2ν	4ν
$\{0,1\}^{16}$	$\{0,1\}$	1544	10	10	20	40
	$\{0,1\}^{127}$	2318	16	16	32	64
$\{0,1\}^{25}$	$\{0,1\}$	2705	19	19	38	76
	$\{0,1\}^{127}$	3479	25	25	50	100
$\{0,1\}^{40}$	$\{0,1\}$	4640	34	34	68	136
	$\{0,1\}^{127}$	5414	40	40	80	160
$\{0,1\}^{80}$	$\{0,1\}$	9800	74	74	148	296
	$\{0,1\}^{127}$	10574	80	80	160	320
$\{0,1\}^{160}$	$\{0,1\}$	20120	154	154	308	612
	$\{0,1\}^{127}$	20894	160	160	320	640

Table 1: Performance of the optimized DPF construction from Figure 1. We use the additional *early termination* optimization, which ends the path that $\mathsf{Gen}^\bullet$ and $\mathsf{Eval}^\bullet$ follow at level $\nu = \min\{\lceil n - \log \frac{\lambda}{\log|\mathbb{G}|}\rceil, n\}$ of the tree. With this optimization, the key length is $\nu(\lambda+2)+2\lambda$. The PRG G expanding $s \in \{0,1\}^{127}$ to 256 bits can be implemented either by AES in counter mode, i.e. $G(s) = \mathrm{AES}_{s||0}(0)||\mathrm{AES}_{s||0}(1)$ or by the fixed-key implementation proposal of [38] using the compression function of [32], i.e. $G(s) = (\mathrm{AES}_{k_0}(s||0) \oplus s||0)||(\mathrm{AES}_{k_1}(s||0) \oplus s||0)$ for two fixed keys k_0, k_1. In either case, $\mathsf{Gen}^\bullet$ requires two AES operations per PRG expansion since it uses the whole expanded string, but $\mathsf{Eval}^\bullet$ requires only one AES operation per PRG invocation since the evaluation uses either the left or the right half of the expanded string, depending on the next bit of x.

3.2.3 Full Domain Evaluation

Some applications of DPF require running the evaluation algorithm $\mathsf{Eval}^\bullet$ on every element of the input domain. This is the case, for instance, for the application to 2-server PIR described in the Introduction, or "writing" applications such as the private updates application described in Appendix A (see also [34]). For input domain of size $N = 2^n$, a straightforward implementation uses N independent invocations of $\mathsf{Eval}^\bullet$, once for each input. In this section we present a more efficient scheme that uses the tree structure of our DPF to reduce the total cost by roughly a factor of n.

Notation 3.10. Let $\mathbb{G}$ be a group, $\beta \in \mathbb{G}$ and let j, i be two integers such that $0 \leq j < i$. For any sequence $a \in \mathbb{G}^i$ we denote by $a[j]$ the j-th element in the sequence. We denote by $e^i_{j,\beta} \in \mathbb{G}^i$ the sequence of i group elements such that $e^i_{j,\beta}[j] = \beta$ and $e^i_{j,\beta}[j']$ is the unit element for any $j' \neq j$. If $i = 1$ then $e^i_{j,\beta}$ is simply β. For any two sequences $\gamma_0, \gamma_1 \in \mathbb{G}^i$ let $\gamma_0 + \gamma_1 \in \mathbb{G}^i$ denote the component-wise addition of elements over the group.

Definition 3.11. In the same setting as Definition 2.2 we say that a protocol $(\mathsf{Gen}, \mathsf{EvalAll})$ is a full domain evaluation protocol for $\mathcal{F}$ if the secrecy property is identical to Definition 2.2 and

- **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 $\forall i\ \mathsf{EvalAll}(i, k_i) \in \mathbb{G}^{2^n}$ and

$$\Pr\left[\sum_{i=1}^{m} \mathsf{EvalAll}(i, k_i)[x] = f(x)\right] = 1.$$

Similarly to Definition 2.2 for $m = 2$ we define a *subtractive* full domain evaluation protocol in the same way as above, except that m ranges over 0 and 1 and $\mathsf{EvalAll}(0, k_0)[x] - \mathsf{EvalAll}(1, k_1)[x] = f(x)$.

We present a protocol $(\mathsf{Gen}^{\bullet,*}, \mathsf{EvalAll}^\bullet)$ for full domain evaluation of a two-party DPF, improving on the computational complexity of the naïve solution by a factor of $O(n)$. We leverage the structure of our particular DPF scheme to optimize the construction in two ways.

Consider a rooted binary tree whose leaves are the inputs $x \in \{0,1\}^n$ and the path from the root to a leaf x reflects the binary representation x. More concretely, if $x_i = 0$ (resp., $x_i = 1$), then the i-th step in the path moves from the current node to its left (resp., right) child. In our DPF construction, a single invocation of $\mathsf{Eval}^\bullet(b, k_b, x)$ traverses the path from the root to a leaf x, and so the naïve algorithm for full domain evaluation traverses each of these paths, requiring a total of $O(nN)$ invocations of the PRG. The first improvement is based on the observation that for every node i 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 output of $\mathsf{Eval}^\bullet$ on every single input, full domain evaluation can be carried out by computing the τ values for each node in the tree, requiring only $O(N)$ PRG invocations.

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 $CW^{(\nu+1)}$ 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 $CW^{(\nu+1)}$ 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 element of $\mathbb{G}$ in every other location.

In order to achieve the early termination optimization, the $\mathsf{Convert}$ algorithm in Figure 3 needs to be slightly modified to return a sequence of group elements instead of a single element. We refer to the modified algorithm as $\mathsf{Convert}^*_{\mathbb{G}}(s)$. The only difference between $\mathsf{Convert}^*$ and

$\mathsf{Convert}$ is that in Line 4 $\mathsf{Convert}^*$ returns $\lfloor \lambda/m \rfloor$ group elements defined by the first $\lfloor \lambda/m \rfloor$ blocks of m bits in s.

The pseudo-code in Figure 4 describes $\mathsf{Gen}^{\bullet,*}$ and $\mathsf{EvalAll}^\bullet$. $\mathsf{Gen}^{\bullet,*}$ makes the necessary adjustments to $\mathsf{Gen}^\bullet$ enabling early termination. $\mathsf{EvalAll}^\bullet$ performs full domain evaluation and outputs a sequence of 2^n group elements. The construction implies Theorem 3.12.

Full Domain Evaluation $(\mathsf{Gen}^{\bullet,*}, \mathsf{EvalAll}^\bullet)$

Let $G : \{0,1\}^\lambda \to \{0,1\}^{2(\lambda+1)}$ be a pseudorandom generator.

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

1: Let $\alpha = \alpha_1, \ldots, \alpha_n \in \{0,1\}^n$ be the bit decomposition of α.
2: Let $\nu = \min\{\lceil n - \log \frac{\lambda}{\log|\mathbb{G}|} \rceil, n\}$.
3: Sample random $s_0^{(0)} \leftarrow \{0,1\}^\lambda$ and $s_1^{(0)} \leftarrow \{0,1\}^\lambda$
4: Let $t_0^{(0)} = 0$ and $t_1^{(0)} = 1$
5: Run the main loop of $\mathsf{Gen}^\bullet(1^\lambda, \alpha, \beta, \mathbb{G})$ for $i = 1, \ldots, \nu$ and obtain $CW^{(1)}, \ldots, CW^{(\nu)}$ and $s_0^{(\nu)}, s_1^{(\nu)}, t_1^{(\nu)}$.
6: Let $\hat{\alpha} = \alpha_{\nu+1}, \ldots, \alpha_n$. // $\hat{\alpha}$ indicates the location of α in the sub-tree with root at level $n-\nu$.
7: Let $CW^{(\nu+1,*)} \leftarrow (-1)^{t_1^{(\nu)}} \left[e^{2^{n-\nu}}_{\hat{\alpha},\beta} - \mathsf{Convert}^*_{\mathbb{G}}(s_0^{(\nu)}) + \mathsf{Convert}_{\mathbb{G}}(s_1^{(\nu)})\right]$.
8: Let $k_b = (s_b^{(0)} || n || CW^{(1)} || \cdots || CW^{(\nu)} || CW^{(\nu+1,*)})$.
9: **return** (k_0, k_1).

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

1: Parse $k_b = (s^{(0)} || n || CW)$ for $CW = CW^{(1)} || \cdots || CW^{(\nu)} || CW^{(\nu+1,*)}$.
2: Let $t^{(0)} = b$.
3: Return $\mathsf{Traverse}(s^{(0)}, t^{(0)}, n, CW, \nu+1, 1, b)$.

$\mathsf{Traverse}(s, t, n, CW, i, j)$:

1: **if** $i > 0$ **then**
2: $\quad i \leftarrow i - 1$
3: $\quad$ Parse $CW^{(i)} = s_{CW} || t^L_{CW} || t^R_{CW}$.
4: $\quad$ Let $\tau^{(i)} \leftarrow G(s) \oplus t \cdot \left[s_{CW} || t^L_{CW} || s_{CW} || t^R_{CW}\right]$.
5: $\quad$ Parse $\tau^{(i)} = s^L || t^L \;||\; s^R || t^R \in \{0,1\}^{2(\lambda+1)}$.
6: $\quad$ Return $\mathsf{Traverse}(s^L, t^L, n, CW, i-1, j) || \mathsf{Traverse}(s^R, t^R, n, CW, i-1, j + 2^{n-\nu+i-1})$.
7: **else** // $i = 0$
8: $\quad$ Let $P_{j'} = \mathsf{Convert}^*_{\mathbb{G}}(s + t \cdot CW^{(\nu+1,*)})[j']$ for $j' = j, \ldots, j + 2^{n-\nu} - 1$.
9: $\quad$ Return $P_j || \ldots || P_{j+2^{n-\nu}-1}$.
10: **end if**

Figure 4: Pseudocode for optimized DPF evaluation on the entire input domain.

Theorem 3.12 (Full domain evaluation)**.** *Let λ be a security parameter, let $G : \{0,1\}^\lambda \to \{0,1\}^{2(\lambda+1)}$ be a PRG and let $\mathbb{G}$ be a group. The scheme $(\mathsf{Gen}^{\bullet,*}, \mathsf{EvalAll}^\bullet)$ in Figure 4 is a full domain evaluation protocol for the family of point functions from $\{0,1\}^n$ to $\mathbb{G}$. The keys that $\mathsf{Gen}^{\bullet,*}$ outputs are of size at most $\nu(\lambda+2) + \lambda + \log n + \max\{\lambda, \log|\mathbb{G}|\}$ bits, the number of PRG invocations in $\mathsf{Gen}^{\bullet,*}$ is at most $2\nu + 2\lceil \frac{\log|\mathbb{G}|}{\lambda+2} \rceil$ and the number of PRG invocations in $\mathsf{EvalAll}^\bullet$ is at most $2^\nu(1 + \lceil \frac{\log|\mathbb{G}|}{\lambda+2} \rceil)$ for $\nu = \min\{\lceil n - \log \frac{\lambda}{\log|\mathbb{G}|} \rceil, n\}$.*

Proof. The security of the protocol follows immediately from the security of the protocol in Figure 1 since the keys produced by $\mathsf{Gen}^{\bullet,*}(1^\lambda, \alpha, \beta, \mathbb{G})$ are identical to the keys generated by $\mathsf{Gen}^{\bullet}(1^\lambda, (\alpha_1, \ldots, \alpha_\nu), e_{\hat{\alpha},\beta}^{2^{n-\nu}}, \mathbb{G}^{2^{n-\nu}})$, where $(\alpha_1, \ldots, \alpha_\nu)$ is the ν-bit prefix of α.

To prove the correctness of the protocol we observe that for every $x \in \{0,1\}^n$, every $i < \nu$ and any b, the value $\tau^{(i)}$ that $\mathsf{EvalAll}^{\bullet}(b, k_b)$ generates for the tree node associated with the i-bit prefix of x is identical to the $\tau^{(i)}$ value that $\mathsf{Eval}^{\bullet}(b, k_b, x)$ produces. For any $1 \leq i < \nu$ and any x that does not have a shared i-bit prefix with α, the invocations $\mathsf{Eval}^{\bullet}(0, k_0, x)$ and $\mathsf{Eval}^{\bullet}(1, k_1, x)$ produce the same values $\tau^{(j)}$ for any $i \leq j \leq \nu$. Therefore

$$\mathsf{EvalAll}^{\bullet}(0, k_0)[x] - \mathsf{EvalAll}^{\bullet}(1, k_1)[x] = 0.$$

If x shares a prefix of length ν with α then the choice of $CW^{\nu+1,*}$ and the fact that $t_0^{(\nu)} \oplus t_1^{(\nu)} = 1$ ensure that $\mathsf{EvalAll}^{\bullet}(0, k_0)[x] - \mathsf{EvalAll}^{\bullet}(1, k_1)[x] = 0$ for $x \neq \alpha$ and $\mathsf{EvalAll}^{\bullet}(0, k_0)[\alpha] - \mathsf{EvalAll}^{\bullet}(1, k_1)[\alpha] = \beta$.

Each key k_b includes the values $s_b^{(0)}$ and n which are of length $\lambda + \log n$ together and the $\nu + 1$ strings $CW^{(1)}, \ldots, CW^{(\nu)}, CW^{(\nu+1,*)}$. $CW^{(i)}$ is of length $\lambda + 2$ for $i = 1, \ldots, \nu$ and the length $CW^{(\nu+1,*)}$ is $|CW^{(\nu+1,*)}| = \max\{\lambda, \log|\mathbb{G}|\}$. Therefore, the key size is at most $\nu(\lambda + 2) + \lambda + \log n + \max\{\lambda, \log|\mathbb{G}|\}$.

$Gen^{\bullet,*}$ executes the PRG G twice for each $i = 1, \ldots, \nu$ expanding the seeds $s_0^{(i-1)}, s_1^{(i-1)}$ and runs two $\mathsf{Convert}$ operations on $s_0^{(\nu)}, s_1^{(\nu)}$, which is together $2\nu + 2\lceil \frac{\log|\mathbb{G}|}{\lambda+2} \rceil$ PRG operations. The tree that $\mathsf{EvalAll}^{\bullet}$ traverses has 2^ν leaves and $2^\nu - 1$ internal nodes. There is one PRG operation per internal node and at most $\lceil \frac{\log|\mathbb{G}|}{\lambda+2} \rceil$ operations per each leaf, which completes the proof. □

Remark 3.13. In the useful case of $|\mathbb{G}| = 2^k$ elements for $k \leq \lambda$, Theorem 3.12 overestimates the number of PRG invocations in $\mathsf{EvalAll}^{\bullet}$ by a factor of two since $\mathsf{Convert}_{\mathbb{G}}(s)$ for each element in level ν of the tree does not require additional PRG operations. For the same reason, in this case the number of PRG operations in $Gen^{\bullet,*}$ is exactly 2ν and the key size is at most $\nu(\lambda + 2) + 2\lambda + \log n$.

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: (1) a tree topology, (2) variable labels on each node v (where the set of possible values of each variable is known; we denote this set for the variable of node v by S_v), (3) value labels on each edge (the possible values from S_v), and (4) output labels on each leaf node.

In our construction, the key size is roughly $\lambda \cdot |V|$ bits, where V is the set of nodes, and evaluation on a given input requires $|V|$ executions of a pseudorandom generator, and a comparable number of additions. The FSS is guaranteed to hide the secret edge value labels and leaf output labels (which we refer to as "Decisions"), but (in order to achieve this efficiency) reveals the base tree topology and the identity of which variable is associated to each node (we refer to this collective revealed information 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 variable 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 outgoing edge, labeled by 1, terminating in a leaf labeled by 1. (See Figure 5(a).) 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.

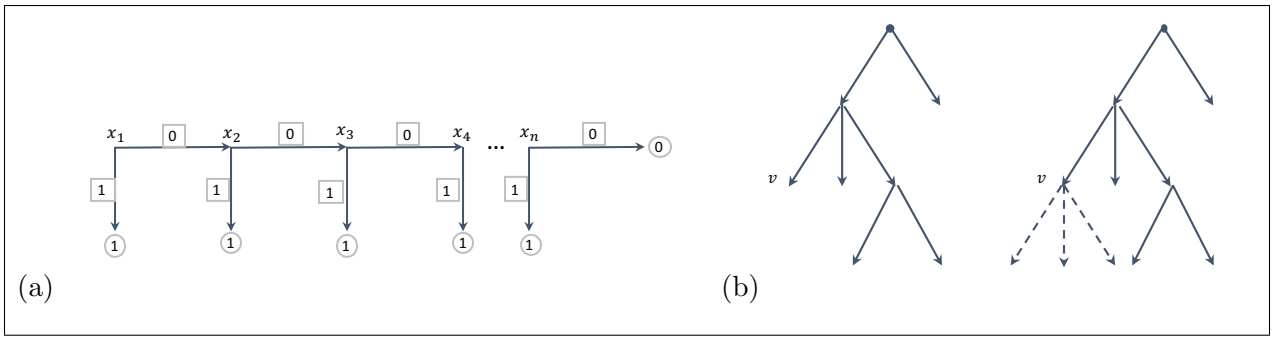

Figure 5: (a) Decision tree for OR function on input $x_1, \ldots, x_n$. Revealed information: Tree consists of graph topology and labels of nodes by input variables x_i. Hidden information: Decisions includes edge labels (boxed) and leaf output labels (circled). (b) Depiction of v-extension tree.

A richer sample application is the class of constant-dimension intervals. Direct application of our FSS for decision trees construction yields FSS for d-dimensional interval functions on input size n with key size $O(\lambda \cdot n^d)$.

In the following three subsections, we present notation and syntax for decision trees, describe our construction of FSS for decision trees, and further discuss the implications to constant-dimension intervals.

3.3.1 Decision Tree Preliminaries

We begin with a formal definition of the decision tree model.

Definition 3.14 (Decision Trees: Syntax)**.** Let a *decision tree* be specified by a tuple

$$\Pi_{\mathsf{tree}} = (n, (1^{\ell_i})_{i \in [n]}, \mathbb{G}, \mathsf{Tree}, \mathsf{Decisions})$$

consisting of the following information:

1. $n \in \mathbb{N}$, indicating the number of input variables to the program, which are denoted by $x_1, \ldots, x_n$.
2. For each $i \in [n]$, the domain size ℓ_i of the ith variable, specified in unary.
3. A description of the output group $\mathbb{G}$. (See Section 2 for discussion.)
4. Description $\mathsf{Tree} = (\Gamma, \mathsf{var})$ of tree structure:
   - Directed tree $\Gamma = (V, E)$.
   - Labeling $\mathsf{var} : V \to [n]$ of each *internal node* in the tree $v \in V$ by an input variable index.

   The degree of each node v must be $\leq \ell_i$ for which $\mathsf{var}(v) = i$. (We sometimes denote this value by ℓ_v.)
5. Decision labels $\mathsf{Decisions} = (D, \mathsf{output})$ of two kinds:
   - For each directed *edge* $(u, v) \in E$ from parent to child, a label $D(u, v) \in \mathbb{Z}_{\ell_u}$, with the property that $D(u, v) \neq D(u, v')$ for $v \neq v'$ (i.e., each of the outgoing edges have distinct labels in $\mathbb{Z}_{\ell_u}$).
   - Output labels for each *leaf* node u in the tree, denoted $\mathsf{output}(u) \in \mathbb{G}$.

Let $v \leftarrow \mathsf{root}(\Gamma)$ output the root of the tree, $b \leftarrow \mathsf{IsLeaf}(v)$ output a bit identifying whether v is a leaf node, $\ell \leftarrow \mathsf{deg}(v)$ return the degree (number of children) of node v, $u \leftarrow \mathsf{parent}(v)$ output the parent of v, $v \leftarrow \mathsf{child}(u, a)$ output the child node of u that corresponds to value $a \in \mathbb{Z}_{\ell_u}$ of

its input variable, and $\{v_a\}_a \leftarrow \mathsf{siblings}(v)$ output the set of all siblings of v (including v itself). Let $x \leftarrow \mathsf{input}(v)$ output an input value $x \in \mathbb{Z}_{\ell_1} \times \cdots \times \mathbb{Z}_{\ell_n}$ which leads to node v: i.e., for which $x_{\mathsf{var}(u)} = D(u, w)$ for every edge (u, w) from $\mathsf{root}(\Gamma)$ to v.

Definition 3.15 (Decision Trees: Semantics)**.** A decision tree $\Pi_{\mathsf{tree}} = (n, (1^{\ell_i})_{i\in[n]}, \mathbb{G}, \mathsf{Tree}, \mathsf{Decisions})$ computes the function $f_\Pi : \mathbb{Z}_{\ell_1} \times \cdots \times \mathbb{Z}_{\ell_n} \to \mathbb{G}$ which evaluates on an input $(x_1, \ldots, x_n)$ as:

1: Let $u \leftarrow \mathsf{root}(\mathsf{Tree})$ and $i \leftarrow \mathsf{var}(u)$
2: **while** $\mathsf{IsLeaf}(u) = 0$ **do**
3: Let $u \leftarrow \mathsf{child}(u, x_i)$
4: **end while**
5: **return** $\mathsf{output}(u)$.

In our FSS construction, it will be helpful to consider the following notion of extending a tree by adding a collection of new children to an originating leaf node (see Figure 5(b)).

Definition 3.16 (v-Extension)**.** For given n, $(1^{\ell_i})_{i\in[n]}$ and $\mathbb{G}$, we say that Tree' is a *v-extension* of Tree if:

- The directed graph $\Gamma = (V, E)$ is formed by adding an additional level of children to the leaf node v in $\Gamma = (V, E)$. That is, $V' = V \cup \{w_j\}_{j\in\mathbb{Z}_\ell}$ and $E \subseteq E' \subseteq E \cup \{(v, w_j)\}_{j\in\ell}$, where $\ell = \ell_i$ for some $i \in [n]$.
- The labeling of internal nodes as per Tree' is consistent with Tree: $\forall u \in \{u \in V : \mathsf{IsLeaf}(u) = 0\}$, $\mathsf{var}(u) = \mathsf{var}'(u)$. Further $\mathsf{var}(v) = i \in [n]$ for the ℓ_i as above.

3.3.2 Constructing FSS for Decision Trees

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, this approach results in key size and evaluation time comparable to $\sum_{v\in V} \mathrm{depth}(v)$ instead of $|V|$, corresponding to an overhead of the average depth. Instead, we show how to "reuse the shared backbone" of the tree. In particular, our DPF constructions have the property that a DPF key contains within it explicitly DPF keys for each of its prefixes. Our construction thus directly applies the tensoring approach to "append" each node onto the backbone structure, one by one.

More specifically, our construction is recursive on the tree graph topology, starting with the root and iteratively appending children. In each step, we begin with an FSS scheme $(\mathsf{Gen}^{\mathsf{Tree}}, \mathsf{Eval}^{\mathsf{Tree}})$ for the class of decision trees

$$\{\Pi = (n, (1^{\ell_i})_{i\in[n]}, \mathbb{G} = \{0,1\}^{\lambda+1}, \mathsf{Tree}, \mathsf{Decisions})\}_{\mathsf{Decisions}}$$

for a given $n, (1^{\ell_i})_{i\in[n]}$ and with structure as specified by some fixed value of Tree (dictating tree topology and variable node labeling), but varying over all possible choices of the secret values $\mathsf{Decisions}$. Further, the FSS evaluation decomposes into the following special structure: $\mathsf{Eval}^{\mathsf{Tree}}(b, x)$ is computed as a sum of separate contributions for each leaf node v,

$$\mathsf{Eval}^{\mathsf{Tree}}(b, x) = \sum_{v:\mathsf{IsLeaf}(v)} \mathsf{Eval}_v^{\mathsf{Tree}}(b, x),$$

(with addition in $\mathbb{G}$), for which $\mathsf{Eval}_v^{\mathsf{Tree}}(b, x) = 0$ for all v except (at most) one, such that x leads to node v in the labeled tree. We will refer to this as a *leaf-decomposable* FSS for decision trees. Note that this structure can be trivially achieved for the base case of a single-node tree by directly secret sharing the single-output truth table.

Given such a leaf-decomposable FSS $(\mathsf{Gen}^{\mathsf{Tree}}, \mathsf{Eval}^{\mathsf{Tree}})$, we will extend to an FSS for the analogous class of functions on a "v-extension" graph Tree' of Tree, i.e. adding an additional level of children to a leaf node $v \in \mathsf{Tree}$, as in Definition 3.16. Note for every leaf $v' \neq v$ in the

original Tree, the v'-function $\mathsf{Eval}_{v'}^{\mathsf{Tree}'} = \mathsf{Eval}_{v'}^{\mathsf{Tree}}$ will remain unchanged. The task is to convert $\mathsf{Eval}_{v}^{\mathsf{Tree}}$ for the special leaf v' (now an internal node) into a collection of new functions $\mathsf{Eval}_{w}^{\mathsf{Tree}'}$, one for each newly introduced leaf w. Let $i = \mathsf{var}(v)$ be the input variable index that v is labeled with in $\mathsf{Decisions}'$, let ℓ be the corresponding specified domain size, and let $w_1, \ldots, w_\ell$ be the new children nodes of v in Tree'. Note that we must specify ℓ new functions $\mathsf{Eval}_{w}^{\mathsf{Tree}'}$, where each is of the form $[x \in \mathsf{input}(v)] \cdot [x_i = D'(v,w)] \cdot \mathsf{output}'(w)$: that is, if the input x agrees with the path down to v, and further down to leaf w, then output the value $\mathsf{output}'(w)$ specified by $\mathsf{Decisions}'$; otherwise output 0. Collectively, *all* ℓ of the $\mathsf{Eval}_{w}^{\mathsf{Tree}'}$ functions can be described by

$$[x \in \mathsf{input}(v)] \cdot \Big([x_i = D'(v,w_1)] \cdot \mathsf{output}'(w_1) \Big|\Big| \cdots \Big|\Big| [x_i = D'(v,w_\ell)] \cdot \mathsf{output}'(w_\ell) \Big),$$

where $[x \in \mathsf{input}(v)]$ and $[x_i = \mathsf{val}]$ denote predicates on x and x_i, respectively. This overall expression can be viewed as a special form of tensor product, as in the sense of Section 3.1.

Recall the tensor approach of Section 3.1 combines a DPF together with an FSS for an arbitrary function class $\mathcal{G}$, yielding FSS for a tensor product of the two function classes. In our setting, the function $\mathsf{Eval}_{v}^{\mathsf{Tree}}$ will serve as the DPF (indeed, it evaluates to 0 everywhere except a single specific path in the tree). The second function class $\mathcal{G}$ will be chosen to encode the concatenated functions in the parentheses above, corresponding to the newly added set of leaves $w_1, \ldots, w_\ell$. Namely, each function $[x_i = D'(v,w)] \cdot \mathsf{output}'(w)$ admits a simple FSS, corresponding to a secret shared truth table for all ℓ possible values of the input x_i. The size of each such truth table is $(\ell)(\log|\mathbb{G}|)$, as there are ℓ inputs and each output is an element of $\mathbb{G}$. Collectively, an FSS of the concatenated functions $[x_i = D'(v,w_1)] \cdot \mathsf{output}'(w_1) || \cdots || [x_i = D'(v,w_\ell)] \cdot \mathsf{output}'(w_\ell)$, which forms the second function class $\mathcal{G}$, has size $(\ell)^2(\log|\mathbb{G}|)$, since there are ℓ concatenated copies.

A description of the resulting FSS scheme $(\mathsf{Gen}^{\mathsf{DT}}, \mathsf{Eval}^{\mathsf{DT}})$ is given in Figure 6. For simplicity, in the figure we describe the case where the output group $\mathbb{G}$ is $\{0,1\}^{\lambda+1}$, i.e., where a final $\mathbb{G}$ correction stage is not needed.

Iteratively performing the above-described appending procedure results in the following main theorem.

Theorem 3.17 (FSS for Decision Trees)**.** *Fix $\ell_{\mathsf{max}} \in \mathbb{N}$. There exists a polynomial p such that given a (T,ϵ)-secure pseudorandom generator $G : \{0,1\}^\lambda \to \{0,1\}^{(\lambda+1)\ell_{\mathsf{max}}^2}$, there exists a (T',ϵ')-secure FSS for the class of decision trees*

$$\{\Pi_{\mathsf{tree}} = (n, (1^{\ell_i})_{i\in[n]}, \mathbb{G} = \{0,1\}^{\lambda+1}, \mathsf{Tree}, \mathsf{Decisions})\}_{n,(\ell_i \le \ell_{\mathsf{max}})_{i\in[n]}, \mathsf{Tree}, \mathsf{Decisions}},$$

with leakage function $\mathsf{Leak}(\Pi_{\mathsf{tree}}) = (n, (1^{\ell_i})_{i\in[n]}, \mathbb{G}, \mathsf{Tree})$, for $T' = T + p(n)$ and $\epsilon' = |V| \cdot \epsilon_{\mathsf{PRG}}$, with key size

$$\lambda + \sum_{v \in V : \neg \mathsf{IsLeaf}(v)} (\lambda+1)\ell_v \mathsf{deg}(v) + \sum_{v \in V : \mathsf{IsLeaf}(v)} \log|\mathbb{G}|,$$

whose evaluation on an input x requires $|V| - |\{v \in V : \mathsf{IsLeaf}(v)\}|$ calls to G.

For instance, for the special case of binary decision trees $\ell = 2$ with output $\mathbb{G} = \{0,1\}^\lambda$, the corresponding key size is bounded by $2|V|(\lambda+1)$ bits.

Proof. We prove by induction on the tree structure, growing up via v-extensions. Namely, the proof of Theorem 3.17 follows directly from the following two sub-claims.

Claim 3.18 (Base Case)**.** *There exists a $(T,0)$-secure FSS scheme (for any T) for the trivial class of decision trees whose topology consists of a single root node (which is also a leaf node) v:*

$$\big\{\Pi_{\mathsf{tree}} = (n, (1^{\ell_i})_{i\in[n]}, \mathbb{G}, \mathsf{Tree}, \mathsf{Decisions}) \big| \Gamma = (V = \{v\}, E = \emptyset)\big\}_{n,(\ell_i \le \ell_{\mathsf{max}})_{i\in[n]}, \mathsf{Decisions}},$$

with leakage function $\mathsf{Leak}(\Pi_{\mathsf{tree}}) = (n, (1^{\ell_i})_{i\in[n]}, \mathbb{G}, \mathsf{Tree})$, and with key size $\log|\mathbb{G}|$ bits.

Proof. Follows from the trivial FSS scheme. The size of the truth table of any program Π_{tree} as above is precisely $\log|\mathbb{G}|$ bits: namely, the description of one group element. (Note that this is precisely the class of constant functions.) □

Claim 3.19 (Recursive Step)**.** *Let* $\mathsf{Tree}' = (\Gamma', \mathsf{var}')$ *be a* v*-extension of* $\mathsf{Tree} = (\Gamma, \mathsf{var})$ *for some leaf node* $v \in \Gamma$*. There exists a polynomial* p *such that the following holds. If there exist*

- $(T, \epsilon_{\mathsf{PRG}})$*-secure pseudorandom generator* $G : \{0,1\}^\lambda \to \{0,1\}^{(\lambda+1)\ell_{\mathsf{max}}^2}$*,*
- $(T, \epsilon_{\mathsf{FSS}})$*-secure FSS for class of decision trees with structure* Tree

$$\left\{\Pi_{\mathsf{Tree}} = (n, (1^{\ell_i})_{i\in[n]}, \mathbb{G} = \{0,1\}^{\lambda+1}, \mathsf{Tree}, \mathsf{Decisions})\right\}_{n,(\ell_i \le \ell_{\mathsf{max}})_{i\in[n]},\mathsf{Decisions}},$$

for the fixed tree structure Tree*, with leakage function* $\mathsf{Leak}(\Pi_{\mathsf{Tree}}) = (n, (1^{\ell_i})_{i\in[n]}, \mathbb{G}, \mathsf{Tree})$*, and with key size* $s(\lambda, n, (\ell_i)_{i\in[n]})$*, requiring* $t(n)$ *PRG calls for evaluation,*

then there exists a (T', ϵ')*-secure FSS for the class of decision trees with topology* Tree'*:*

$$\left\{\Pi_{\mathsf{Tree}'} = (n, (1^{\ell_i})_{i\in[n]}, \mathbb{G} = \{0,1\}^{\lambda+1}, \mathsf{Tree}', \mathsf{Decisions})\right\}_{n,(\ell_i \le \ell_{\mathsf{max}})_{i\in[n]},\mathsf{Decisions}},$$

with leakage function $\mathsf{Leak}(\Pi_{\mathsf{Tree}'}) = (n, (1^{\ell_i})_{i\in[n]}, \mathbb{G}, \mathsf{Tree}')$ *for* $T' = T + p(n)$ *and* $\epsilon' = \epsilon_{\mathsf{PRG}} + \epsilon_{\mathsf{FSS}}$*, with key size* $s(\lambda, n, (\ell_i)_{i\in[n]}) + \ell_v \log|\mathbb{G}|$*, requiring* $t(n) + 1$ *PRG calls for evaluation.*

Proof. Follows by the correctness and security analyses of the tensor operation (Section 3.1). □

This concludes the proof of Theorem 3.17. □

3.3.3 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 $a_i \le x_i \le b_i$ for some secret interval ranges $(a_i, b_i)_{i\in[d]}$. (See, e.g., [37] 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 3.20. *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 [8]).

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

FSS For Decision Trees $(\mathsf{Gen}^{\mathsf{DT}}, \mathsf{Eval}^{\mathsf{DT}})$ for bounded ℓ_{max}
Let $G : \{0,1\}^\lambda \to \{0,1\}^{(\lambda+1)\ell_{\mathsf{max}}^2}$ be pseudorandom generator.
Let $\mathsf{Convert} : \{0,1\}^\lambda \to \mathbb{G}$ be a map converting a random λ-bit string to a pseudorandom group element of $\mathbb{G}$ (see Figure 3).

$\mathsf{Gen}^{\mathsf{DT}}(1^\lambda, (n, (1^{\ell_i})_{i\in[n]}, \mathbb{G} = \{0,1\}^{\lambda+1}, (\Gamma = (V,E), \mathsf{var}), (D, \mathsf{output}))$:

while $\exists v \in V$ st $(v \neq \mathsf{root}(\Gamma)) \wedge \forall w \in \mathsf{siblings}(v)$ $(\mathsf{IsLeaf}(w))$ **do**
  $u \leftarrow \mathsf{parent}(v)$
  $V' \leftarrow V \setminus \mathsf{siblings}(v)$, $E' \leftarrow E \setminus \{(u,w) : w \in \mathsf{siblings}(v)\}$
  Let var', D' be restrictions of var, D to $\Gamma' = (V', E')$
  For $v \neq w \in V'$, let $\mathsf{output}'(w) \leftarrow \mathsf{output}(w)$
  Sample random $s \leftarrow \{0,1\}^\lambda$. Set $\mathsf{output}'(v) \leftarrow [s||1] \in \{0,1\}^{\lambda+1}$
  Generate FSS key $(\sigma_0^{(u)}, \sigma_1^{(u)}) \leftarrow \mathsf{Gen}^{\mathsf{DT}}(1^\lambda, (n, (1^{\ell_i})_{i\in[n]}, \mathbb{G}, (\Gamma', \mathsf{var}'), (D', \mathsf{output}')))$
  Let $x \leftarrow \mathsf{input}(u)$ be an input leading to node v in the tree
  Compute $s_b||t_b \leftarrow \mathsf{Eval}^{\mathsf{DT}}(b, \sigma_b, x)$ for $b \in \{0,1\}$ // Corresponds to $\mathsf{Eval}_u^{\mathsf{DT}}$
  Let $i = \mathsf{var}(u)$, ℓ_u the corresponding domain size, and $\mathsf{siblings}(v) = \{w_1, \ldots, w_{\mathsf{deg}(u)}\}$
  For each $w \in \mathsf{siblings}(v)$, denote by $f_w(x_i) := [x_i = D(v,w)] \cdot \mathsf{output}(w)$
  Let $cw^{(u)} \leftarrow \mathsf{TruthTable}(f_{w_1}||f_{w_2}||\cdots||f_{w_{\mathsf{deg}(u)}}) \in \{0,1\}^{(\lambda+1)\ell_u \mathsf{deg}(u)}$
  Let $CW^{(u)} \leftarrow (-1)^{t_1}[cw^{(v)} - G(s_0) + G(s_1)]$, with addition in $\{0,1\}^{(\lambda+1)\ell_u \mathsf{deg}(u)}$ (ignoring extra output bits of G if $\ell_u \mathsf{deg}(u) < \ell_{\mathsf{max}}^2$).
end while
Let $k_b \leftarrow \sigma_b^{(\mathsf{root}(\Gamma))}, \Gamma, \mathsf{var}, (CW^{(v)})_{v\in V}$
return (k_0, k_1)

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

Parse $k_b = \sigma_b^{(\mathsf{root}(\Gamma))}, \Gamma, \mathsf{var}, (CW^{(v)})_{v\in V}$. (We will reference Γ, var, and $(CW^{(v)})_{v\in V}$ globally)
Parse $\sigma_b^{(\mathsf{root}(\Gamma))} = s_b^{(\mathsf{root}(\Gamma))}||t_b^{(\mathsf{root}(\Gamma))}$
return $\mathsf{EvalNode}(\mathsf{root}(\Gamma), s_b^{(\mathsf{root}(\Gamma))}, t_b^{(\mathsf{root}(\Gamma))}, CW^{(\mathsf{root}(\Gamma))})$

$\mathsf{EvalNode}(v, s^{(v)}, t^{(v)}, CW^{(v)})$:

Let $\tau^{(v)} \leftarrow G(s^{(v)}) \oplus (t^{(v)} \wedge CW^{(v)})$
Parse $\tau^{(v)} = (\sigma^{(w_1)}||\sigma^{(w_2)}||\cdots||\sigma^{(w_{\mathsf{deg}(v)})})$. ($\sigma^{(w)}$ is a secret share of $\mathsf{TruthTable}(f_w)$, as above)
Let $i = \mathsf{var}(v)$
for $w \in \mathsf{children}(v)$ **do**
  if $\mathsf{IsLeaf}(w)$ **then**
    $\mathsf{FnOutput}^{(w)} \leftarrow \sigma^{(w)}(x_i)$ // i.e., secret share of $f_w(x_i)$ as given by $\sigma^{(w)}$. This is $\mathsf{Eval}_w^{\mathsf{DT}}$.
  else
    let $s^{(w)}||t^{(w)} \leftarrow \sigma^{(w)}(x_i)$.
    $\mathsf{FnOutput}^{(w)} \leftarrow \mathsf{EvalNode}(w, s^{(w)}, t^{(w)}, CW^{(w)})$
  end if
end for
return $\sum_{w\in\mathsf{children}(v)} \mathsf{FnOutput}^{(w)} \in \mathbb{G}$

Figure 6: Pseudocode for FSS $(\mathsf{Gen}^{\mathsf{DT}}, \mathsf{Eval}^{\mathsf{DT}})$ for Decision Trees. $\mathsf{Gen}^{\mathsf{DT}}$ generates the key iteratively, peeling off one set of siblings at a time. $\mathsf{Eval}^{\mathsf{DT}}$ recursively calls the $\mathsf{EvalNode}$ subroutine, from the root down.

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 $\mathcal{F}$ for which the output domain of each $f \in \mathcal{F}$ is equipped with a ring structure R_f. For two functions f_1, f_2 with the same input domain $D_{f_1} = D_{f_2}$ and output ring $R_{f_1} = R_{f_2}$, we naturally define the product $f = f_1 \cdot f_2$ by $f(x) = f_1(x)f_2(x)$.

Definition 3.21 (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.

We now describe the implementation of the FSS product operator, using (m, t)-FSS as an abbreviation for an m-party t-secure FSS scheme.

Theorem 3.22. *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*.*

Proof. We label the $m_1 m_2$ parties by pairs in $[m_1] \times [m_2]$ and define $(\mathsf{Gen}, \mathsf{Eval})$ as follows.

- $\mathsf{Gen}(1^\lambda, f_1 \cdot f_2)$ lets $(k_1^1, \ldots, k_{m_1}^1) \leftarrow \mathsf{Gen}_1(1^\lambda, f_1)$ and $(k_1^2, \ldots, k_{m_2}^2) \leftarrow \mathsf{Gen}_2(1^\lambda, f_2)$. It outputs the keys $k_{i_1,i_2} = (k_{i_1}^1, k_{i_2}^2)$ for $(i_1, i_2) \in [m_1] \times [m_2]$.
- $\mathsf{Eval}((i_1, i_2), k_{i_1,i_2}, x)$ outputs $\mathsf{Eval}_1(i_1, k_{i_1}, x) \cdot \mathsf{Eval}_2(i_2, k_{i_2}, x)$.

Correctness follows from the fact that $\left(\sum_{i_1} f_{i_1}\right) \cdot \left(\sum_{i_2} f_{i_2}\right) = \sum_{i_1,i_2} f_{i_1} \cdot f_{i_2}$. The t-secrecy property follows from the t-secrecy of Gen_1 and Gen_2, noting that every set of t party indices (i_1, i_2) covers at most t distinct i_1 indices and t distinct i_2 indices. □

As a corollary, one can compose a 2-party FSS scheme for $\mathcal{F}$ with itself $d \geq 2$ times, obtaining a $(2^d, 1)$-FSS for $\mathcal{F}^d$ with comparable per-party efficiency. Combining with a simple closure under addition [8], one can similarly obtain a $(2^d, 1)$-FSS for the class of degree-d multivariate polynomials in functions of $\mathcal{F}$.

Finally, we show how to reduce the number of parties required by the product operator by lowering the security threshold. We focus here on the simplest special case, where the given FSS schemes have a full security threshold and the same number of parties.

Theorem 3.23. *Let* $(\mathsf{Gen}_i, \mathsf{Eval}_i)$ *be an* $(m, m-1)$*-FSS for* $\mathcal{F}_i$*, for* $1 \leq i \leq m-1$*. Then, there exists an* $(m, 1)$*-FSS* $(\mathsf{Gen}, \mathsf{Eval})$ *for* $\mathcal{F} = \prod_{i=1}^{m-1} \mathcal{F}_i$ *in which the size of the key produced by* Gen *on* $f_1 \cdot f_2 \cdots f_{m-1}$ *is* $m-1$ *times the total key sizes of* Gen_i *on* f_i*.*

Proof. We define $(\mathsf{Gen}, \mathsf{Eval})$ as follows.

- $\mathsf{Gen}(1^\lambda, f_1 \cdot f_2 \cdots f_{m-1})$ lets $(k_1^i, \ldots, k_m^i) \leftarrow \mathsf{Gen}_i(1^\lambda, f_i)$, for $1 \le i \le m-1$. It outputs the keys $k_i = (k_j^i)_{j \neq i}$.
- $\mathsf{Eval}(i, (k_j^i)_{j \neq i}, x)$ is defined by assigning each monomial of the form $\prod_{j=1}^{m-1} f_{j,i_j}$ to the first party i such that $i \notin \{i_1, \ldots, i_{m-1}\}$ (such i exists because there are m parties and only $m-1$ indices i_j). Eval outputs the sum of the products of all monomials assigned to it on the input x, where $f_{j,i_j}(x) = \mathsf{Eval}(i_j, k_j^{i_j}, x)$.

Correctness again follows by expressing a product of $m-1$ sums as a sum of monomials, noting that each monomial is added to exactly one output of Eval. The 1-secrecy property follows from the fact that each party gets $m-1$ out of the m keys produced by each Gen_i and the $(m-1)$-security of Gen_i. □

As a concrete instance, using a 3-party PRG-based DPF construction from [8] (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 the m keys are generated by a potentially malicious client, but are processed by *semi-honest* servers, to which we will also refer 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 communicate 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. In contrast, a previous verification protocol from [15], which applies to a specific DPF scheme, involves an additional party, and has a relatively high asymptotic communication complexity (roughly the square root of the domain size).

We would like our verification protocols to only make a *black-box* use of the underlying FSS scheme. Combined with our concrete FSS schemes, this implies that the protocols make a black-box use of a PRG. The latter requirement (which is also satisfied by the verification protocol from [15]) 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 $D = zo^n$ 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. It is also motivated by "writing" applications where only a strict subset of the input space is being updated. This is the case, for example, for the web usage statistics example described in the Introduction. However, in other applications of FSS, including PIR, private updates (see Appendix A) 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 length n and output group $\mathbb{G}$ 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 $(\mathsf{Gen}, \mathsf{Eval})$ is an FSS scheme 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 length 1^n describing an input domain $D = \{0,1\}^n$ of (an unknown) $f \in \mathcal{F}$, output group $\mathbb{G}$ 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 $D = \{0,1\}^n$ and output group $\mathbb{G}$, then for all $D' \subseteq D$, in the execution of Ver on local inputs $(k_1, \ldots, k_m)$ and common inputs $D, D', \mathbb{G}$, all parties output "Accept" with

probability 1.

- **Soundness:** Consider the following security experiment defined by an efficient non-uniform adversary $\mathcal{A}$ running on input 1^λ:
  1: $\mathcal{A}(1^\lambda)$ outputs an FSS input length n describing the input domain $D = \{0,1\}^n$, Abelian group $\mathbb{G}$, FSS keys $(k_1^*, \ldots, k_m^*)$, and a set $D' \subseteq D$ (represented by an explicit list of elements or $\emptyset$ to indicate that $D' = D$).
  2: The protocol Ver is executed on local inputs $(k_1^*, \ldots, k_m^*)$ and common inputs $1^\lambda, D, \mathbb{G}, D'$.
  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 $\mathbb{G}$ 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 $D' \subseteq D$), the joint view of any t parties should not reveal anything about f except its input domain D and output group $\mathbb{G}$. 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.[3] 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 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; it suffices 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)

[3] 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.

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 7 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 function families $\mathcal{F}$ and support very efficient MPC protocols for $\mathcal{V}$. All instantiations rely on the standard Schwartz-Zippel (SZ) lemma, which we quote below for convenience.

Lemma 4.2 (Schwartz-Zippel [36, 40])**.** *Let P be a nonzero N-variate polynomial over a field $\mathbb{F}$ with total degree d and let $S \subseteq \mathbb{F}$ be a finite set. Then* $\Pr_{\mathbf{r} \in S^N}[P(\mathbf{r}) = 0] \leq d/|S|$.

We now separately consider different choices of function families $\mathcal{F}$, starting with restricted classes of point functions.

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 $\mathcal{V}_{\mathsf{sq}}(z_1, z_2) = z_1^2 - z_2 = 0$.

Claim 4.3. *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*

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

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 $\mathbf{z}^i$, possibly using k_i^* as a source of correlated randomness, to evaluate $\mathcal{V}(\mathbf{z})$. Accept if the output of Π_{MPC} is the all-0 vector, otherwise reject.

Figure 7: Template for FSS verification protocol.

Proof. Let $\mathbf{y} = (y_1, \ldots, y_N)$. The condition $z_1^2 - z_2 = 0$ can be written as

$$\left(\sum_{j=1}^{N} y_j r_j\right)^2 - \sum_{j=1}^{N} y_j r_j^2 = \sum_{j=1}^{N} (y_j^2 - y_j) r_j^2 + \sum_{1 \leq j < k \leq N} (2 y_j y_k) \cdot r_j r_k = 0$$

We view the above expression as a degree-2 polynomial in $r_1, \ldots, r_N$ whose coefficients are determined by $\mathbf{y}$. Now, suppose $\mathbf{y}$ does not satisfy the assumption. Then either it has two nonzero entries y_j, y_k, in which case the coefficient $2y_j y_k$ of the monomial $r_j r_k$ is nonzero (here we use the fact that $p > 2$) or $\mathbf{y}$ has an entry j such that $y_j \notin \{0,1\}$, in which case the coefficient $y_j^2 - y_j$ of the monomial r_j^2 is nonzero. The conclusion now follows from the SZ lemma. □

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 $\mathcal{V}_{\mathsf{prod}}(z_1, z_2, z_3) = z_1 z_2 - z_3$. The proof of the following claim is very similar to that of Claim 4.3 and is thus omitted.

Claim 4.4. *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*

$$\Pr\left[L \leftarrow \mathcal{L}_{\mathsf{prod}}(\mathbb{F}, N); (z_1, z_2, z_3) \leftarrow L \cdot \mathbf{y} \,:\, z_3 = z_1 z_2\right] \leq 2/|\mathbb{F}|.$$

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 4.5. *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*

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

Proof. The condition $z_1 z_2 = 1$ can be written as

$$\left(\sum_{j=1}^{N} y_j r_j\right) \cdot \left(\sum_{j=1}^{N} y_j r_j^{-1}\right) = 1. \tag{1}$$

Let $R = \Pi_{j=1}^{N} r_j$ and $R_{j,k} = R \cdot r_j / r_k$ (we view R and $R_{j,k}$ as distinct formal monomials in $r_1, \ldots, r_N$). Multiplying both sides of Eq. (1) by R and grouping monomials we get:

$$\left(\sum_{j=1}^{N} y_j^2 - 1\right) \cdot R + \sum_{1 \leq j \neq k \leq N} (y_j y_k) \cdot R_{j,k} = 0.$$

Now, suppose $\mathbf{y}$ satisfies the assumption and consider the following two cases. If $\mathbf{y}$ has at least two nonzero entries j, k, then $y_j y_k \neq 0$, in which case the monomial $R_{j,k}$ has nonzero coefficient. If $\mathbf{y}$ has exactly one nonzero entry and this entry different from $1, -1$, or alternatively if $\mathbf{y}$ is the all-0 vector, then the coefficient of the monomial R is nonzero. The conclusion again follows from the SZ lemma. □

DPF $f_{\alpha,\beta}$ with $\beta = 1$. The above sketching schemes allow β to take two possible values, except for the scheme for $\beta \in \{1,-1\}$ over fields of characteristic 2. For general fields, 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 $\mathcal{V}'(z_1, z_2, z_3) = (\mathcal{V}(z_1, z_2), z_3 - 1)$.

A more direct approach is to use the following variant $\mathcal{L}'_{\mathsf{sq}}$ of $\mathcal{L}_{\mathsf{sq}}$. The matrix $L \in \mathbb{F}^{2\times N}$ is defined by $L_{1,j} = r_j$ and $L_{2,j} = (r_j + 1)^2$. The verification predicate $\mathcal{V}'_{\mathsf{sq}}$ checks that the sketch $\mathbf{z} = (z_1, z_2)$ satisfies the condition $\mathcal{V}'_{\mathsf{sq}}(z_1, z_2) = (z_1 + 1)^2 - z_2 = 0$.

Claim 4.6. *Let $\mathbb{F}$ be a finite field of characteristic $p > 2$. If $\mathbf{y} \in \mathbb{F}^N$ is not a unit vector, then*

$$\Pr\left[L \leftarrow \mathcal{L}'_{\mathsf{sq}}(\mathbb{F}, N); (z_1, z_2) \leftarrow L \cdot \mathbf{y} \,:\, z_2 = (z_1 + 1)^2\right] \leq 2/|\mathbb{F}|.$$

Proof. Let $\mathbf{y} = (y_1, \ldots, y_N)$. The condition $(z_1 + 1)^2 - z_2 = 0$ can be written as

$$\left(\sum_{j=1}^{N} y_j r_j + 1\right)^2 - \sum_{j=1}^{N} y_j (r_j + 1)^2 = \sum_{j=1}^{N} (y_j^2 - y_j) r_j^2 + \sum_{1 \leq j < k \leq N} (2 y_j y_k) \cdot r_j r_k - \sum_{j=1}^{N} y_j + 1 = 0$$

We view the above expression as a degree-2 polynomial in $r_1, \ldots, r_N$ whose coefficients are determined by $\mathbf{y}$. Now, suppose $\mathbf{y}$ does not satisfy the assumption. Then either (1) it has two nonzero entries y_j, y_k, in which case the coefficient $2y_jy_k$ of the monomial r_jr_k is nonzero (here we use the fact that $p > 2$), or (2) $\mathbf{y}$ has an entry j such that $y_j \notin \{0, 1\}$, in which case the coefficient $y_j^2 - y_j$ of the monomial r_j^2 is nonzero, or (3) $\mathbf{y}$ is the all-0 vector, in which case the free coefficient is nonzero. The conclusion now follows from the SZ lemma. □

Remark 4.7 (On the case $D' \subset D$.)**.** Note that the later two sketching schemes (namely, all those that do not allow $\beta = 0$) are only useful when $D' = D$. When D' is a strict subset of D, the verifiable FSS scheme obtained from these sketching schemes would fail to be *complete* when $\alpha \notin D'$, because in this case $\mathbf{y}$ is the all-0 vector which should be rejected.

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 7 by allowing the verification predicate $\mathcal{V}$ to be non-deterministic. That is, $\mathcal{V}$ may depend on an additional input $w \in \mathbb{F}$ that is secret-shared by the client as part of Gen. Such a non-deterministic verification procedure should satisfy the following requirements. First, given an honestly generated w, the verification should succeed. Second, even a maliciously generated w should not 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:

- $\mathcal{V}'_{\mathsf{sq}}(z_1, z_2, w) = z_1^2 - z_2w$, where an honest client uses $w = \beta$.
- $\mathcal{V}'_{\mathsf{prod}}(z_1, z_2, z_3, w) = z_1z_2 - z_3w$, where an honest client uses $w = \beta$.
- $\mathcal{V}'_{\mathsf{inv}}(z_1, z_2, w) = z_1z_2 - w$, 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.

DPF $f_{\alpha,\beta}$ with unrestricted $\beta \in \mathbb{F}^d$. It is sometimes useful to verify a DPF whose range is of the form $\mathbb{F}^d$ (namely, the additive group of length-d vectors over $\mathbb{F}$). This easily reduces to the previous case by viewing $\mathbb{F}^d$ as the additive group of the degree-d extension of $\mathbb{F}$. However, this reduction is computationally inefficient because due to the cost of multiplications in a big field. Instead, one can use the following more efficient alternative. Viewing all outputs as a $d \times N$ matrix, the goal is to verify that the matrix has at most one nonzero row. A natural approach is to take a random linear combination of the rows and checks that the resulting vector has at most one nonzero entry. But checking the latter using the procedure described above requires the client to secret-share the value of the nonzero entry, which is not known a-priori (since it depends on the linear combination).

One way around the problem is to reveal the linear combination to the client and then have the client secret-share the nonzero value resulting from this linear combination. But this adds more interaction that we would ideally like to avoid. A non-interactive alternative is the following:

1. Check that every row has Hamming weight at most 1. This can be done by applying the previous verification procedure (DPF $f_{\alpha,\beta}$ with unrestricted $\beta \in \mathbb{F}$) to each row (here one can use the help of the client because the nonzero value is known to the client).
2. Check that the sum of every two consecutive rows has Hamming weight at most 1. This is done by applying the previous procedure to the sum of every two consecutive rows. The

above two conditions are not sufficient, since a malicious client can use all-zero rows to violate soundness. This possibility is eliminated by the following condition.

3. Check that no row is an all-zero row. The latter can be done (for each row separately) by having the servers reveal the sum of all entries, and further multiplied by a random field element picked by the servers, multiplied by a secret random field element shared by the client (where the latter product is computed using the client-assisted MPC protocol from Section 4.4.1). The first multiplication is needed to ensure that a malicious client cannot change a nonzero value to 0 (since the protocol from Section 4.4.1 allows a malicious client to add a fixed value to the output). The second multiplication is needed to prevent the servers from learning the nonzero value.

FSS for intervals. Denote by $f_{[a,b]}$ the interval function that evaluates 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. We handle general intervals via a reduction to the class of special intervals of the form $f_{[0,b]}$. In this case, a valid $\mathbf{y}$ is of the form $(1, 1, \ldots, 1, 0, 0, \ldots, 0)$. A sketching for special intervals can in turn be reduced to a sketching for the family of point functions $f_{\alpha,\beta}$ with $\beta \in \{0,1\}$ in the following way. Let $(\mathcal{L}, \mathcal{V})$ be any sketching scheme for this family (e.g., $(\mathcal{L}_{\mathsf{sq}}, \mathcal{V}_{\mathsf{sq}})$ or $(\mathcal{L}_{\mathsf{prod}}, \mathcal{V}_{\mathsf{prod}})$ will do). The sketching scheme for special intervals is $(\mathcal{L}', \mathcal{V})$, where $\mathcal{L}'$ is defined as follows:

- Sample $L \leftarrow \mathcal{L}(\mathbb{F}, N)$;
- Return L' defined by $L'_1 = L_1$ and $L'_j = L_j - L_{j-1}$ for $2 \le j \le N$, where L_j is the jth column of L and L' is the matrix with columns L'_j.

Let B be the set of $N+1$ vectors of the form $(1, \ldots, 1, 0, \ldots, 0)$. Completeness follows by observing that $L'\mathbf{y} = L'(\mathbf{y} - \mathbf{y}')$ where $\mathbf{y}'$ is obtained from $\mathbf{y}$ by shifting it one entry to the left and adding 0 on the right. If $\mathbf{y} \in B$, then $\mathbf{y} - \mathbf{y}'$ is either a unit vector or the all-0 vector, as required.

For soundness we argue that for any $\mathbf{y} \notin B$, the probability that $\mathcal{V}(L'(\mathbf{y})) = 0$ is bounded by the soundness error of $(\mathcal{L}, \mathcal{V})$. This follows from the fact that every linear combination of the columns of L can be uniquely described as a linear combination of the columns of L'. Hence, every nontrivial linear combination of L' whose coefficient vector is *not* in B is a nontrivial linear combination of the columns of L whose coefficient vector is *not* a unit vector. The soundness bound of $(\mathcal{L}, \mathcal{V})$ applies to this case.

Finally, to share a general interval $f_{[a,b]}$, the client generates keys for $f_{[0,a-1]}$ and $f_{[0,b]}$. Once both keys are verified, the parties can use them to evaluate $f_{[a,b]} = f_{[0,b]} - f_{[0,a-1]}$. Since a malicious client can pick a, b such that $a > b$, this verification actually applies to an extended function family $\mathcal{F}$ that also includes *negative* intervals of the form $-f_{[a,b]}$. However, in the context of applications that involve reading, which serve as the main motivation for FSS for intervals, sharing a negative interval does not give the client any advantage.

When $D' = D$, negative intervals can be eliminated by requiring the client to share between the servers each bit in the binary representation of the interval length. Assuming that this representation is d-bit long and the field characteristic is bigger than $2^d + N$, the sum of the entries in a negative interval cannot coincide with a k-bit integer, even when summation is done in $\mathbb{F}$. Checking that the field elements shared by the client are indeed in $\{0,1\}$ and comparing the integer represented by these bits to the sum of the entries in $\mathbf{y}$ can be done efficiently (communicating only a constant number of field elements per party) using the previous sketching techniques.

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 [26] or an arithmetic variant of this protocol [23, 30]. 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, 28]), 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.

4.4.1 MPC 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. Alternatively, the correlated randomness can be securely generated 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, 16] for efficient solutions to the latter.) In [22] 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 [22] 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 $[c]_i = \Delta b[a]_i + \Delta a[b]_i + [c']_i - \Delta a \Delta b$.

To evaluate $\mathcal{V}$ on the shared sketch $[\mathbf{z}]$, we use the above procedure for evaluating each multiplication,[4] where additions are implemented non-interactively, and the output is reconstructed by simply exchanging shares of the output.

We now 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 client-supplied 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.[5]

4.4.2 MPC using linear PCPs

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. Here we put forward a different technique for a natural application scenario in which this is not the case. Concretely, the technique applies to the case where we need to verify that the value β of a point function $f_{\alpha,\beta}$, which is fully known to the client, has a special structure. We start by explaining the motivation.

Consider the case of using a DPF for *writing* a value $\beta \in \mathbb{F}$ into a secret location α in a secret-shared array. A natural approach is to share the point function $f = f_{\alpha,\beta}$ and update each entry by adding the corresponding value of f. If the array is initially empty, i.e., it contains 0 in all of its entries, the value β can indeed be reconstructed from the shares of entry α. But what if different values β_j are written to the same entry α? In this case, the different values mix and there is generally no way to recover them. Such a situation arises, for example, when applying a DPF in the context of anonymous communication [15], where messages β_j that originate from different clients are mixed by writing them into random entries of an array which is shared between the parties. Here we expect collisions to occur almost certainly when the number of entries is less than quadratic in the number of messages, but the expected number of collisions will not be large.

An effective way of handling a bounded number of collisions is by encoding each message β_j into a bigger message $\hat{\beta}_j$, such that given the sum of the encodings of up to d different messages, all messages can be recovered. Such an encoding can be implemented efficiently via syndrome

[4] In fact, for the case of $\mathcal{V}_{\mathsf{sq}}$ we can use a leaner variant that computes $[a^2]$ from $[a]$.

[5] The only exception is the sketching scheme used for verifying that $\beta = 1$, where verification involves two polynomials of which one has degree 1. In this case, the parties can compute the degree-1 polynomial directly, without the help of the client, by using a simple $(m-1)$-secure protocol for addition.

decoding of linear error-correcting codes such as Reed-Solomon codes [21]. Perhaps the simplest such encoding maps a message $\beta \in \mathbb{F}$ into a vector $\hat{\beta} = (\beta, \beta^2, \ldots, \beta^d)$ [7, 11, 15]. By using such an encoding, data loss is prevented as long as there is no bin containing more than d messages.

So far we only considered the case of an honest client. Suppose a malicious client tries to destroy the data in an array by using invalid DPF keys. The above verification techniques can be used to ensure that a single "write" query can only change the value of a single entry of the array, thus limiting the corruption power of the client. Concretely, we use the technique for verifying a DPF for an arbitrary β, where we view β as an element of the extension field $\mathbb{F}^d$ (alternatively, one can use the more efficient procedure for verifying DPF $f_{\alpha,\beta}$ with unrestricted $\beta \in \mathbb{F}^d$ describe above). However, one can hope to do much better: if we could ensure that $\hat{\beta}$ is indeed of the expected special form $(\beta, \beta^2, \ldots, \beta^d)$, for some $\beta \in \mathbb{F}$, then the corruption power of such an adversary can be decreased by up to a factor of d: to prevent the correct reconstruction of an entry containing $d' < d$ messages of honest users, a malicious client will need more than $d - d'$ queries.

To verify a point function $f_{\alpha,\hat{\beta}}$ with the above special form, the m parties first check that it is indeed a point function for *some* $\hat{\beta}$, and then have the parties locally add their N entries of $\mathbf{y}^i$. In the end of this step, the parties hold additive shares of $d = 2\ell + 1$ field elements $\gamma = (\gamma_1, \ldots, \gamma_d)$, and they need to check that $\gamma_j = \gamma_1^j$ for $j = 2, \ldots, d$. The previous approach of using a simple client-assisted MPC protocol does not apply here, since the knowledge of the values γ_i allows a malicious client to violate the soundness by using an additive attack that directly changes the output.

Instead, we use the following client-assisted protocol, where the client helps the parties verify the validity of γ 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 [29, 6], exploiting the simple algebraic structure of the statement for better efficiency.

- INPUT: Parties hold additive shares $[\gamma_1], [\gamma_2], \ldots, [\gamma_d]$ of an input $\gamma = (\gamma_1, \ldots, \gamma_d) \in \mathbb{F}^d$ known to the client.
- OUTPUT: Parties should output "Accept" if $\gamma_j = \gamma_1^j$ for all $2 \leq j \leq d$, otherwise they should output "Reject" with overwhelming probability.
- PROOF: Client computes $\pi = (\gamma_1^{d+1}, \ldots, \gamma_1^{2d})$ and additively shares π between the parties. Let $\pi' = (\gamma, \pi) \in \mathbb{F}^{2d}$.
  // Shares of π are included in the keys output by Gen.
- VERIFICATION: Parties use common randomness to pick random field elements r_j, s_k, $1 \leq j, k \leq D$, and locally compute additive shares of the following linear combinations of π':
  - $z_1 = \sum_{j=1}^{d} r_j \pi'_j$
  - $z_2 = \sum_{k=1}^{d} s_k \pi'_k$
  - $z_3 = \sum_{1 \leq j,k \leq d} r_j s_k \pi'_{j+k}$
    // z_3 can be computed in $\tilde{O}(d)$ time using fast polynomial multiplication.
- DECISION: Parties use client-assisted MPC to check that $z_3 - z_1 z_2 = 0$ and accept or reject depending on whether this predicate holds.

Completeness and secrecy are straightforward. Soundness can again be analyzed via the SZ Lemma. Consider the following two cases:

1. For all $1 \leq j, k \leq d$, we have $\pi'_{j+k} = \pi'_j \pi'_k$: Since $\pi'_j = \gamma_j$ for $1 \leq j \leq d$, it follows that $\gamma_{j+1} = \gamma_j \cdot \gamma_1$ for $1 \leq j \leq d-1$, hence the input γ is well-formed.
2. For some $1 \leq j, k \leq d$ we have $\pi'_{j+k} \neq \pi'_j \pi'_k$: In this case, the monomial $r_j s_k$ will have a nonzero coefficient in the verification predicate, leading the parties to reject except with

at most $2/|\mathbb{F}|$ probability. As before, this holds even if the client-assisted computation of $z_3 - z_1 z_2$ uses badly formed randomness.

Hence, the soundness error of the protocol is bounded by $2/|\mathbb{F}|$.

4.5 Putting the Pieces Together

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

Theorem 4.8 (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 $\mathcal{F}_i$ with the following properties: (1) keys generated by Gen' with output field $\mathbb{F}$ include keys of Gen and a constant number of additional field elements, and (2) the parties in Ver 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 soundness error of the protocol is $O(1/|\mathbb{F}|)$.*

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 $D' = D$, where the soundness error in the latter case is $O(|D'|/|\mathbb{F}|)$);*
- *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.*

Acknowledgments. We thank Henry Corrigan-Gibbs for helpful clarifications about the results of [15]. 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.

The first author was additionally supported by ISF grant 1709/14 and ERC starting grant 307952. The second author was 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). The third author was additionally supported by DARPA Brandeis program under Contract N66001-15-C-4065, 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.

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] J. N. Bos and B. den Boer. Detection of disrupters in the DC protocol. In *EUROCRYPT '89*, pages 320–327, 1989.

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

[9] E. Boyle, N. Gilboa, and Y. Ishai. Breaking the circuit size barrier for secure computation under DDH. In *CRYPTO 2016, Part I*, pages 509–539, 2016. Full version: IACR Cryptology ePrint Archive 2016: 585 (2016).

[10] E. Boyle, N. Gilboa, and Y. Ishai. Function secret sharing: Improvements and extensions. In *ACM CCS 2016*, pages 1292–1303, 2016.

[11] R. T. Chien and W. D. Frazer. An application of coding theory to document retrieval. *IEEE Trans. Information Theory*, 12(2):92–96, 1966.

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

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

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

[15] 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.

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

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

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

[19] Z. Dvir and S. Gopi. 2-server PIR with sub-polynomial communication. In *Proceedings of 47th Annual ACM on Symposium on Theory of Computing, STOC*, pages 577–584, 2015.

[20] K. Efremenko. 3-query locally decodable codes of subexponential length. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing*, pages 39–44, 2009.

[21] 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.

[22] 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.

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

[24] N. Gilboa and Y. Ishai. Distributed point functions and their applications. In *EUROCRYPT 2014*, pages 640–658, 2014.

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

[26] 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.

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

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

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

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

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

[32] S. M. Matyas. Generating strong one-way functions with cryptographic algorithm. *IBM Technical Disclosure Bulletin*, 27:5658–5659, 1985.

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

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

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

[36] J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, 1980.

[37] 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.

[38] F. Wang, C. Yun, S. Goldwasser, V. Vaikuntanathan, and M. Zaharia. Splinter: Practical private queries on public data. *IACR Cryptology ePrint Archive*, 2016:1148, 2016. To appear in *NSDI 2017*.

[39] S. Yekhanin. Towards 3-query locally decodable codes of subexponential length. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing*, pages 266–274, 2007.

[40] R. Zippel. Probabilistic algorithms for sparse polynomials. In *Symbolic and Algebraic Computation, EUROSAM '79*, pages 216–226, 1979.

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 $\mathbb{Z}_2^m$ and additions are performed 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 $\mathbb{G} = \mathbb{Z}_{N+1}$. The client uses an FSS scheme for $\mathcal{F}$ to split 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 [35, 21].

Private updates. Consider the following application scenario for a writing analogue of PIR [34]. A client 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 client 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 $\beta = x_\alpha \oplus x'_\alpha$ 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

In this section we give a brief overview of different approaches for Private Information Retrieval (PIR) and compare their efficiency to the PIR protocols implied by this work. The first approach, presented in the original PIR work of Chor, Goldreich, Kushilevitz, and Sudan [14], 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 [12], still assumes $k \ge 2$ non-colluding servers but relaxes the security requirement to hold against computationally bounded servers. The third, introduced by Kushilevitz and Ostrovsky [31], assumes that the database is held by a single server and security is again computational.

The theoretical study of PIR is mainly concerned with minimizing the communication cost. However, in practice the performance of PIR can be dominated by either communication or computation, depending on the size of the database, the type of computations that need to be performed on the server side, and the execution environment.

In the application of DPF to PIR, as described in the Introduction, the query sent to each server consists of the DPF key. Using our optimized DPF construction with the early termination optimization, the key size is roughly $\lambda \cdot (n - \log \lambda)$ for a database of size $N = 2^n$ records, where $\lambda = 128$ for an AES-based implementation. This improves over the query size of the previous best DPF-based construction from [8] by roughly a factor of 4. The size of the answer sent by each server is equal to the record size. Using the optimized algorithm for full domain evaluation (Theorem 3.12), expanding a DPF key into a binary vector of length N requires roughly $N/64$ 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 reading and 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 (for 1-bit records); the computational cost of other types of PIR protocols is even worse.[6]

We now briefly compare the efficiency of our DPF-based protocol to alternative protocols. A simple 2-server protocol from [14] requires the client to send N bits to every server and receive

[6]To simplify the exposition, when considering multi-server PIR 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. The recent protocol of Dvir and Gopi [19] achieves the best known total asymptotic communication complexity for 2-server information-theoretic PIR with unrestricted answer size.

ℓ bits in return, where ℓ is the record size.[7] This protocol provides good concrete efficiency in applications that involve a small number of records (e.g., see [27]). The optimized DPF protocol in this paper leads to better communication complexity for databases that have 400 records or more, and comparable computational complexity on the server side.

With $k \geq 3$ servers, there are information-theoretic PIR protocol with low asymptotic communication complexity and 1-bit answers [39, 20]. However, employing an additional server may be costly. Moreover, even the best known protocols that are optimized for practical database sizes (e.g., the 3-server protocol from [3] with communication complexity $14N^{1/4}$) have higher communication than our protocols 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 vector of N ring elements and then compute the inner product of the database with the expanded key. In the two-server solution of [14] the expanded vector is explicitly sent to each server; other information-theoretic PIR protocols require several ring operations per database bit to perform this expansion.

Single-server PIR protocols with sublinear communication are known to imply public-key encryption [17]. Moreover, all known single-server PIR protocols require the server to perform a large number of "public key" operations (typically, an additive homomorphic operation on ciphertexts, for some additively homomorphic encryption, for each bit of the database). This is typically several orders of magnitude slower than multi-server protocols such as ours. See [33] and references therein for the state of the art on practical single-server PIR.

We finally note that all of the alternative approaches to PIR discussed above 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. In contrast, the FSS-based approach directly applies to these more general types of searches.

C Proof of Tensor Product Operation

In this section we provide the full proof of Theorem 3.2 of the security of the construction $(\mathsf{Gen}^{\otimes}, \mathsf{Eval}^{\otimes})$.

We begin by presenting the formal construction.

Note that keys output by $(\mathsf{Gen}^{\otimes}, \mathsf{Eval}^{\otimes})$ have the form of one key from $(\mathsf{Gen}^{\bullet}, \mathsf{Eval}^{\bullet})$ and two elements in the key space $\mathcal{K}$ of $(\mathsf{Gen}^{\mathcal{F}}, \mathsf{Eval}^{\mathcal{F}})$: that is, the resulting key size $\mathsf{size}_{\otimes}(n_1 + n_2, \lambda)$ is indeed $\mathsf{size}_{\bullet}(n_1, \lambda) + 2\mathsf{size}_{\mathcal{F}}(n_2, \lambda)$.

C.1 Proof of Theorem 3.2

Proof. We address the correctness and then security of the proposed construction $(\mathsf{Gen}^{\otimes}, \mathsf{Eval}^{\otimes})$.

Claim C.1 (Correctness)**.** *The resulting scheme* $(\mathsf{Gen}^{\otimes}, \mathsf{Eval}^{\otimes})$ *is correct: i.e., for all* $g_{\alpha,f} \in \mathcal{F}^{\bullet} \otimes \mathcal{F}$ *and every input* $(x_1, x_2) \in \{0,1\}^{n_1} \times \{0,1\}^{n_2}$*,*

$$Pr[(k_0, k_1) \leftarrow \mathsf{Gen}^{\otimes}(1^{\lambda}, g_{\alpha,f}) : \mathsf{Eval}^{\otimes}(k_0, 0, (x_1, x_2)) - \mathsf{Eval}^{\otimes}(k_1, 1, (x_1, x_2)) = g_{\alpha,f}(x_1, x_2)] = 1.$$

Proof. Recall the keys are of the form $k_b = \sigma_b || CW_0 || CW_1$, where $(\sigma_0, \sigma_1) \leftarrow \mathsf{Gen}^{\bullet}(1^{\lambda}, f_{\alpha, s_\alpha || 1})$ for random value $s_\alpha \in \{0,1\}^{\lambda}$, and the correction words satisfy $CW_{t^b_\alpha} = r_b + PRG(s^b_\alpha)$, where

[7] This protocol can be viewed as utilizing a naïve DPF scheme that secret-shares the entire truth-table. Different flavors of DPF are implicitly used in other PIR protocols from the literature. In particular, the first non-trivial 2-party DPF is implicit in the 2-server computational PIR scheme of Chor and Gilboa [12]. However, the key size of this DPF is super-polynomial in n. Similarly, a k-party DPF for $k \geq 3$ implicitly serves as the basis of most information-theoretic PIR protocols, originating from [14]; however, in these DPF constructions security only holds against $t < k - 1$ parties (typically $t = 1$) and the key size is super-polynomial in n when k is constant.

Tensor Product FSS $(\mathsf{Gen}^{\otimes}, \mathsf{Eval}^{\otimes})$

Let $PRG : \{0,1\}^{\lambda} \to \mathcal{K}$ be a pseudorandom generator (where $\mathcal{K}$ is the keyspace of $(\mathsf{Gen}^{\mathcal{F}}, \mathsf{Eval}^{\mathcal{F}})$. 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 $\mathbb{G}$.

$\mathsf{Gen}^{\otimes}(1^{\lambda}, g_{\alpha,f})$:

1: Sample a random λ-bit string $s_\alpha \leftarrow \{0,1\}^{\lambda}$, and let $t_\alpha = 1$ be a bit.
2: Generate an FSS key pair for the multi-bit point function $f_{\alpha,s_\alpha||t_\alpha}$ that outputs $(s_\alpha||t_\alpha) \in \{0,1\}^{\lambda+1}$ on input α and 0 otherwise. That is, $(\sigma_0, \sigma_1) \leftarrow \mathsf{Gen}^{\bullet}(1^{\lambda}, f_{\alpha,s_\alpha||t_\alpha})$.
3: Identify the resulting shares of the two parties for the evaluation input α: Let $s^0_\alpha||t^0_\alpha \leftarrow \mathsf{Eval}^{\bullet}(\sigma_0, 0, \alpha)$ and $s^1_\alpha||t^1_\alpha \leftarrow \mathsf{Eval}^{\bullet}(\sigma_1, 1, \alpha)$. Recall that $s^0_\alpha||t^0_\alpha$ and $s^1_\alpha||t^1_\alpha$ are an additive secret sharing of $s_\alpha||t_\alpha$ in $\{0,1\}^{\lambda+1}$.
4: Generate an FSS key pair for the function $f \in \mathcal{F}$: Let $(r_0, r_1) \leftarrow \mathsf{Gen}^{\mathcal{F}}(1^{\lambda}, f)$.
5: Mask keys r_0, r_1 with randomness generated from s^0_α, s^1_α, and set as correction words in the order dictated by t^0_α, t^1_α: ie, let $CW_{t^b_\alpha} \leftarrow PRG(s^b_\alpha) + r_b$, for $b = 0, 1$, with addition in $\mathcal{K}$.
6: Output final keys $k_b \leftarrow \sigma_b||CW_0||CW_1$, encoded as $\mathsf{size}_{\bullet}(n_1, \lambda) + 2\mathsf{size}_{\mathcal{F}}(n_2, \lambda)$ bits, for $b = 0, 1$.

$\mathsf{Eval}^{\otimes}(k_b, b, (x_1, x_2))$:

1: Parse k as $k = \sigma||CW_0||CW_1$.
2: Let $s_{x_1}||t_{x_1} \leftarrow \mathsf{Eval}^{\bullet}(\sigma, b, x_1)$.
   // Note that if $x_1 \neq \alpha$, the value $s_{x_1}||t_{x_1}$ is *equal* for both parties.
   // If $x_1 = \alpha$, then it is exactly the special string $s^b_\alpha||t^b_\alpha$ as in Step 3 of $\mathsf{Gen}^{\otimes}$.
3: Let $\tau \leftarrow CW_{t_{x_1}} - PRG(s_{x_1})$.
   // Similarly, here for $x_1 \neq \alpha$, the value τ is *equal* for both parties, since both s_{x_1} and t_{x_1} are the same across parties.
   // If $x_1 = \alpha$, then by the choice of CW_0, CW_1 (Step 5 of $\mathsf{Gen}^{\otimes}$), party 0 will compute $\tau = r_0$ and party 1 will compute $\tau = r_1$.
4: Compute $y \leftarrow \mathsf{Eval}^{\mathcal{F}}(\tau, b, x_2)$.
5: Output y.

Figure 8: FSS $(\mathsf{Gen}^{\otimes}, \mathsf{Eval}^{\otimes})$ for function class $\mathcal{F}^{\bullet} \otimes \mathcal{F}$, given schemes $(\mathsf{Gen}^{\bullet}, \mathsf{Eval}^{\bullet})$, $(\mathsf{Gen}^{\mathcal{F}}, \mathsf{Eval}^{\mathcal{F}})$.

$s^b_\alpha||t^b_\alpha$ is party b's share of the output $s_\alpha||1$ on input α (i.e., $s^b_\alpha||t^b_\alpha \leftarrow \mathsf{Eval}^\bullet(\sigma_b, b, \alpha)$), and (r_0, r_1) is an FSS key pair for the function f, generated as $(r_0, r_1) \leftarrow \mathsf{Gen}^{\mathcal{F}}(1^\lambda, f)$. We address the correctness of the scheme in two cases.

Case 1: $x_1 \neq \alpha$. By the correctness of the FSS scheme $(\mathsf{Gen}^\bullet, \mathsf{Eval}^\bullet)$, for $x_1 \neq \alpha$, we have that $\mathsf{Eval}^\bullet(\sigma_0, 0, x_1) - \mathsf{Eval}^\bullet(\sigma_1, 1, x_1) = 0$: that is, that the evaluation of $s_{x_1}||t_{x_1}$ in Step 2 in $\mathsf{Eval}^\otimes$ will be *equal* for both party 0 and party 1. This means that both parties will arrive at the same value τ in Step 3, and thus both parties will produce the same output y in the final step, producing $\mathsf{Eval}^\otimes(k_0, 0, (x_1, x_2)) - \mathsf{Eval}^\otimes(k_1, 1, (x_1, x_2)) = 0$, as desired. Note that this case does not rely on any correctness properties of the second FSS scheme $(\mathsf{Gen}^{\mathcal{F}}, \mathsf{Eval}^{\mathcal{F}})$: in fact, the parties will be evaluating with respect to "garbage" keys τ, but whatever resulting evaluations will be consistent between parties.

Case 2: $x_1 = \alpha$. By definition, each party $b \in \{0, 1\}$ will compute $s^b_\alpha||t^b_\alpha$ from the evaluation $\mathsf{Eval}^\bullet(\sigma_b, b, \alpha)$ in Step 2 of $\mathsf{Eval}^\otimes$, which implies (from the construction of the correction words $CW_{t^b_\alpha} = r_b + PRG(s^b_\alpha)$ in Step 5 of $\mathsf{Gen}^\otimes$) that he will exactly recover the key r_b as his result τ in Step 3. Therefore, we have that $\mathsf{Eval}^\otimes(k_0, 0, (x_1, x_2)) - \mathsf{Eval}^\otimes(k_1, 1, (x_1, x_2)) = \mathsf{Eval}^{\mathcal{F}}(r_0, 0, x_2) - \mathsf{Eval}^{\mathcal{F}}(r_1, 1, x_2)$, which by the correctness of the FSS scheme $(\mathsf{Gen}^{\mathcal{F}}, \mathsf{Eval}^{\mathcal{F}})$, is precisely $f(x_2) = g_{\alpha,f}(x_1, x_2)$.

□

Claim C.2 (Security)**.** *There exists a polynomial $p(n) \in \mathsf{poly}(n)$ such that, given the tools listed above, the scheme $(\mathsf{Gen}^\otimes, \mathsf{Eval}^\otimes)$ is a (t', ϵ')-secure FSS scheme for $t' = t - p(n_1 + n_2)$ and $\epsilon' = \epsilon_{\mathsf{DPF}} + 2 \cdot \epsilon_{\mathsf{PRG}} + 2 \cdot \epsilon_{\mathsf{FSS}}$.*

Proof. Recall that for $b \in \{0, 1\}$, by construction, $k_b = \sigma_b||CW_0||CW_1$ where $CW_{t^b_\alpha} = G(s^b_\alpha) + r_b$ and $CW_{1-b} = G(s^{1-b}_\alpha) + r_{1-b}$. At a high level, we will show that the correction words CW_0, CW_1 are *pseudorandom* given the remaining view of each party because of this masking, and thus together with the security for the underlying scheme $(\mathsf{Gen}^\bullet, \mathsf{Eval}^\bullet)$, we will have indistinguishability of keys for our new scheme $(\mathsf{Gen}^\otimes, \mathsf{Eval}^\otimes)$.

Formally, for any pair of functions $g_{\alpha,f}, g_{\hat{\alpha},\hat{f}} \in \mathcal{F}^\bullet \otimes \mathcal{F}$, we consider a sequence of hybrid distributions that begins with an honestly generated FSS key for $g_{\alpha,f}$ and ends with an honestly generated key for $g_{\hat{\alpha},\hat{f}}$. We will show that an adversary who wins in the FSS security game for $(\mathsf{Gen}^\otimes, \mathsf{Eval}^\otimes)$ with advantage greater than ϵ' must succeed in distinguishing between these distributions for some $(g_{\alpha,f}, g_{\hat{\alpha},\hat{f}})$, and thus distinguishes between some adjacent hybrids with advantage that contradicts the security of one of the underlying tools.

In what follows, we define the hybrid experiments. Within the distribution descriptions, we denote with boxes the lines which were modified from the previous step.

Hybrid 0: Real key distribution for function $g_{\alpha,f}$.

$$H_0(b, \alpha, f) := \left\{ \sigma_b||CW_0||CW_1 \ : \begin{array}{c} s_\alpha \leftarrow \{0,1\}^\lambda, \\ (\sigma_0, \sigma_1) \leftarrow \mathsf{Gen}^\bullet(1^\lambda, f_{\alpha, s_\alpha||1}), \\ (r_0, r_1) \leftarrow \mathsf{Gen}^{\mathcal{F}}(1^\lambda, f), \\ s^i_\alpha||t^i_\alpha = \mathsf{Eval}^\bullet(\sigma_i, \alpha)\ \forall i \in \{0,1\}, \\ CW_{t^b_\alpha} = r_b + PRG(s^b_\alpha), \\ CW_{1-t^b_\alpha} = r_{1-b} + PRG(s^{1-b}_\alpha) \end{array} \right\}$$

By correctness of the FSS, $\mathsf{Eval}^\bullet(\sigma_{1-b}, \alpha) = f_{\alpha, s_\alpha||1}(\alpha) - \mathsf{Eval}^\bullet(\sigma_b, \alpha) = s_\alpha||1 - s^b_\alpha||t^b_\alpha$. So,

$t_\alpha^{1-b} = 1 - t_\alpha^b$ and $s_\alpha^{1-b} = s_\alpha - s_\alpha^b$:

$$H_0(b,\alpha,f) \equiv \left\{ \sigma_b||CW_0||CW_1 \ : \begin{array}{c} s_\alpha \leftarrow \{0,1\}^\lambda, \\ (\sigma_0,\sigma_1) \leftarrow \mathsf{Gen}^\bullet(1^\lambda, f_{\alpha,s_\alpha||1}), \\ (r_0,r_1) \leftarrow \mathsf{Gen}^{\mathcal{F}}(1^\lambda, f), \\ \boxed{s_\alpha^b||t_\alpha^b = \mathsf{Eval}^\bullet(\sigma_b,\alpha),} \\ CW_{t_\alpha^b} = r_b + PRG(s_\alpha^b), \\ \boxed{CW_{1-t_\alpha^b} = r_{1-b} + PRG(s_\alpha - s_\alpha^b)} \end{array} \right\}$$

Hybrid 1: (DPF Security). Replace DPF key for $f_{\alpha,s_\alpha||1}$ with key for $f_{\hat{\alpha},s_{\hat{\alpha}}||1}$. Note all remaining steps are still performed with respect to the original α, s_α.

$$H_1(b,\alpha,\hat{\alpha},f) := \left\{ \sigma_b||CW_0||CW_1 \ : \begin{array}{c} \boxed{s_\alpha, s_{\hat{\alpha}} \leftarrow \{0,1\}^\lambda,} \\ \boxed{(\sigma_0,\sigma_1) \leftarrow \mathsf{Gen}^\bullet(1^\lambda, f_{\hat{\alpha},s_{\hat{\alpha}}||1}),} \\ (r_0,r_1) \leftarrow \mathsf{Gen}^{\mathcal{F}}(1^\lambda, f), \\ s_\alpha^b||t_\alpha^b = \mathsf{Eval}^\bullet(\sigma_b,\alpha), \\ CW_{t_\alpha^b} = r_b + PRG(s_\alpha^b), \\ CW_{1-t_\alpha^b} = r_{1-b} + PRG(s_\alpha - s_\alpha^b) \end{array} \right\}$$

Claim C.3. *There exists a polynomial p_1 such that for every $b \in \{0,1\}, \alpha, \hat{\alpha} \in \{0,1\}^{n_1}, f \in \mathcal{F}$, and auxiliary input z, no adversary running in time $T - p_1(n_1+n_2)$ can distinguish the distributions $(H_0(b,\alpha,f), z)$ and $(H_1(b,\alpha,\hat{\alpha},f), z)$ with advantage greater than ϵ_{DPF}.*

Proof. Suppose there exists $b^* \in \{0,1\}, \alpha^*, \hat{\alpha}^* \in \mathbb{G}_{N_1}, f^* \in \mathcal{F}$, auxiliary input z^*, and an adversary $\mathcal{A}^*$ that runs in time T'' for which

$$|\Pr[k_{b^*} \leftarrow H_0(b^*,\alpha^*,f^*) : \mathcal{A}^*(k_{b^*},z^*) = 1] - \Pr[k_{b^*} \leftarrow H_1(b^*,\alpha^*,\hat{\alpha}^*,f^*) : \mathcal{A}^*(k_{b^*},z^*) = 1]| > \epsilon_{\mathsf{DPF}}.$$

We will use this adversary $\mathcal{A}^*$ to construct an adversary $\mathcal{B}$ for the underlying FSS scheme $(\mathsf{Gen}^\bullet, \mathsf{Eval}^\bullet)$. Define auxiliary input $z_{\mathcal{B}} := \{b^*, \alpha^*, \hat{\alpha}^*, f^*, z^*\}$.

Adversary $\mathcal{B}(1^\lambda, z_{\mathcal{B}})$:

1. Parse $z_{\mathcal{B}} = \{b^*, \alpha^*, \hat{\alpha}^*, f^*, z^*\}$.
2. Choose b^* as the corrupted party and $f_{\alpha^*,s_{\alpha^*}||1}, f_{\hat{\alpha}^*,s_{\hat{\alpha}^*}||1} \in \mathcal{F}^\bullet$ as the pair of DPF challenge functions.
3. Receive as input from the DPF challenger a key σ_{b^*}, sampled as $(\sigma_0,\sigma_1) \leftarrow \mathsf{Gen}^\bullet(1^\lambda, h)$ for either $h = f_{\alpha^*,s_\alpha^*||1}$ or $h = f_{\hat{\alpha}^*,s_{\hat{\alpha}^*}||1}$.
4. Sample $s_\alpha, s_{\hat{\alpha}} \leftarrow \{0,1\}^\lambda$.
5. Sample a key pair $(r_0,r_1) \leftarrow \mathsf{Gen}^{\mathcal{F}}(1^\lambda, f^*)$.
6. Compute $s_{\alpha^*}^{b^*}||t_{\alpha^*}^{b^*} = \mathsf{Eval}^\bullet(\sigma_{b^*},\alpha^*)$, using the challenge key σ_{b^*}.
7. Let $CW_{t_{\alpha^*}^{b^*}} = r_{b^*} + PRG(s_{\alpha^*}^{b^*})$.
8. Let $CW_{1-t_{\alpha^*}^{b^*}} = r_{1-b^*} + PRG(s_{\alpha^*} - s_{\alpha^*}^{b^*})$.
9. Take $k_{b^*} := \sigma_{b^*}||CW_0||CW_1$.
10. Let $\mathsf{guess} \leftarrow \mathcal{A}^*(k_{b^*}, z^*)$. Output guess.

The runtime of $\mathcal{B}$ is equal to $\mathsf{time}(\mathcal{A}^*) + \mathsf{time}(\mathsf{Gen}^{\mathcal{F}}[1^\lambda, n_2]) + \mathsf{time}(\mathsf{Eval}^\bullet[1^\lambda, n_1]) + 2 \cdot \mathsf{time}(PRG) = \mathsf{time}(\mathcal{A}^*) + p_1(n_1+n_2)$ for some fixed polynomial p_1. By construction, if the challenge key σ_{b^*} is generated using $h = f_{\alpha^*,s_{\alpha^*}||1}$ then k_{b^*} is distributed precisely as $H_0(b^*,\alpha^*,f^*)$, whereas if it is generated using $h = f_{\hat{\alpha}^*,s_{\hat{\alpha}^*}||1}$ then k_{b^*} is distributed

precisely as $H_1(b^*, \alpha^*, \hat{\alpha}^*, f^*)$. Thus the advantage of $\mathcal{B}$ in the DPF security game is equal to the advantage of $\mathcal{A}^*$ in distinguishing distributions, which is $> \epsilon_{\mathsf{DPF}}$. If $\mathcal{A}^*$ runs in time $T'' \leq T - p_1(n)$, then $\mathcal{B}$ runs in time $\leq T$, which would contradict the $(T, \epsilon_{\mathsf{DPF}})$-security of the underlying DPF scheme.

□

Hybrid 2: (PRG Security). Replace $PRG(s_\alpha - s_\alpha^b)$ with random R.

$$H_2(b, \alpha, \hat{\alpha}, f) := \left\{ \sigma_b||CW_0||CW_1 \ : \begin{array}{c} \boxed{R \leftarrow \mathcal{K},} \\ s_{\hat{\alpha}} \leftarrow \{0,1\}^\lambda, \\ (\sigma_0, \sigma_1) \leftarrow \mathsf{Gen}^\bullet(1^\lambda, f_{\hat{\alpha}, s_{\hat{\alpha}}||1}), \\ (r_0, r_1) \leftarrow \mathsf{Gen}^{\mathcal{F}}(1^\lambda, f), \\ s_\alpha^b||t_\alpha^b = \mathsf{Eval}^\bullet(\sigma_b, \alpha), \\ CW_{t_\alpha^b} = r_b + PRG(s_\alpha^b), \\ \boxed{CW_{1-t_\alpha^b} = r_{1-b} + R} \end{array} \right\}$$

Claim C.4. *There exists a polynomial p_2 such that for every $b \in \{0,1\}, \alpha, \hat{\alpha} \in \{0,1\}^{n_1}, f \in \mathcal{F}$, and auxiliary input z, no adversary running in time $T - p_2(n_1 + n_2)$ can distinguish the distributions $(H_1(b, \alpha, \hat{\alpha}, f), z)$ and $(H_2(b, \alpha, \hat{\alpha}, f), z)$ with advantage greater than ϵ_{PRG}.*

Proof. Suppose there exists $b^* \in \{0,1\}, \alpha^*, \hat{\alpha}^* \in \mathbb{G}_{N_1}, f^* \in \mathcal{F}$, auxiliary input z^*, and an adversary $\mathcal{A}^*$ that runs in time T'' for which

$$|\Pr[k_{b^*} \leftarrow H_1(b^*, \alpha^*, \hat{\alpha}^*, f^*) : \mathcal{A}^*(k_{b^*}, z^*) = 1] - \Pr[k_{b^*} \leftarrow H_2(b^*, \alpha^*, \hat{\alpha}^*, f^*) : \mathcal{A}^*(k_{b^*}, z^*) = 1]| > \epsilon_{\mathsf{PRG}}.$$

We will use this adversary $\mathcal{A}^*$ to construct an adversary $\mathcal{B}$ for the underlying PRG. Define auxiliary input $z_{\mathcal{B}} := \{b^*, \alpha^*, \hat{\alpha}^*, f^*, z^*\}$. In the PRG challenge, $\mathcal{B}$ receives a string $K \in \mathcal{K}$ which is either sampled either randomly as $K \leftarrow \mathcal{K}$ or pseudorandomly as $K = PRG(s) : s \leftarrow \{0,1\}^\lambda$.

Adversary $\mathcal{B}(1^\lambda, K, z_{\mathcal{B}})$:

1. Parse $z_{\mathcal{B}} = \{b^*, \alpha^*, \hat{\alpha}^*, f^*, z^*\}$.
2. Sample $s_{\hat{\alpha}^*} \leftarrow \{0,1\}^\lambda$.
3. Sample DPF keys $(\sigma_0, \sigma_1) \leftarrow \mathsf{Gen}^\bullet(1^\lambda, f_{\hat{\alpha}^*, s_{\hat{\alpha}^*}||1})$.
4. Sample FSS keys $(r_0, r_1) \leftarrow \mathsf{Gen}^{\mathcal{F}}(1^\lambda, f^*)$.
5. Compute $s_{\alpha^*}^{b^*}||t_{\alpha^*}^{b^*} = \mathsf{Eval}^\bullet(\sigma_{b^*}, \alpha^*)$.
6. Take $CW_{t_{\alpha^*}^{b^*}} = r_{b^*} + PRG(s_{\alpha^*}^{b^*})$.
7. Take $CW_{1-t_{\alpha^*}^{b^*}} = r_{1-b^*} + K$.
8. Define $k_{b^*} = \sigma_{b^*}||CW_0||CW_1$.
9. Let $\mathsf{guess} \leftarrow \mathcal{A}^*(k_{b^*}, z^*)$. Output guess.

The runtime of $\mathcal{B}$ is equal to $\mathsf{time}(\mathcal{A}^*) + \mathsf{time}(\mathsf{Gen}^\bullet[1^\lambda, n_1]) + \mathsf{time}(\mathsf{Gen}^{\mathcal{F}}[1^\lambda, n_2]) + \mathsf{time}(\mathsf{Eval}^\bullet[1^\lambda, n_1]) + \mathsf{time}(PRG) = \mathsf{time}(\mathcal{A}^*) + p_2(n_1 + n_2)$ for some fixed polynomial p_2. By construction, if K is pseudorandom then k_{b^*} is distributed precisely as $H_1(b^*, \alpha^*, \hat{\alpha}^*, f^*)$, whereas if it is sampled randomly from $\mathcal{K}$ then k_{b^*} is distributed precisely as $H_2(b^*, \alpha^*, \hat{\alpha}^*, f^*)$. Thus the advantage of $\mathcal{B}$ in the DPF security game is equal to the advantage of $\mathcal{A}^*$ in distinguishing distributions, which is $> \epsilon_{\mathsf{PRG}}$. If $\mathcal{A}^*$ runs in time $T'' \leq T - p_2(n)$, then $\mathcal{B}$ runs in time $\leq T$, which would contradict the $(T, \epsilon_{\mathsf{PRG}})$-security of the underlying PRG scheme.

□

Hybrid 3: (Pseudorandomness of keys for $(\mathsf{Gen}^{\mathcal{F}}, \mathsf{Eval}^{\mathcal{F}})$). Replace r_b with random $R'' \in \mathcal{K}$.

$$H_3(b,\alpha,\hat{\alpha}) := \left\{ \sigma_b||CW_0||CW_1 \ : \ \begin{array}{c} \boxed{R', R'' \leftarrow \mathcal{K},} \\ s_{\hat{\alpha}} \leftarrow \{0,1\}^\lambda, \\ (\sigma_0,\sigma_1) \leftarrow \mathsf{Gen}^\bullet(1^\lambda, f_{\hat{\alpha},s_{\hat{\alpha}}||1}), \\ s^b_\alpha||t^b_\alpha = \mathsf{Eval}^\bullet(\sigma_b,\alpha), \\ \boxed{CW_{t^b_\alpha} = R'' + PRG(s^b_\alpha),} \\ CW_{1-t^b_\alpha} = R' \end{array} \right\}$$

Now that we've removed all information about the second key r_{1-b} generated via $\mathsf{Gen}^{\mathcal{F}}(1^\lambda, f)$ in the previous hybrids, we may here rely on the pseudorandomness of any *single* key generated by $\mathsf{Gen}^{\mathcal{F}}$.

Claim C.5. *There exists a polynomial p_3 such that for every $b \in \{0,1\}, \alpha, \hat{\alpha} \in \{0,1\}^{n_1}, f \in \mathcal{F}$, and auxiliary input z, no adversary running in time $T - p_3(n_1+n_2)$ can distinguish the distributions $(H_2(b,\alpha,\hat{\alpha},f),z)$ and $(H_3(b,\alpha,\hat{\alpha}),z)$ with advantage greater than ϵ_{FSS}.*

Proof. Suppose there exists $b^* \in \{0,1\}, \alpha^*, \hat{\alpha}^* \in \mathbb{G}_{N_1}, f^* \in \mathcal{F}$, auxiliary input z^*, and an adversary $\mathcal{A}^*$ that runs in time T'' for which

$$|\Pr[k_{b^*} \leftarrow H_2(b^*,\alpha^*,\hat{\alpha}^*,f^*) : \mathcal{A}^*(k_{b^*},z^*)=1] - \Pr[k_{b^*} \leftarrow H_3(b^*,\alpha^*,\hat{\alpha}^*) : \mathcal{A}^*(k_{b^*},z^*)=1]| > \epsilon_{\mathsf{FSS}}.$$

We will use this adversary $\mathcal{A}^*$ to construct an adversary $\mathcal{B}$ for the pseudorandomness of the keys of $(\mathsf{Gen}^{\mathcal{F}}, \mathsf{Eval}^{\mathcal{F}})$. Define auxiliary input $z_{\mathcal{B}} := \{b^*, \alpha^*, \hat{\alpha}^*, f^*, z^*\}$.

Adversary $\mathcal{B}(1^\lambda, z_{\mathcal{B}})$:

1. Parse $z_{\mathcal{B}} = \{b^*, \alpha^*, \hat{\alpha}^*, f^*, z^*\}$.
2. Select party b^* and function $f^* \in \mathcal{F}$ for the challenge.
3. Receive a key K_{b^*} that is sampled either as $(K_0,K_1) \leftarrow \mathsf{Gen}^{\mathcal{F}}(1^\lambda, f^*)$ or randomly sampled as $K_{b^*} \leftarrow \mathcal{K}$.
4. Sample $s_{\hat{\alpha}^*} \leftarrow \{0,1\}^\lambda$.
5. Generate $(\sigma_0,\sigma_1) \leftarrow \mathsf{Gen}^\bullet(1^\lambda, f_{\hat{\alpha}^*,s_{\hat{\alpha}^*}||1})$.
6. Evaluate $s^{b^*}_{\alpha^*}||t^{b^*}_{\alpha^*} = \mathsf{Eval}^\bullet(\sigma_{b^*},\alpha)$.
7. Take $CW_{t^{b^*}_{\alpha^*}} = K_{b^*} + PRG(s^{b^*}_{\alpha^*})$.
8. Sample $CW_{1-t^{b^*}_{\alpha^*}} \leftarrow \mathcal{K}$.
9. Define $k_{b^*} = \sigma_{b^*}||CW_0||CW_1$.
10. Let $\mathsf{guess} \leftarrow \mathcal{A}^*(k_{b^*}, z^*)$. Output guess.

The runtime of $\mathcal{B}$ is equal to $\mathsf{time}(\mathcal{A}^*)+\mathsf{time}(\mathsf{Gen}^\bullet[1^\lambda,n_1])+\mathsf{time}(\mathsf{Eval}^\bullet[1^\lambda,n_1])+\mathsf{time}(PRG) = \mathsf{time}(\mathcal{A}^*) + p_3(n_1+n_2)$ for some fixed polynomial p_3. By construction, if K_{b^*} is sampled from $\mathsf{Gen}^{\mathcal{F}}(1^\lambda, f^*)$ then k_{b^*} is distributed precisely as $H_2(b^*,\alpha^*,\hat{\alpha}^*,f^*)$, whereas if it is sampled randomly from $\mathcal{K}$ then k_{b^*} is distributed precisely as $H_3(b^*,\alpha^*,\hat{\alpha}^*)$. Thus the advantage of $\mathcal{B}$ in the DPF security game is equal to the advantage of $\mathcal{A}^*$ in distinguishing distributions, which is $> \epsilon_{\mathsf{FSS}}$. If $\mathcal{A}^*$ runs in time $T'' \leq T - p_3(n)$, then $\mathcal{B}$ runs in time $\leq T$, which would contradict the $(t,\epsilon_{\mathsf{FSS}})$-key-pseudorandomness of the underlying FSS scheme $(\mathsf{Gen}^{\mathcal{F}}, \mathsf{Eval}^{\mathcal{F}})$.

□

Note that this distribution no longer depends at all on the original α and can be rewritten as:

$$H_3(b,\hat{\alpha}) \equiv \left\{ \sigma_b||CW_0||CW_1 \ : \ \begin{array}{c} R, R' \leftarrow \mathcal{K}, \\ s_{\hat{\alpha}} \leftarrow \{0,1\}^\lambda, \\ (\sigma_0,\sigma_1) \leftarrow \mathsf{Gen}^\bullet(1^\lambda, f_{\hat{\alpha},s_{\hat{\alpha}}||1}), \\ \boxed{CW_0 = R,} \\ \boxed{CW_1 = R'} \end{array} \right\}$$

And further, since the values of CW_0, CW_1 are random, we can add in fixed values depending on $\hat{\alpha}$ and it will not affect this distribution.

$$H_3(b,\hat{\alpha}) \equiv \left\{ \sigma_b||CW_0||CW_1 \ : \ \begin{array}{c} R, R' \leftarrow \mathcal{K}, \\ s_{\hat{\alpha}} \leftarrow \{0,1\}^\lambda, \\ (\sigma_0,\sigma_1) \leftarrow \mathsf{Gen}^\bullet(1^\lambda, f_{\hat{\alpha},s_{\hat{\alpha}}||1}), \\ \boxed{s^b_{\hat{\alpha}}||t^b_{\hat{\alpha}} = \mathsf{Eval}^\bullet(\sigma_b,\hat{\alpha}),} \\ \boxed{CW_{t^b_{\hat{\alpha}}} = R + PRG(s^b_{\hat{\alpha}}),} \\ CW_{1-t^b_{\hat{\alpha}}} = R' \end{array} \right\}$$

In the following hybrids we "undo" the last 2 steps, but using the *new* function $g_{\hat{\alpha},\hat{f}}$.

Hybrid 4: (Pseudorandomness of keys for $(\mathsf{Gen}^{\mathcal{F}}, \mathsf{Eval}^{\mathcal{F}})$). Replace random R with $r_b + PRG(s^b_{\hat{\alpha}})$, where r_b is a key generated by $\mathsf{Gen}^{\mathcal{F}}$ for the (new) function $\hat{f}$.

$$H_4(b,\hat{\alpha},\hat{f}) := \left\{ \sigma_b||CW_0||CW_1 \ : \ \begin{array}{c} R' \leftarrow \mathcal{K}, \\ s_{\hat{\alpha}} \leftarrow \{0,1\}^\lambda, \\ (\sigma_0,\sigma_1) \leftarrow \mathsf{Gen}^\bullet(1^\lambda, f_{\hat{\alpha},s_{\hat{\alpha}}||1}), \\ \boxed{(r_0,r_1) \leftarrow \mathsf{Gen}^{\mathcal{F}}(1^\lambda,\hat{f}),} \\ s^b_{\hat{\alpha}}||t^b_{\hat{\alpha}} = \mathsf{Eval}^\bullet(\sigma_b,\hat{\alpha}), \\ \boxed{CW_{t^b_{\hat{\alpha}}} = r_b + PRG(s^b_{\hat{\alpha}}),} \\ CW_{1-t^b_{\hat{\alpha}}} = R' \end{array} \right\}$$

Claim C.6. *There exists a polynomial p_4 such that for every $b \in \{0,1\}, \alpha, \hat{\alpha} \in \{0,1\}^{n_1}, f \in \mathcal{F}$, and auxiliary input z, no adversary running in time $T - p_4(n_1+n_2)$ can distinguish the distributions $(H_3(b,\hat{\alpha}), z)$ and $(H_4(b,\hat{\alpha},\hat{f}), z)$ with advantage greater than ϵ_{FSS}.*

Proof. The proof is nearly identical to that of Claim C.5. □

Hybrid 5: Replace random R' with $r_{1-b} + PRG(s_{\hat{\alpha}} - s^b_{\hat{\alpha}})$. (Security of PRG).

$$H_5(b,\hat{\alpha},\hat{f}) := \left\{ \sigma_b||CW_0||CW_1 \ : \ \begin{array}{c} R' \leftarrow \mathcal{K}, \\ s_{\hat{\alpha}} \leftarrow \{0,1\}^\lambda, \\ (\sigma_0,\sigma_1) \leftarrow \mathsf{Gen}^\bullet(1^\lambda, f_{\hat{\alpha},s_{\hat{\alpha}}||1}), \\ (r_0,r_1) \leftarrow \mathsf{Gen}^{\mathcal{F}}(1^\lambda,\hat{f}), \\ s^b_{\hat{\alpha}}||t^b_{\hat{\alpha}} = \mathsf{Eval}^\bullet(\sigma_b,\hat{\alpha}), \\ CW_{t^b_{\hat{\alpha}}} = r_b + PRG(s^b_{\hat{\alpha}}), \\ \boxed{CW_{1-t^b_{\hat{\alpha}}} = r_{1-b} + PRG(s_{\hat{\alpha}} - s^b_{\hat{\alpha}})} \end{array} \right\}$$

Claim C.7. *There exists a polynomial p_5 such that for every $b \in \{0,1\}, \alpha, \hat{\alpha} \in \{0,1\}^{n_1}, f \in \mathcal{F}$, and auxiliary input z, no adversary running in time $T - p_5(n_1+n_2)$ can distinguish the distributions $(H_4(b,\hat{\alpha},\hat{f}), z)$ and $(H_5(b,\hat{\alpha},\hat{f}), z)$ with advantage greater than ϵ_{PRG}.*

Proof. The proof is nearly identical to that of Claim C.4. □

By the correctness of $(\mathsf{Gen}^\bullet, \mathsf{Eval}^\bullet)$, we have that $s_{\hat{\alpha}} - s^b_{\hat{\alpha}} = s^{1-b}_{\hat{\alpha}}$, and so we have

$$H_5(b,\hat{\alpha},\hat{f}) \equiv \left\{ \sigma_b||CW_0||CW_1 \ : \ \begin{array}{c} R' \leftarrow \mathcal{K}, \\ s_{\hat{\alpha}} \leftarrow \{0,1\}^\lambda, \\ (\sigma_0,\sigma_1) \leftarrow \mathsf{Gen}^\bullet(1^\lambda, f_{\hat{\alpha},s_{\hat{\alpha}}||1}), \\ (r_0,r_1) \leftarrow \mathsf{Gen}^{\mathcal{F}}(1^\lambda,\hat{f}), \\ s^i_{\hat{\alpha}}||t^i_{\hat{\alpha}} = \mathsf{Eval}^\bullet(\sigma_i,\hat{\alpha}) \ \forall i \in \{0,1\}, \\ CW_{t^b_{\hat{\alpha}}} = r_b + PRG(s^b_{\hat{\alpha}}), \\ \boxed{CW_{1-t^b_{\hat{\alpha}}} = r_{1-b} + PRG(s^{1-b}_{\hat{\alpha}})} \end{array} \right\}$$

Note that this distribution $H_5(b,\hat{\alpha},\hat{f})$ is now precisely the distribution of honestly generated keys for the function $g_{\hat{\alpha},\hat{f}}$: That is, $H_5(b,\hat{\alpha},\hat{f}) \equiv H_0(b,\hat{\alpha},\hat{f})$.

We now combine Claims C.3-C.7 to finalize the security proof.

Claim C.8. *There exists a polynomial $p(n) \in \mathsf{poly}(n)$ such that the scheme $(\mathsf{Gen}^\otimes, \mathsf{Eval}^\otimes)$ is a (T',ϵ')-secure FSS scheme for $T' = T - p(n_1+n_2)$ and $\epsilon' = \epsilon_{\mathsf{DPF}} + 2\epsilon_{\mathsf{PRG}} + 2\epsilon_{\mathsf{FSS}}$.*

Proof. Suppose there exists $b^* \in \{0,1\}$ and an adversary $\mathcal{A}^*$ running in some time T'' who succeeds in the FSS security game for $(\mathsf{Gen}^\otimes, \mathsf{Eval}^\otimes)$ for corrupted party b^* with advantage greater than ϵ' (see Appendix D). That is,

$$\left| \Pr\left[\begin{array}{l} (g_0,g_1,\mathsf{state}) \leftarrow \mathcal{A}(1^\lambda) \\ (k_0,k_1) \leftarrow \mathsf{Gen}(1^\lambda,g_1) \\ \mathsf{guess} \leftarrow \mathcal{A}(k_{b^*},\mathsf{state}) \end{array} : \mathsf{guess} = 1 \right] - \Pr\left[\begin{array}{l} (g_0,g_1,\mathsf{state}) \leftarrow \mathcal{A}(1^\lambda) \\ (k_0,k_1) \leftarrow \mathsf{Gen}(1^\lambda,g_0) \\ \mathsf{guess} \leftarrow \mathcal{A}(k_{b^*},\mathsf{state}) \end{array} : \mathsf{guess} = 1 \right] \right| > \epsilon'.$$

In particular, there must exist a pair of functions $g_0 = g_{\alpha_0,f_0}$ and $g_1 = g_{\alpha_1,f_1}$ and value of state for which

$$\left| \Pr\left[\begin{array}{l} (k_0,k_1) \leftarrow \mathsf{Gen}(1^\lambda,g_1) \\ \mathsf{guess} \leftarrow \mathcal{A}(k_{b^*},\mathsf{state}) \end{array} : \mathsf{guess} = 1 \right] - \Pr\left[\begin{array}{l} (k_0,k_1) \leftarrow \mathsf{Gen}(1^\lambda,g_0) \\ \mathsf{guess} \leftarrow \mathcal{A}(k_{b^*},\mathsf{state}) \end{array} : \mathsf{guess} = 1 \right] \right| > \epsilon'.$$

Note that the distribution of k_{b^*} received by $\mathcal{A}^*$ corresponds exactly to the distribution of $k_{b^*} \leftarrow H_0(b^*,\alpha_c,f_c)$ for the corresponding function g_{α_c,f_c} (indeed, H_0 was defined to be the honest key distribution). That is, (together with some notation shrinking), for this $(g_{\alpha_0,f_0}, g_{\alpha_1,f_1}, \mathsf{state})$, it holds that

$$\left| \Pr[k_{b^*} \leftarrow H_0(b^*,\alpha_1,f_1) : \mathcal{A}^*(k_{b^*},\mathsf{state}) = 1] - \Pr[k_{b^*} \leftarrow H_0(b^*,\alpha_0,f_0) : \mathcal{A}^*(k_{b^*},\mathsf{state}) = 1] \right| > \epsilon'.$$

Now, since $\epsilon' = \epsilon_{\mathsf{DPF}} + 2\epsilon_{\mathsf{PRG}} + 2\epsilon_{\mathsf{FSS}}$, then at least one of the following must hold:

1. $\epsilon_{\mathsf{DPF}} < |\Pr[k_{b^*} \leftarrow H_0(b^*,\alpha_1,f_1) : \mathcal{A}^*(k_{b^*},\mathsf{state}) = 1] - \Pr[k_{b^*} \leftarrow H_1(b^*,\alpha_1,\alpha_0,f_1) : \mathcal{A}^*(k_{b^*},\mathsf{state}) = 1]|$
2. $\epsilon_{\mathsf{PRG}} < |\Pr[k_{b^*} \leftarrow H_1(b^*,\alpha_1,\alpha_0,f_1) : \mathcal{A}^*(k_{b^*},\mathsf{state}) = 1] - \Pr[k_{b^*} \leftarrow H_2(b^*,\alpha_1,\alpha_0,f_1) : \mathcal{A}^*(k_{b^*},\mathsf{state}) = 1]|$
3. $\epsilon_{\mathsf{FSS}} < |\Pr[k_{b^*} \leftarrow H_2(b^*,\alpha_1,\alpha_0,f_1) : \mathcal{A}^*(k_{b^*},\mathsf{state}) = 1] - \Pr[k_{b^*} \leftarrow H_3(b^*,\alpha_0) : \mathcal{A}^*(k_{b^*},\mathsf{state}) = 1]|$
4. $\epsilon_{\mathsf{FSS}} < |\Pr[k_{b^*} \leftarrow H_3(b^*,\alpha_0) : \mathcal{A}^*(k_{b^*},\mathsf{state}) = 1] - \Pr[k_{b^*} \leftarrow H_4(b^*,\alpha_0,f_0) : \mathcal{A}^*(k_{b^*},z^*) = 1]|$
5. $\epsilon_{\mathsf{PRG}} < |\Pr[k_{b^*} \leftarrow H_4(b^*,\alpha_0,f_0) : \mathcal{A}^*(k_{b^*},\mathsf{state}) = 1] - \Pr[k_{b^*} \leftarrow H_0(b^*,\alpha_0,f_0) : \mathcal{A}^*(k_{b^*},z^*) = 1]|$

But, by Claims C.3-C.7, this cannot happen if $\mathcal{A}^*$ runs in time $T < T - \max_{i=1}^5 p_i(n_1+n_2)$, where each p_i is as appears within the corresponding claim. Therefore, security of $(\mathsf{Gen}^\otimes, \mathsf{Eval}^\otimes)$ holds, for the polynomial $p = \max_{i=1}^5 p_i$.

□

□

□

C.2 Reconstructing Prior DPF Constructions via Tensor Product

We demonstrate that for two particular choices of iterative applications of the FSS tensoring operation of Theorem 3.2, we directly obtain the (seemingly quite different) DPF constructions of Gilboa and Ishai [24] and of Boyle, Gilboa, and Ishai [8].

Recursion Structure 1. By recursively combining two DPFs with *equal* input bit lengths n, in each such step, the input length doubles from n to $2n$, and the FSS key size grows by a factor of 3. Thus, starting with a trivial DPF construction on 1-bit inputs (and $(\lambda+1)$-bit outputs)—i.e., just a secret sharing of the evaluation table, with size $2(\lambda+1)$—by applying the transformation $\log_2 n$ times we obtain a DPF for n-bit inputs. The resulting key size will be $2(\lambda+1)\cdot 3^{\log_2 n} \in O(\lambda n^{\log_2 3})$. The resulting DPF is that of Gilboa and Ishai [24].

Corollary C.9 (Reconstructing Gilboa-Ishai [24])**.** *There exists a polynomial $p(n)$ such that, if a (T,ϵ)-secure pseudorandom generator exists, then there exists a $(T-p(n),\epsilon\cdot p(n))$-secure FSS scheme for the multi-bit point function family $\mathcal{F}^\bullet : \{0,1\}^n \to \mathbb{G}$ for n-bit inputs, with key size bounded by $\frac{4}{3}(\lambda+1)n^{\log_2 3}$.*

Proof. We apply Theorem 3.2 recursively. For initial domain $\{0,1\}^d$, there exists a trivial $(T',0)$-secure FSS for the function family $\mathcal{F}^\bullet : \{0,1\}^d \to \{0,1\}^{\lambda+1}$ with key size $2^d(\lambda+1)$ (for any T': i.e., perfect security). Say we begin with (t',ϵ')-secure PRG. After i iterations of Theorem 3.2, we reach an FSS for the point function family for $(d\cdot 2^i)$-bit inputs with a resulting key size of $3^i 2^d(\lambda+1)$, and $(T'-\sum_{j=1}^{i} p'(2^j d), 5^i\cdot\epsilon')$-security, for fixed polynomial p' as guaranteed by Theorem 3.2 (where the 5 is taking a bound of equal (growing) values for $\epsilon_{\mathsf{DPF}},\epsilon_{\mathsf{PRG}},\epsilon_{\mathsf{FSS}}$). Thus, for a desired target domain size $N=2^n$, we must repeat $\left\lceil \log\left(\frac{n}{d}\right)\right\rceil$ times to reach this size; the resulting key size will then be $3^{\lceil \log(n/d)\rceil} 2^d(\lambda+1)$. Choosing the initial domain bit-length $d=2$ (to minimize $2^d/3^{\log_2 d}$), we reach a final key size bounded by $\frac{4}{3}(\lambda+1)n^{\log_2 3}$. The resulting FSS will have security no worse than $(T'-\log n\cdot p'(n), 5^{\log n}\cdot\epsilon) = (T'-p''(n), n^{\log_2 5}\cdot\epsilon)$ for some fixed polynomial $p''(n)$. Thus, taking $p(n)=\max\{p''(n), n^{\log_2 5}\}$, the claim holds. □

Recursion Structure 2. Observe, however that the key size of the tensor product of Theorem 3.2 grows *asymmetrically* in the key sizes of the two original DPFs: i.e., $\mathsf{size}_\otimes(\lambda, n_1+n_2) = \mathsf{size}_\bullet(\lambda,n_1) + 2\cdot\mathsf{size}_\bullet(\lambda,n_2)$. Thus, it is advantageous to instead recursively combine a DPF with input length n_1 together with a DPF of a *single*-bit input $n_2=1$ (which has a trivial construction with fixed key size s). In each step, the input length increases by 1 bit, and the FSS key size grows *additively* as $\mathsf{size}_\bullet(\lambda,n_1)+2s$. As the conclusion of n such steps, if we began with just the trivial 1-bit-input DPF construction, we will have constructed a DPF for n-bit inputs whose key size grows only linearly in n. The resulting DPF is the tree-based construction of Boyle, Gilboa, and Ishai [8].

Corollary C.10 (Reconstructing Boyle-Gilboa-Ishai [8])**.** *There exists a polynomial $p(n)$ such that, if a (T,ϵ)-secure pseudorandom function exists, then there exists a $(T-p(n),2n\epsilon)$-secure FSS scheme for the multi-bit point function family $\mathcal{F}^\bullet : \{0,1\}^n \to \mathbb{G}$ for n-bit inputs, with key size $4n(\lambda+1)$.*

Proof. We again apply Theorem 3.2 recursively, beginning with the trivial FSS $(\mathsf{Gen}^1,\mathsf{Eval}^1)$ for single-bit-input point functions $\mathcal{F}^\bullet : \{0,1\} \to \{0,1\}^{\lambda+1}$, with key size $2(\lambda+1)$ bits. In each iteration $i=1,\ldots,n-1$, apply Theorem 3.2 using $(\mathsf{Gen}^i,\mathsf{Eval}^i)$ together with $(\mathsf{Gen}^1,\mathsf{Eval}^1)$ to yield an FSS for $(i+1)$-bit-input point functions $\mathcal{F}^\bullet : \{0,1\}^{i+1} \to \{0,1\}^{\lambda+1}$, with resulting key size $\mathsf{size}_{i+1}(i+1,\lambda) = \mathsf{size}_i(i,\lambda) + 2\cdot\mathsf{size}_1(1,\lambda) = \mathsf{size}_i(i,\lambda) + 2\cdot 2(\lambda+1)$. Thus, after $n-1$ iterations, we obtain an FSS for the desired function class $\mathcal{F}^\bullet : \{0,1\}^n \to \{0,1\}^{\lambda+1}$ with key size $2(\lambda+1)+(n-1)4(\lambda+1) \le 4n(\lambda+1)$. Assuming a (T,ϵ)-secure PRG, the security of the resulting FSS (given $(n-1)$ applications of Theorem 3.2) will be no worse than $(T-n\cdot p(n),\epsilon')$,

where p is the polynomial guaranteed by Theorem 3.2, and $\epsilon' = \epsilon_1 + \sum_{i=1}^{n-1}(2\epsilon_1 + 2\epsilon_{\mathsf{PRG}}) \leq 2n\epsilon$ since ϵ_1 of $(\mathsf{Gen}^1, \mathsf{Eval}^1)$ is 0 due to its perfect security.

□

D An Indistinguishability-Based Security Definition

We quote here the alternative indistinguishability-based security definition of FSS from [8], naturally extended to the case of a general leakage function Leak. This definition is equivalent to our original one for every function family $\mathcal{F}$ and leakage function Leak for which Leak can be efficiently inverted, which will be the case for all $\mathcal{F}$ and Leak considered in this work. In our security proofs, it will be convenient to use this alternative definition.

- **Security:** Consider the following indistinguishability challenge experiment for set of corrupted parties $S \subset [m]$ of size $|S| \leq t$:
  1: The adversary outputs $(f_0, f_1, \mathsf{state}) \leftarrow \mathcal{A}(1^\lambda)$, where $f_0, f_1 \in \mathcal{F}$ with $\mathsf{Leak}(f_0) = \mathsf{Leak}(f_1)$.
  2: The challenger samples $b \leftarrow \{0,1\}$, $(k_1, \ldots, k_m) \leftarrow \mathsf{Gen}(1^\lambda, f_b)$.
  3: The adversary outputs a guess $b' \leftarrow \mathcal{A}((k_i)_{i \in S}, \mathsf{state})$, given the keys for corrupted S.

  Denote by $\mathsf{Adv}(1^\lambda, \mathcal{A})$ the adversary's advantage in guessing the challenge bit b: $\mathsf{Adv}(1^\lambda, \mathcal{A}) := \Pr[b = b'] - 1/2$. We say the scheme $(\mathsf{Gen}, \mathsf{Eval})$ is *t-secure* if for every $S \subseteq [m]$ of size $\leq t$ and non-uniform PPT adversary $\mathcal{A}$, there exists a negligible function ν, such that for all positive integer λ we have $\mathsf{Adv}(1^\lambda, \mathcal{A}) \leq \nu(\lambda)$.