Lower Bounds on FSS from Dynamic Data Structures

Niv Gilboa
Faculty of Computer and Information Science, Ben-Gurion University of the Negev, Be'er-Sheva, Israel

Daniel Weber
Faculty of Computer and Information Science, Ben-Gurion University of the Negev, Be'er-Sheva, Israel

Abstract

In Function Secret Sharing (FSS), a dealer with a given function $f : \{0,1\}^n \to \mathbb{G}$ from n bits to a commutative group $\mathbb{G}$ such that f is in a function class $\mathcal{F}$ shares succinct keys with two properties. Evaluating each key separately on a common input x results in additive shares of $f(x)$ and any subset of the keys does not provide information on f. Two-party FSS schemes which are reducible to One-way Functions (OWF) have applications in cryptography, complexity, and in practical data security systems.

We establish a two-way transformation between a two-party FSS scheme for a function class $\mathcal{F}$, which is black-box reducible to an OWF, or even black-box reducible to a family of Pseudo-Random Functions (PRF) and a dynamic data structure that supports range queries on $\mathcal{F}$. A data structure of this type enables dynamically adding functions to a multiset of functions $F \subseteq \mathcal{F}$, and answering range queries on the output of F, i.e., returning $\sum_{f \in F} f(x)$ for a query x. The data structures are defined in one of several models which abstract RAM.

The correspondence together with known lower bounds on the update time and the query time in data structures leads to the first non-trivial lower bounds on FSS schemes which are black-box reducible to PRF. These lower bounds apply to FSS schemes with polynomial key size and include:

- For $\mathcal{F}^d_{box}$, the class of all functions which assign a constant group element $\beta \in \mathbb{G}$ to any input in a specified d-dimensional box and 0 to all other inputs: if the key sharing function, Gen, runs in time polynomial in n and the evaluation function is Eval then:
  - If $d \geq 2$ and $\mathbb{G} = \mathbb{Z}_2$ then Eval's running time is $\Omega\left(\frac{n^{3/2}}{\log^3 n}\right)$.
  - If $d \geq 2$ and $\mathbb{G}$ is cyclic such that $\log|\mathbb{G}| = (1+\varepsilon)n$ then Eval's running time is $\Omega\left(\left(\frac{n}{\log n}\right)^2\right)$.
  - If $d > 2$ is a constant and further, Gen and Eval correspond to operations on data structures in the *Oblivious Group Model* (this includes all known FSS from OWF techniques), then the product of Eval's time and the key size is $\Omega(n^{d-1})$.
- For $\mathcal{F}_{mono}$, the class of all monomials $ax^b \in \mathbb{F}_{2^n}[X]$ such that $b \leq B$, assuming $n^{\omega(1)} \leq B \leq 2^{n/4}$: if Gen runs in polynomial time, then Eval's running time is $\Omega\left(\frac{n\sqrt{\log B}}{\log^2 n}\right)$.

2012 ACM Subject Classification Theory of computation → Cell probe models and lower bounds; Theory of computation → Cryptographic protocols

Keywords and phrases FSS, Data Structures, Lower Bounds, Black-Box Reductions

Digital Object Identifier 10.4230/LIPIcs.ITCS.2026.71

1 Introduction

In this work, we show a surprising connection between two seemingly very different objects: Function Secret Sharing (FSS) and dynamic data structures which support range queries.

FSS [11, 13] is a generalization of secret sharing [32, 3], which enables a dealer to distribute succinct and secret shares of a function in a given function class $\mathcal{F}$. In more detail, FSS is made up of a pair of probabilistic, polynomial-time algorithms (Gen, Eval) such that on input


© Niv Gilboa and Daniel Weber;
licensed under Creative Commons License CC-BY 4.0

$f \in \mathcal{F}$ and a number of parties $k \geq 2$, Gen produces k keys with three properties. First, the sum of executing Eval on each key with a common input x results in $f(x)$ for all x in the domain of f. In addition, an adversary that has access to at most $k-1$ keys does not obtain information on f, beyond its inclusion in $\mathcal{F}$. Finally, the keys are *succinct*, i.e. much smaller than the size of the truth table of f, preventing the trivial FSS scheme in which $f(x)$ is secret-shared independently for every possible input x.

In this work, as in most works on FSS, we are interested in FSS for two parties, i.e. $k = 2$. In this setting, the class of functions that admits FSS schemes depends on the computational power of the adversary. At one extreme, if the adversary is computationally unbounded, then even very simple function classes such as the class of all point functions, which assign 1 to a single point in the domain and 0 to all the others, require exponential key size [18, 11].

At the other extreme, assuming the hardness of some problems that imply additively homomorphic public-key encryption such as Decision Diffie-Hellman [12], variants of Learning With Errors (LWE) [15] or Decisional Composite Residuosity (DCR) [28] leads to FSS for the class NC^1 of all logarithmic depth circuits with constant fan-in, while stronger lattice assumptions imply FSS for all circuits of polynomial size [21]. While these FSS schemes that require public-key encryption have small keys and asymptotically efficient evaluation time, the concrete evaluation time is typically too high for use in practical applications of FSS.

Arguably, the most useful type of FSS schemes for practical applications are those whose security relies on the existence of One-way Functions (OWF). The classes of functions that have FSS from OWF are, as far as we currently know, far more limited than classes of functions that have FSS from LWE or DCR. Prominent examples of such classes include the class of point functions, the class of intervals, i.e. all functions that are 1 on a contiguous interval of $\{0,1\}^n$ and 0 on the rest, and more generally, the class of functions that are computed by a polynomial size decision tree, while leaking the topology of that tree [13].

FSS schemes that are reducible to OWF are typically defined based on cryptographic primitives that are polynomially equivalent to OWF, but are more convenient to use and in a sense more powerful such as length-doubling Pseudo-Random Generators (PRG) or Pseudo-Random Functions (PRF). Thus, the key size of the FSS scheme is determined by the number of PRF keys (or PRG seeds) it uses and the computational complexity of its generation and evaluation algorithms are determined by the number of calls to a PRF (or a PRG).

FSS schemes from OWF have found a range of applications due to their simplicity, their small key size and their concretely efficient generation and evaluation times. These applications include privately reading from and writing to databases [11], anonymous messaging [19], Multi-Party Computation (MPC) with non-arithmetic gates [14, 7], Pseudo-Correlation Generators (PCG), which are used to locally generate long, correlated pseudorandom strings for use in MPC [8, 9, 27], and others.

All of these applications of FSS from OWF depend on a number of useful schemes, especially in the two-party setting. However, a rapid string of successes in construction of schemes after FSS was defined in [11], has recently been followed by much slower progress. At this point it is unclear whether we have reached the limit on the class of functions that admit FSS from OWF and on the efficiency of such FSS schemes or whether significant progress is still possible.

A large part of the problem is our lack of lower bounds on FSS schemes. The only known lower bounds on key size and computational complexity of any two-party FSS are trivial bounds that are implied by the communication complexity and circuit complexity of functions in the class without the security requirement of FSS.

The main goal of this paper is to improve our understanding of the FSS landscape by proving non-trivial lower bounds on the efficiency measures of FSS. We achieve this by connecting FSS with the second topic of this work which is dynamic data structures for range queries.

In its simplest form, the problem of range queries considers points in some domain, e.g. $\mathbb{R}^d$, which are inserted into the data structure by Update operations. The data structure supports Query operations on a range, which return data on all the points in the data structure that are included in the range, e.g., the number of points in the range.

A more general form of the problem, which is required in this work, associates a weight w with each input point. The weight is an element of a commutative group $\mathbb{G}$. An Update operation updates the weight of a point, while a Query operation returns the sum of all the weights of points that fall in the queried range. For example, to support counting the number of points in a given range the weight of every point is $1 \in (\mathbb{Z}, +)$.

Data structures for range searching have been an active research area for quite some time [4, 1]. In particular, work on lower bounds for range searching improved our understanding of what is feasible in this domain [22, 16, 17, 30, 29, 20, 24, 25, 26]. Demonstrating lower bounds on data structures requires a formal model in which such lower bounds can be proved.

An important model for lower bounds is Yao's cell probe model [33]. This model, which is an abstraction of Random Access Memory, views the memory as a set of cells that each store w bits. An update probes a number of cells and updates them, while a query again probes a number of cells and returns a response. In both the update and the query operations the location of the cells read or written to can be an arbitrary function of the operation's input and the content of previously probed cells. The time of the update and query operations is measured by the number of probed cells, disregarding any other computation. The cell probe model is not restrictive and can therefore model almost any conceivable data structure. On the other hand this permissiveness results in relatively weak lower bounds.

Some of the lower bounds proved in the cell probe model are directly useful for our work. These bounds include a $\Omega\left(\left(\frac{n}{\log n}\right)^2\right)$ bound[1] from [24] on the query time of any data structure in the cell probe model for cells and weights with description size $\Theta(n)$ that supports weighted orthogonal range counting in two-dimensional space for any polynomial update time. In the version of the same problem in which the goal is to return the parity of the number of points in a range (i.e. a rectangle), the lower bound on the query time is only $\Omega\left(\frac{n^{3/2}}{\log^3 n}\right)$ [26]. The same work [26] discusses a data structure for a different problem in which the data structure represents a polynomial p over a binary field $\mathbb{F}_{2^n}$ with degree at most B for some $B \leq 2^{n/4}$. An update operation changes a coefficient in the polynomial and a query on point x returns the last bit of $p(x)$. Then, the query time has a lower bound $\Omega\left(\min\left\{\frac{n\sqrt{\log B}}{\log^2 nt_u}, \frac{\sqrt{B}}{(nt_u)^{O(1)}}\right\}\right)$, where t_u is the update time.

A more restrictive model in which lower bounds have been proved is the *Oblivious Group Model* [23]. In this model, the weights are again elements in a commutative group, but the data structure is not aware of the particular group. The memory cells store linear combinations of the weights of each item in the data structure. The data structure responds to range queries by computing linear combinations of the weights of the cells it probes until it achieves the sum of the weights of all the stored items that intersect the desired range. An

[1] Following FSS literature, the input domain in this work is $\{0,1\}^n$, while in most data structure works the input domain is $\{0, 1, \ldots, n\}$, leading to an exponential difference between the bound we report and the bound in the original works, which in this case is e.g. $\Omega\left(\left(\frac{\log n}{\log\log n}\right)^2\right)$.

update operation consists of reevaluating every cell with a linear combination that contains the weight of the updated point with non-zero coefficient. In this model, orthogonal range counting in d-dimensional space for a constant d has a lower bound on the update time t_u and the query time t_q which is $t_u t_q = \Omega(n^{d-1})$ (see the full version of [25]).

1.1 Our Contribution

The first contribution of this work is proof of a correspondence between two-party FSS that is black-box reducible to OWF, or even FSS that has black-box access to a family of PRF, and data structures for range queries.

One side of this correspondence is a transformation from an FSS scheme for a function class $\mathcal{F}$ to a data structure that answers range queries related to $\mathcal{F}$. Initially, the data structure is populated by one of the keys that Gen creates for the constant 0 function (which can be assumed to be part of $\mathcal{F}$) and the truth table of a random oracle, which plays the part of a PRF. Subsequently, each update to the data structure corresponds to an invocation of Gen for a function $f_i \in \mathcal{F}$ that uses a modified version of the oracle, replacing the key in the data structure, and a subset of the possible values in the truth-table of the new oracle. Each query with input $x \in \{0,1\}^n$ requires two calls to Eval to return $\sum_i f_i(x)$ where f_i are all the functions that have previously been inserted into the data structure.

A lower bound on the update time or query time in the data structure translates to a lower bound on Gen time or Eval time, respectively, in the FSS scheme.

However, the transformation outlined above does not create a data structure in the cell probe model, but in a generalization of it which we refer to as the *write probe* model. In this model, the memory cells can be initialized to arbitrary values prior to the first update and query operations, and further, the cost of updates is taken to be the number of cells to which the operation writes, essentially allowing an update operation to read all the cells for free. We show that a number of lower bounds in the cell probe model carry over with almost no change to the write probe model, enabling the translation of lower bounds on range queries to lower bounds on FSS.

An additional issue is that the natural form of the data structures stores points in a domain and provides a range in a query asking for the number of stored points in the range (or some other statistic on all the points in the range). However, the data structure that the transformation creates exchanges points and ranges, storing ranges (i.e. functions) in the data structure and providing a point x as a query asking for the number of ranges in which this point appears. This inversion between domain points and ranges in the two-dimensional orthogonal range query case is called *rectangle stabbing*. A folklore result shows that in general for orthogonal range searching the inversion is possible with small cost in query and update time.

The second contribution of this work is the first non-trivial lower bounds for efficiency measures of two-party FSS from OWF on *any* function class. We use the correspondence between FSS and data structures together with minor updates to lower bounds from the literature on data structures in the cell probe model to prove the following:

- For $\mathcal{F}^d_{box}$, the class of all functions which assign a constant group element $\beta \in \mathbb{G}$ to any input in a specified d-dimensional box and 0 to all other inputs: if the security of the FSS scheme is black-box reducible to PRG, the key sharing function, Gen, runs in time polynomial in n and the evaluation function is Eval then:
  - If $d \geq 2$ and $\mathbb{G} = \mathbb{Z}_2$ then Eval's running time is $\Omega\left(\frac{n^{3/2}}{\log^3 n}\right)$.
  - If $d \geq 2$ and $\mathbb{G}$ is cyclic such that $\log|\mathbb{G}| = (1+\varepsilon)n$ then Eval's running time is $\Omega\left(\left(\frac{n}{\log n}\right)^2\right)$.

- For $\mathcal{F}_{mono}$, the class of all monomials $ax^b \in \mathbb{F}_{2^n}[X]$ such that $b \leq B$, assuming $n^{\omega(1)} \leq B \leq 2^{n/4}$: if again the security of the FSS scheme is black-box reducible to PRG and Gen runs in polynomial time, then Eval's running time is $\Omega\left(\frac{n\sqrt{\log B}}{\log^2 n}\right)$.

The second side of the above correspondence is a transformation from a data structure for range queries in the *Oblivious Group Model* to two-party FSS. The transformation requires developing an *incremental* variant of known schemes for Distributed Multi-Point Functions (DMPF) [10], i.e., FSS for the class of multi-point functions. DMPF is the class of functions f that are 0 across the whole domain except for B points $\alpha_1, \ldots, \alpha_B$ such that $f(\alpha_i) = \beta_i$ for all $i = 1, \ldots, B$. The incremental variant of DMPF is a generalization of Incremental Distributed Point Functions [6] which enables the generation and evaluation of secret shares of values for all prefixes of the B points $\alpha_1, \ldots, \alpha_B$ with the same cost as DMPF.

Despite the restriction of the transformation to data structures in the oblivious group model, this transformation covers all the FSS schemes we currently know. Therefore, any lower bound for data structures in the oblivious group model applies to all FSS we know, and breaking such lower bounds will require substantially new FSS techniques

The final contribution of this paper is a lower bound on two-party FSS from OWF for the class of $\mathcal{F}^d_{box}$ for $d > 2$. The lower bound applies only to FSS that can be viewed as the result of our transformation from a data structure for range searching boxes in d-dimensional space in the oblivious group model and states that the product of the key size and Eval time is $\Omega(n^{d-1})$.

1.2 Our Techniques

1.2.1 FSS to Data Structure

When constructing the data structure from an FSS scheme, our goal is to maintain a multiset S of functions from a function class $\mathcal{F}$, allow updates that add functions to S, and efficiently answer queries on input x to which the response is $\sum_{f \in S} f(x)$.

Let $(\mathsf{Gen}^{\mathcal{O}}, \mathsf{Eval}^{\mathcal{O}})$ be a two-party FSS scheme for the class $\mathcal{F}$, where $\mathsf{Gen}^{\mathcal{O}}$ and $\mathsf{Eval}^{\mathcal{O}}$ denote the FSS generation and evaluation algorithms with access to an oracle $\mathcal{O}$. $\mathsf{Gen}^{\mathcal{O}}$ takes as input a security parameter 1^λ and a function $f \in \mathcal{F}$ and returns two keys k_0, k_1. $\mathsf{Eval}^{\mathcal{O}}$ takes as input 1^λ, a party index $b \in \{0,1\}$, the associated key k_b, and a point x and returns the evaluation result at x. The correctness of FSS ensures that $\sum_{b=0}^{1} \mathsf{Eval}^{\mathcal{O}}(1^\lambda, b, k_b, x) = f(x)$. In many cases, it is convenient to use the vectors $\mathsf{EvalAll}(1^\lambda, i, k_i) = (\mathsf{Eval}^{\mathcal{O}}(1^\lambda, i, k_i, x))_{x \in \{0,1\}^n}$.

In the FSS literature, a length-doubling Pseudo-Random Generator (PRG) is most commonly used as the oracle $\mathcal{O}$. In this work, we assume that the FSS has access to a more powerful primitive: Pseudo-Random Functions (PRF), since it strengthens our results. We use t_g for the number of oracle queries Gen makes and t_e for the number of oracle queries Eval makes.

We work under the assumption of fully black-box security [2, 13], where if there is an algorithm A that breaks the FSS with probability ε, then there is an algorithm B that uses A as an oracle and breaks the PRF in polynomial time with success probability polynomial in $\varepsilon, \frac{1}{\lambda}$.

The transformation we show requires a single invocation of $\mathsf{Gen}^{\mathcal{O}}$ for each update operation and two invocations of $\mathsf{Eval}^{\mathcal{O}}$ to respond to each query. Storing a single function could easily be done using the FSS by randomly sampling an oracle $\mathcal{O}$ and generating $k_0, k_1 \leftarrow \mathsf{Gen}^{\mathcal{O}}(1^\lambda, f)$. When answering queries, we compute $\mathsf{Eval}^{\mathcal{O}}(1^\lambda, 0, k_0, x) + \mathsf{Eval}^{\mathcal{O}}(1^\lambda, 1, k_1, x)$.

To add more functions, we will use a procedure we call side-switching. In this procedure, we take a function f, a key k_i and have an oracle $\mathcal{O}$, we return a new key k'_{1-i} and a new oracle $\mathcal{O}'$ which is only different from $\mathcal{O}$ in very few positions such that $\mathsf{EvalAll}^{\mathcal{O}}(1^\lambda, i, k_i) + \mathsf{EvalAll}^{\mathcal{O}'}(1^\lambda, 1-i, k'_{1-i}) = f$.

From [11] we can assume WLOG that the all-zero function 0 is in the function class $\mathcal{F}$. Using two side-switching operations with 0 and then f we can therefore obtain a key k'_0 and oracle $\mathcal{O}''$ with $\mathsf{EvalAll}^{\mathcal{O}''}(1^\lambda, 0, k'_0) = f - \mathsf{EvalAll}^{\mathcal{O}'}(1^\lambda, 1, k'_1) = f - (0 - \mathsf{EvalAll}^{\mathcal{O}}(1^\lambda, 0, k_0)) = f + \mathsf{EvalAll}^{\mathcal{O}}(1^\lambda, 0, k_0)$.

This gives us a way to build the data structure. We maintain two keys and two oracles. At the start they are initialized such that $\mathsf{EvalAll}^{\mathcal{O}_0}(1^\lambda, 0, k_0) + \mathsf{EvalAll}^{\mathcal{O}_1}(1^\lambda, 1, k_1) = 0$, and when doing updates we use the side-switching operations to update $\mathcal{O}_0$ and k_0 to $\mathcal{O}'_0, k'_0$ such that $\mathsf{EvalAll}^{\mathcal{O}'_0}(1^\lambda, 0, k'_0) = \mathsf{EvalAll}^{\mathcal{O}_0}(1^\lambda, 0, k_0) + f$.

The only thing left to explain is how to achieve the side-switching operation. Note that because we are working in the write probe model the computation required does not matter.

Given a key $k_i \leftarrow \mathsf{Gen}^{\mathcal{O}}(1^\lambda, f)_i$ and the function f, we want to efficiently find an oracle $\mathcal{O}'$ and an execution trace (e.g. the random choices taken) of $\mathsf{Gen}^{\mathcal{O}'}$ which returns k_i, and from it we can obtain k_{1-i}. We shall then use the FSS's security to argue that the success of this procedure cannot significantly depend on k_i being originally generated from f, and this procedure must work with high probability for all keys.

We cannot just use $\mathcal{O}' = \mathcal{O}$, as that requires breaking the security of the FSS. To bypass this obstacle, we utilize the fact that the required efficiency is in the context of a reduction that is fully black-box. This means that the algorithm is allowed to do exponential work, as long as it invokes the oracle a polynomial number of times. It can therefore search for a second oracle $\mathcal{O}'$ and execution trace of $\mathsf{Gen}^{\mathcal{O}'}$ which produces k_i. The execution of $\mathsf{Gen}^{\mathcal{O}'}$ only queries t_g oracle locations, so it suffices to change $\mathcal{O}$ at these locations.

This gives a similar oracle $\mathcal{O}'$ and a new key k_{1-i} such that $\mathsf{EvalAll}^{\mathcal{O}'}(1^\lambda, 0, k_0) + \mathsf{EvalAll}^{\mathcal{O}'}(1^\lambda, 1, k_1) = f$. This is close to the desired outcome, but what we actually want is to have $\mathsf{EvalAll}^{\mathcal{O}}(1^\lambda, i, k_i)$ with the original oracle $\mathcal{O}$ in place of $\mathsf{EvalAll}^{\mathcal{O}'}(1^\lambda, i, k_i)$ with the second oracle $\mathcal{O}'$. To resolve this, we search for $\mathcal{O}'$ that does not "contradict" $\mathcal{O}$ on the values that the FSS requires. That is, we start by computing $\mathsf{EvalAll}^{\mathcal{O}}(1^\lambda, i, k_i)$ and keep track of the locations it queries. When searching for $\mathcal{O}'$, we ensure that it returns the same values as $\mathcal{O}$ when querying these locations. Now, the side-switching procedure makes oracle queries – up to $2^n t_e$ queries. By using $\lambda' = \lambda + n + \log(t_e)$ instead of the original λ in the FSS, we get from the reduction that if this new side-switching method does not work with high probability, we can break the oracle PRF with good probability using just $\mathrm{poly}(\lambda) 2^n t_e$ queries, which is impossible for a random oracle.

We can thus conclude that side-switching works correctly with high probability, and we have our data structure. There are a few details involved in proving it works correctly when it is used many times, as the oracle and key distribution may shift from the original distribution, which are worked out precisely in Lemma 17.

1.2.2 Data Structure to FSS

Recall that a data structure in the oblivious group model consists of adding constant multiples of the weight to some cells in Update and returning a linear combination of cells in Query.

We can consider the updates as a matrix $\mathcal{U}$, whose rows are indexed by the possible input points and columns are indexed by the memory cells. The content of the matrix at i, j is the coefficient of input point i in the j-th memory cell.

The complexity of an update i is the number of non-zero elements in the i-th row of $\mathcal{U}$, and t_u is the maximum number of non-zero elements in a row.

We can also regard the queries as a matrix $\mathcal{Q}$, whose rows are indexed by memory cells and whose columns are indexed by the possible queries. The content of the matrix at i, j is the coefficient of the i-th memory cell in the linear combination for query j.

The complexity of a query j is the number of non-zero elements in the j-th column of $\mathcal{Q}$.

The weight of input point i in query j can be found by looking at the content of the product $\mathcal{UQ}$ at i, j.

We can now construct an FSS for the function family whose functions are the input points of the range queries problem, and the functions' domain is the queries of the range queries problem solved by the data structure. We do this by having Gen share a DMPF of the appropriate row of $\mathcal{U}$, $\mathcal{U}_f$. In Eval we will compute a linear combination of the values of $\mathcal{U}$ shared in the DMPF, $\sum_{\mathcal{Q}_{j,x}\neq 0} \mathcal{Q}_{j,x}\mathsf{DMPF.Eval}(1^\lambda, i, k_i, j)$. When summing both sides we have $\mathsf{DMPF.Eval}(1^\lambda, 0, k_0, j) + \mathsf{DMPF.Eval}(1^\lambda, 1, k_1, j) = \mathcal{U}_{f,j}$, so we get $\sum_{1\leq j\leq S} \mathcal{U}_{f,j}\mathcal{Q}_{j,x} = (\mathcal{UQ})_{f,x}$ which is exactly the desired result.

While for some problems like DPF or DMPF this approach gives the best known result (up to constant factors), there are also many problems (DCF, prefix DPF) where this solution is worse by a factor of n. Essentially, this is due to costs incurred by the DMPF that the specific schemes can avoid by utilizing the specific patterns in the matrices.

We construct a combination of DMPF and incremental DPF [6], called incremental DMPF. This variant enables sharing a binary tree with up to B nodes and values at its nodes, instead of having any B points, but its cost is proportional to B instead of nB. The evaluation is also incremental, which means we can traverse the tree and evaluate as we walk, instead of having to traverse a path from the root to a leaf for each evaluation.

FSS schemes based on Incremental DMPF obtain for all function classes key size and evaluation time that are equivalent to the best known direct FSS constructions up to constant factors. The correspondence of such FSS schemes with data structures in the oblivious group model means that the lower bounds on the data structures hold for FSS based on Incremental DMPF.

2 Preliminaries

▶ **Notation 1.** *U_λ^O is the uniform distribution on functions $\mathcal{O} : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}^\lambda$. $U_\lambda^{O'}$ is the uniform distribution on functions $\mathcal{O} : \{0,1\}^\lambda \to \{0,1\}^\lambda$.*

2.1 FSS

We adapt the definition of FSS to add an oracle and specialize it to two parties. As in [13], we assume that a function description contains the input size 1^n.

▶ **Definition 2** (FSS, adapted from [13]). *A (1-secure 2-party) FSS scheme for a function family $\mathcal{F}$ with an oracle $\mathcal{O}$ is a pair of*

1. *a PPT algorithm $\mathsf{Gen}^{\mathcal{O}}(1^\lambda, \hat{f})$: takes a description of a function $\hat{f} \in \mathcal{F}$, and returns two keys, k_0, k_1.*
2. *a deterministic polynomial algorithm $\mathsf{Eval}^{\mathcal{O}}(1^\lambda, i, k_i, x)$: takes a key and a point to do the evaluation and returns a group element.*

We define the key size $t_k = t_k(\lambda, n)$ as the worst-case number of λ-bit cells required to represent a key returned by Gen. We define the evaluation efficiency $t_e = t_e(\lambda, n)$ of an FSS as the maximum number of oracle queries and key accesses (to λ-bit cells) done by Eval. Finally, we define the generation efficiency $t_g = t_g(\lambda, n)$ as the maximum number of oracle queries done in Gen.

▶ **Definition 3** (FSS correctness, from [13]). *A (1-secure 2-party) FSS scheme for a function family $\mathcal{F}$ with an oracle is correct if for all oracles $\mathcal{O} : \{0,1\}^\lambda \to \{0,1\}^\lambda$, $f \in \mathcal{F}$ with input size n and input $x \in \{0,1\}^n$, if $k_0, k_1 \leftarrow \mathsf{Gen}^{\mathcal{O}}(1^\lambda, f)$ then* $\Pr[\mathsf{Eval}^{\mathcal{O}}(1^\lambda, 0, k_0, x) + \mathsf{Eval}^{\mathcal{O}}(1^\lambda, 1, k_1, x) = f(x)] = 1$

Another useful function is EvalAll, which executes Eval in all locations at the same time. For our purposes we can simply regard it as being implemented by calling Eval for each value of x, but in many schemes it can be implemented more efficiently.

We also define

▶ **Definition 4** (FSS observation). *Given an FSS scheme* $(\mathsf{Gen}, \mathsf{Eval})$*, a party index $i \in \{0,1\}$ and a key k_i, we define* $\mathrm{Obs}^{\mathcal{O}}(1^\lambda, i, k_i)$ *as a partial function containing all values of $\mathcal{O}$ that are queried while running* $\mathsf{EvalAll}^{\mathcal{O}}(1^\lambda, i, k_i)$.

Note that if $\mathrm{Obs}^{\mathcal{O}_1}(1^\lambda, i, k_i) = \mathrm{Obs}^{\mathcal{O}_2}(1^\lambda, i, k_i)$ then $\mathsf{EvalAll}^{\mathcal{O}_1}(1^\lambda, i, k_i) = \mathsf{EvalAll}^{\mathcal{O}_2}(1^\lambda, i, k_i)$.

We further define Gen' which is a modified version of Gen. In addition to the two keys it also outputs an execution trace tr, which is a partial function containing input and output pairs for all oracle invocations that Gen performed.

As for secrecy we will specialize the definition of fully black-box reductions (or BBB-reductions) [2, 31] to the case of FSS being secure from the oracle being a PRF.

The usual definition of FSS security (without an oracle) states that there is a simulator Sim such that $\mathsf{Gen}(1^\lambda, f)_i$ is indistinguishable from $\mathsf{Sim}(1^\lambda, \mathsf{Leak}(f))$ for an acceptable leakage function $\mathsf{Leak}(f)$. When adding an oracle to the definition, we can consider two variants: a weaker one in which the simulator has access to the same oracle that Gen and Eval call, and a stronger one in which the Sim simulates the correct distribution without access to the oracle. Our proof requires the stronger variant.

In particular, cases where keys are pseudorandom fit under this more restrictive definition.

▶ **Definition 5** (FSS fully black-box secrecy, adapted from [2, 13]). *We say that an FSS scheme* $(\mathsf{Gen}, \mathsf{Eval})$ *for a function family $\mathcal{F}$ with an oracle is fully black-box reducible to PRF with reduction constant c if for any party $i \in \{0,1\}$ there exists a PPT simulator* Sim_i *and a PPT PRF-distinguisher $\mathcal{S}$ such that for every security parameter λ, function f with description size $|f|$, adversary $\mathcal{A}$* (not necessarily polynomial-time) *and oracle $\mathcal{O} : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}^\lambda$, if*

$$\left|\Pr[\mathcal{A}^{\mathcal{O}}(1^\lambda, \mathsf{Sim}_i(1^\lambda, \mathsf{Leak}(f_\lambda))) = 1] - \Pr[\mathcal{A}^{\mathcal{O}}(1^\lambda, k_i) = 1 \mid k_0, k_1 \leftarrow \mathsf{Gen}^{\mathcal{O}}(1^\lambda, f_\lambda)]\right| \geq \varepsilon$$

then

$$\left|\Pr[\mathcal{S}^{\mathcal{O},\mathcal{A},\mathcal{O}(x,\cdot)}(1^\lambda, f) = 1 \mid x \leftarrow U_\lambda] - \Pr[\mathcal{S}^{\mathcal{O},\mathcal{A},\mathcal{O}'}(1^\lambda, f) = 1 \mid \mathcal{O}' \leftarrow U_\lambda^{O'}]\right| \geq \left(\frac{\varepsilon}{\lambda + |f|}\right)^c$$

▶ **Example 6.** The DPF from [13] is fully black-box secure (when simulating length-doubling PRG via two calls to the PRF)

Proof. In the security proof of [13] there is a sequence of $n+1$ hybrid distributions such that H_0 is the same distribution as $\mathsf{Gen}(1^\lambda, f)_i$, H_n is the same distribution as $\mathsf{Sim}_i(1^\lambda, \mathsf{Leak}(f))$. It also gives algorithms $Hyb_0, \ldots, Hyb_{n-1}$ such that the distribution of $Hyb_i(U_{2\lambda})$ is H_i and of $Hyb_i(\mathrm{PRG}(U_\lambda))$ is H_{i+1}. The distinguisher $\mathcal{S}$ will randomly choose $0 \leq i < n$ and call $\mathcal{A}$ on the result of $Hyb_i(\mathcal{O}'(0) \| \mathcal{O}'(1))$. In the $\Pr[\mathcal{S}^{\mathcal{O},\mathcal{A},\mathcal{O}(x,\cdot)}(1^\lambda, f) = 1 \mid x \leftarrow U_\lambda]$ case we will invoke $\mathcal{A}$ on one of $H_1, H_2, \ldots, H_n$ and in the $\Pr[\mathcal{S}^{\mathcal{O},\mathcal{A},\mathcal{O}'}(1^\lambda, f) = 1 \mid \mathcal{O}' \leftarrow U_\lambda^{O'}]$ case we will invoke it on one of $H_0, H_1, \ldots, H_{n-1}$. The difference is exactly H_0 versus H_n which is $\mathsf{Gen}(1^\lambda, f)_i$ versus $\mathsf{Sim}_i(1^\lambda, \mathsf{Leak}(f))$, so this distinguisher has advantage $\frac{\varepsilon}{n}$. ◀

A particular type of FSS which we require in this paper is DMPF [10], which is an FSS which can represent all functions $\{0,1\}^n \to \mathbb{G}$, but leaks an upper bound B on the number of non-zero points in the function, and whose runtime depends on B.

Using [10]'s DMPF construction based on Oblivious Key-Value Store (OKVS), instantiated with the OKVS from [5], gives a DMPF secure from One-way Functions with key size $t_k = O(Bn)$, generation efficiency $t_g = O(Bn)$ and evaluation efficiency $O(n)$, as decoding the OKVS of [5] requires reading $O(\lambda)$ **continuous** bits, so with the key being stored in λ-bit cells we only need to read $O(1)$ cells per decoding.

2.2 Data Structures

In dynamic data structures, we consider the problem of maintaining a data structure which is able to perform updates and answer queries.

We are interested in particular in the problem of (weighted) range counting [25, 24]. This problem has two similar variants:

1. In the first, we have a function family $\mathcal{F}$ with an abelian group as a range $\mathbb{G}$ and integer weights $w : \mathcal{F} \to \mathbb{Z}$, and we need to support changing the weight of a particular function for updates and computing $\sum_{f\in\mathcal{F}} w(f)f(x)$ for a given x for queries.
2. In the second presentation, the range of the functions is $\mathbb{Z}$, and the weights come from an abelian group. Here too we need to support changing weights and computing $\sum_{f\in\mathcal{F}} w(f)f(x)$.

There are multiple models in which these data structures can be considered. We are generally interested in two efficiency parameters, t_u, the update cost, and t_q, the query cost. One of the most restricted is the oblivious group model [25] where we assume the weights come from a black-box abelian group $\mathbb{G}$, and the data structure is a long array of group elements. This model fits the second presentation of weighted range counting. Queries consist of summing at most t_q array elements with integer weights, and an update of adding x to the weight of f consists of adding integer multiples of x to at most t_u array elements. In the oblivious model the integer multipliers are independent of the preexisting data structure. Formally, we can define it as:

▶ **Definition 7** (Oblivious group model, adapted from [25]). *A data structure in the oblivious group model for $\mathcal{F}$ with domain $\{0,1\}^n$ and range $\mathbb{Z}$ is a factorization of the matrix $M \in \mathbb{Z}^{|\mathcal{F}|\times 2^n}$ representing the function family to an update matrix $\mathcal{U} \in \mathbb{Z}^{|\mathcal{F}|\times S}$ and a query matrix $\mathcal{Q} \in \mathbb{Z}^{S\times 2^n}$, such that each row of $\mathcal{U}$ has at most t_u non-zero values, and each column of $\mathcal{Q}$ has at most t_q non-zero values. We call S the internal size of the data structure.*

Another commonly considered model is the cell probe model. In this computational model the data structure is a long array of ℓ-bit cells. Updates can read and modify at most t_u cells, and queries can read t_q cells.

One data structure problem we consider multiple times is the 2D range counting problem, which we define as storing m points on an $[U] \times [U]$ grid for $U = m^{O(1)}$, potentially with weights, and answering queries of the number/sum of weights of points stored in the data structure which are dominated by a given query point q – that is, points that are smaller than q in both coordinates.

In the rectangle stabbing problem we maintain rectangles contained in a $[U] \times [U]$ grid, potentially with weights, and want to find the number/sum of weights of rectangles containing a given point.

These two problems are equivalent by the folklore reduction outlined in the full version of [26].

These problems can be defined similarly in an arbitrary dimension d, and the same reduction applies (when we treat d as a constant).

3 Write Probe Model

The target of our reduction will be a new model, even more general than the cell probe model, which we call the write probe model.

In the write probe model we have an unrestricted initialization step which occurs before all updates and queries. Updates can modify at most t_u cells given by an arbitrary function of the update and the current content of the data structure. Queries can adaptively read up to t_q cells – the location of the next cell to read can be an arbitrary function of the contents of previous read cells and the query.

We first observe that as long as errors are detectable, we can allow for a small error probability:

▶ **Proposition 8.** *Let there be a data structure in the write probe model for a problem $\mathcal{P}$ with worst-case update time t_u and worst-case query time t_q. Suppose each update requires $O(d)$ cells to describe, and fails with total failure probability $O(\frac{1}{dm})$ for m updates. Suppose further that if an update operation fails then it returns an error value, notifying the caller of this failure. Then, we have a data structure in the write probe model for the same problem with worst-case update time $O(t_u)$ and* expected *query time $O(t_q)$ for m updates, and no failure probability.*

Proof. The modified data structure will additionally keep a list of all updates which were performed, and whether any update operation failed. In query, if the data structure has detected any failure, it will instead directly compute the answer from the list of updates. This requires reading $O(dm)$ cells of updates, but only with probability $O(\frac{1}{dm})$, so in expectation it only changes the query complexity by $O(1)$. ◀

We can show that while the write probe model is theoretically stronger than the cell probe model, in practice they are very similar, and in particular many cell probe lower bounds transfer without any modifications to the proof. The underlying reason for this is the use of the *chronogram* method:

Our description is based on the overview of [24, 26]. In the chronogram method of Fredman and Saks [22], we have m updates which are batched to $\log_\beta(m)$ batches $U_1, U_2, \ldots, U_{\log_\beta(m)}$ (for some parameter β, slightly larger than t_u), with batch i consisting of performing β^i updates to the data structure. The batches are performed from big to small. Very broadly, the argument then follows that for any i the batches $j > i$ can contain no information about update i, as they occur before it. Additionally, batches $j < i$ have total size $O(t_u\beta^{i-1}) = o(\beta^i)$, so they cannot contain much information about batch i.

The argument then roughly follows information theoretically by showing that, together with some partial information about the data saved in the data structure while performing U_i, one can recover more information about U_i than is possible.

Remarkably, this methodology doesn't care at all about which cells the data structure *reads* during an update, only about the *modifications* it makes to it. In essence, this is because each batch is only treated as statically, where we look at the overall modifications it made to the data structure. Additionally, this methodology is not affected if the data structure is initialized from whatever distribution we wish. This means that results following it apply as-is to the write probe model, and in particular:

▶ **Theorem 9** (Adapted from [24] Theorem 1). *Any data structure for dynamic weighted orthogonal range counting in the* write probe model *for m updates must satisfy $t_q = \Omega((\log m/\log(wt_u))^2)$. Here w is the cell size, t_q is the expected average query time and t_u is the worst case update time. This lower bound holds when the weights of the inserted points are $\Theta(\log m)$-bit integers.*

Proof Sketch. We argue that Larsen's proof works as-is for the write probe model case.

As described previously, [24] uses the chronogram methodology. To fill in more details, the information given about U_i is a set of cells S'_i which is a subset of the cells modified by U_i, such that many queries are *resolved* by S'_i – that is, all of the cells with last update time i that are read when performing those queries are in S'_i. When encoding we describe S'_i and a set of resolved queries Q (as well as the batches $j < i$), and we condition on the batches $> i$. In the decoding we can then evaluate all queries in Q: when reading a location, if it was modified in batches $< i$ we have the correct value, else if it is in S'_i we also know the stored value, and otherwise, since S'_i resolves Q, this must be from a batch $> i$ which we know.

The main challenge here is showing that if a query does $o(\log_\beta(n))$ probes to the cells modified by U_i for $i \geq \frac{15}{16}\log_\beta n$ then there exist sets $|S'_i| = O(\beta^{i-1}w)$, $|Q| = \Omega(\beta^{i-3/4})$, such that Q is resolved by S'_i, and the queries in it are sufficiently well-spread that their answers provide more information about U_i than it takes to specify S'_i and Q, yielding the desired contradiction.

This existence is shown in cell probe mode by Lemma 3 in [24], and as the proof of the lemma doesn't depend on the initialization nor does it utilize a bound on the number of cells an update reads, only the amount of cells it modifies, it also applies to the write probe model. ◀

Using the equivalence to rectangle stabbing mentioned in Section 2.2 we also get a lower bound for this problem.

▶ **Corollary 10.** *Any data structure for dynamic weighted rectangle stabbing in the* write probe model *for m updates must satisfy $t_q = \Omega((\log m/\log(wt_u))^2)$. Here w is the cell size, t_q is the expected average query time and t_u is the worst case update time. This lower bound holds when the weights of the inserted rectangles are $\Theta(\log m)$-bit integers.*

▶ **Theorem 11** (One-way weak simulation theorem for the write probe model, adapted from [26] Theorem 1). *Let $\mathcal{P}$ be a dynamic boolean data structure problem, with m random updates grouped into epochs $\boldsymbol{\mathcal{U}} = U_1, U_2, \ldots, U_\ell$, such that $|U_i| = \beta^i$, followed by a single query $q \in \boldsymbol{\mathcal{Q}}$. If $\mathcal{P}$ admits a dynamic data structure* in the write probe model *with cell size w, worst-case update time t_u and average (over $\boldsymbol{\mathcal{Q}}$) expected query time t_q, satisfying $t_q, t_u, w \leq m^{0.1}$, then there exists some epoch $i \in [\ell]$ for which there is a one-way randomized communication protocol in which Alice sends Bob a message of only $\beta^i/(wt_u)^{\Theta(1)}$ bits, and after which Bob successfully computed $\mathcal{P}(q, \boldsymbol{\mathcal{U}})$ with probability at least $1/2 + \exp(-t_q \log^2(wt_u)/\sqrt{\log m})$.*

Proof Sketch. The proof of Larsen, Weinstein and Yu also works without modification for the write probe model. One should note, however, that the epoch associated with a cell changes its meaning from the last epoch in which it was written or read to the last epoch in which it was written. Looking at how their proof uses associated cells, however, this is completely fine.

As an overview, the proof works by having Alice send Bob a random sample of the cells written during epoch i. The probability of this sample sufficing to resolve a query is sufficiently large that if Bob could detect this random event we would finish, by outputting a random bit if it doesn't happen and simulating the data structure if it does.

However, Bob cannot detect that event. Instead, Bob will proceed as if this event occurred, and consider the set S_q of locations he probes when answering the query. By the Peak-to-Average Lemma from [26], there exists a small subset $Y \subseteq S_q$, such that conditioned on the values of the data structure in Y, the maximum-likelihood answer to query, conditioned on Bob's information, has sufficient advantage over random guessing, and Bob can compute this Y privately. However, Alice does not know Y. To resolve this, she will use *public randomness* to randomly sample some cells, and out of those send the cells associated with epoch i to Bob. Using the public randomness Bob can determine whether Y is contained in the sampled cells. If it is not he will abort and return a random bit, otherwise he knows the values of Y and can return the more likely query answer conditioned on his information.

The proof in [26] shows that the described protocol succeeds with sufficient probability, and this proof works identically in the write probe model.

Note that no step in the argument depends on the cells read during updates, or on the initialization, so it works as-is for the write probe model. ◀

Since the weak simulation theorem also applies for the write probe model, so do the results from [26] which depend on it, and in particular we have:

▶ **Theorem 12** (Adapted from [26] Theorem 2). *Any* write probe *data structure for least-bit polynomial evaluation over* $\mathbb{F}_{2^n}$ *supporting degree* B *polynomials and* m *updates,*[2] *having cell size* w*, worst case update time* t_u *and expected query time* t_q *must satisfy* $t_q = \Omega(\min(\frac{n\sqrt{\log \min(m,B)}}{\log^2(t_u w)}, \frac{\sqrt{\min(m,B)}}{(t_u w)^{O(1)}}))$.

▶ **Theorem 13** (Adapted from [26] Corollary 1). *Any* write probe *data structure for 2D range parity, having cell size* w*, worst case update time* t_u *and expected query time* t_q *must satisfy* $t_q = \Omega(\frac{\log^{3/2} m}{\log^3(t_u w)})$.

And using the reduction to 2D rectangle parity we also get a lower bound for it:

▶ **Corollary 14** (Adapted from [26]). *Any* write probe *data structure for 2D rectangle parity, having cell size* w*, worst case update time* t_u *and expected query time* t_q *must satisfy* $t_q = \Omega(\frac{\log^{3/2} m}{\log^3(t_u w)})$.

4 FSS to Data Structure

For the following section, let $\mathcal{F}$ be a function family that admits an FSS with key size $t_k = t_k(\lambda, n)$, generation time $t_g = t_g(\lambda, n)$ and evaluation time $t_e = t_e(\lambda, n)$ which is fully black-box reducible to PRF with reduction constant c. We also assume that the leakage consists only of n and $\mathbb{G}$, and will ignore it in this section. We use $\mathcal{F}_n$ for the functions in $\mathcal{F}$ with input size n and assume that the function description size is bounded by a polynomial in λ, $sz(\lambda)$.

We start by proving a mostly technical result that uses the FSS scheme's black-box security to show that an adversary which makes at most $\varepsilon 2^\lambda$ queries cannot distinguish between a random oracle and a key generated from it and a random oracle and a key sampled from Sim with significantly higher advantage than ε.

[2] Compared to [26] we have different parameters for the degree and number of updates. We reduce the larger one to the smaller to obtain the result.

▶ **Lemma 15.** *Let $i \in \{0,1\}$. There is a polynomial q such that for any $\varepsilon = \varepsilon(\lambda)$, any algorithm with oracle that makes $O(\varepsilon 2^\lambda)$ oracle queries $A^{\mathcal{O}}(1^\lambda, k)$ and any $f \in \mathcal{F}_n$,*

$$\left|\Pr[A^{\mathcal{O}_1}(1^\lambda, \mathsf{Gen}^{\mathcal{O}_1}(1^\lambda, f)_i) \mid \mathcal{O}_1 \leftarrow U_\lambda^O] - \Pr[A^{\mathcal{O}_2}(1^\lambda, \mathsf{Sim}_i(1^\lambda)) \mid \mathcal{O}_2 \leftarrow U_\lambda^O]\right| \leq q(\lambda)\varepsilon^{1/c}$$

Proof. Let v be a constant such that A makes at most $v\varepsilon 2^\lambda$ oracle queries. Let $h(\lambda + |f|)$ be an increasing polynomial bounding the runtime of $\mathcal{S}$ from Definition 5. We will set $q(\lambda) = (v+1)(\lambda + sz(\lambda))h(\lambda + sz(\lambda))$.

Set $p(\mathcal{O}) = |\Pr[A^{\mathcal{O}}(1^\lambda, \mathsf{Gen}^{\mathcal{O}}(1^\lambda, f)_i)] - \Pr[A^{\mathcal{O}}(1^\lambda, \mathsf{Sim}(1^\lambda))]|$. By the triangle inequality, it suffices to prove $\mathbb{E}[p(\mathcal{O})] \leq q(\lambda)\varepsilon^{1/c}$.

Suppose by way of contradiction that $\mathbb{E}[p(\mathcal{O})] \geq q(\lambda)\varepsilon^{1/c}$.

By Definition 5,

$$\left|\Pr[\mathcal{S}^{\mathcal{O},\mathcal{A},\mathcal{O}(x,\cdot)}(1^\lambda, f) = 1 \mid x \leftarrow U_\lambda] - \Pr[\mathcal{S}^{\mathcal{O},\mathcal{A},\mathcal{O}'}(1^\lambda, f) = 1 \mid \mathcal{O}' \leftarrow U_\lambda^{O'}]\right| \geq \left(\frac{p(\mathcal{O})}{\lambda + |f|}\right)^c$$

Applying Jensen's inequality, we get

$$\mathbb{E}_{\mathcal{O}}\left[\left|\Pr[\mathcal{S}^{\mathcal{O},\mathcal{A},\mathcal{O}(x,\cdot)}(1^\lambda, f) = 1 \mid x \leftarrow U_\lambda] - \Pr[\mathcal{S}^{\mathcal{O},\mathcal{A},\mathcal{O}'}(1^\lambda, f) = 1 \mid \mathcal{O}' \leftarrow U_\lambda^{O'}]\right|\right] \geq \varepsilon((v+1)h(\lambda + sz(\lambda)))^c \geq (v+1)\varepsilon h(\lambda + sz(\lambda))$$

This is impossible, however – $\mathcal{S}$ makes at most $v\varepsilon h(\lambda + sz(\lambda))2^\lambda$ oracle queries (counting indirect queries using the A oracle). This means that it can query $\mathcal{O}$ with x as the first coordinate only with probability $v\varepsilon h(\lambda + sz(\lambda))$, and otherwise there is no way for it to distinguish between $\mathcal{O}(x,\cdot)$ and a random oracle, so its advantage is bounded by $v\varepsilon h(\lambda + sz(\lambda))$. ◀

By the triangle inequality we also get immediately that one cannot distinguish between keys of different functions:

▶ **Corollary 16.** *Let $i \in \{0,1\}$. There is a polynomial q such that for any $\varepsilon = \varepsilon(\lambda)$, any algorithm with oracle which makes $O(\varepsilon 2^\lambda)$ oracle queries $A^{\mathcal{O}}(1^\lambda, k)$ and any $f, g \in \mathcal{F}_n$,*

$$\left|\Pr[A^{\mathcal{O}_1}(1^\lambda, \mathsf{Gen}^{\mathcal{O}_1}(1^\lambda, f)_i) \mid \mathcal{O}_1 \leftarrow U_\lambda^O] - \Pr[A^{\mathcal{O}_2}(1^\lambda, \mathsf{Gen}^{\mathcal{O}_2}(1^\lambda, g)) \mid \mathcal{O}_2 \leftarrow U_\lambda^O]\right| \leq q(\lambda)\varepsilon^{1/c}$$

We now prove the main technical result behind the construction of the data structure: the *FSS side-switching lemma*. Informally, this lemma states that for any function $f \in \mathcal{F}_n$, given a random oracle and a key k_i for it, after observing only $2^n t_e$ oracle locations, we can produce a modification of t_g locations in the random oracle and a second key k_{1-i}, such that:

1. The modification does not change the evaluation of k_i at all.
2. After applying the modification, (k_0, k_1) are a valid pair of FSS keys for f with respect to the modified oracle.
3. The distribution of the oracle after applying the modification together with k_{1-i} is hard to distinguish from that of a random oracle together with a key for it.

The first two properties are baked into the algorithm, and the third is the content of Lemma 17.

Algorithm 1 FSS side-switching algorithm.

FSS side-switching
$\mathrm{KS}^{\mathcal{O}}(1^\lambda, i, k_i, f)$:

1: Evaluate $\mathrm{Obs}^{\mathcal{O}}(1^\lambda, i, k_i)$ by simulating $\mathsf{EvalAll}$ and keeping track of all oracle accesses.
2: Sample $\mathcal{O}_2 \leftarrow U_\lambda^O$, $(k'_0, k'_1, tr) \leftarrow \mathsf{Gen}'^{\mathcal{O}_2}(1^\lambda, f)$ conditioned on $k'_i = k_i$ and $\mathrm{Obs}^{\mathcal{O}}(1^\lambda, i, k_i) = \mathrm{Obs}^{\mathcal{O}_2}(1^\lambda, i, k_i)$. If this is impossible, return $\perp$. ▷ This step may be implemented inefficiently, in time exponential in the number of random bits used.
3: Define $\mathcal{O}_3(x) := \begin{cases} tr(x) & tr(x) \text{ exists}, \\ \mathcal{O}(x) & \text{otherwise.} \end{cases}$
4: **return** $(\mathcal{O}_3, k'_{1-i})$

▶ **Lemma 17** (FSS side-switching correctness)**.** *Let $i \in \{0,1\}$. Then there is a polynomial q such that for any $\varepsilon = \varepsilon(\lambda)$, any algorithm with oracle which makes $O(\varepsilon 2^\lambda)$ oracle queries $A^{\mathcal{O}}(1^\lambda, k)$ and any $f, g \in \mathcal{F}_n$,*

$$|\Pr[A^{\mathcal{O}_3}(1^\lambda, k_{1-i}) \mid \mathcal{O}_1 \leftarrow U_\lambda^O, k_i \leftarrow \mathsf{Gen}^{\mathcal{O}_1}(1^\lambda, g)_i, (\mathcal{O}_3, k_{1-i}) \leftarrow \mathrm{KS}^{\mathcal{O}_1}(1^\lambda, i, k_i, f)] - \Pr[A^{\mathcal{O}_2}(1^\lambda, \mathsf{Gen}^{\mathcal{O}_2}(1^\lambda, f)_{1-i}) \mid \mathcal{O}_2 \leftarrow U_\lambda^O]| \leq q(\lambda)(\varepsilon + t_e 2^{n-\lambda})^{1/c}$$

For KS *from Algorithm 1.*

Proof. We shall argue this result by showing a sequence of hybrid distributions of $(\mathcal{O}_3, k_{i-1})$ with the first one being $(\mathcal{O}_3, k_{1-i}) \leftarrow \mathrm{KS}^{\mathcal{O}_1}(1^\lambda, i, k_i, f)$, and the last one being $\mathcal{O}_3 \leftarrow U_\lambda^O, k_{1-i} \leftarrow \mathsf{Gen}^{\mathcal{O}_3}(1^\lambda, f)_{1-i}$. We will show that A cannot distinguish between any pair of adjacent distributions better than $\mathrm{poly}(\lambda)(\varepsilon + t_e 2^{n-\lambda})^{1/c}$, and the result will follow by the triangle inequality.

Let us start by writing the first distribution in more detail:

1. Sample $\mathcal{O}_1 \leftarrow U_\lambda^O$
2. Sample $k_i \leftarrow \mathsf{Gen}^{\mathcal{O}_1}(1^\lambda, g)_i$
3. Sample $\mathcal{O}_3, k_{1-i} \leftarrow \mathrm{KS}^{\mathcal{O}_1}(1^\lambda, 1, k_i, f)$

Because KS only queries $2^n t_e$ locations of $\mathcal{O}_1$ to calculate Obs, and then any query to $\mathcal{O}_3$ requires at most one query to $\mathcal{O}_1$, we have by Lemma 15 that A cannot distinguish with probability greater than $\mathrm{poly}(\lambda)(\varepsilon + t_e 2^{n-\lambda})^{1/c}$ between this distribution and

1. Sample $\mathcal{O}_1 \leftarrow U_\lambda^O$
2. Sample $k_i \leftarrow \mathsf{Sim}_i(1^\lambda)$
3. Sample $\mathcal{O}_3, k_{1-i} \leftarrow \mathrm{KS}^{\mathcal{O}_1}(1^\lambda, 1, k_i, f)$

Because steps 1,2 are independent, we can swap them. We also add a step sampling $\mathcal{O}_2$ which we don't use yet:

1. Sample $k_i \leftarrow \mathsf{Sim}_i(1^\lambda)$
2. Sample $\mathcal{O}_1 \leftarrow U_\lambda^O$
3. Sample $\mathcal{O}_2 \leftarrow U_\lambda^O$ conditioned on $\mathrm{Obs}^{\mathcal{O}_1}(1^\lambda, i, k_i) = \mathrm{Obs}^{\mathcal{O}_2}(1^\lambda, i, k_i)$
4. Sample $\mathcal{O}_3, k_{1-i} \leftarrow \mathrm{KS}^{\mathcal{O}_1}(1^\lambda, 1, k_i, f)$

Because 2 and 3 sample a pair of values from the same distribution conditioned on some function of them being equal, we obtain the same joint distribution by first sampling $\mathcal{O}_2$ uniformly and then sampling $\mathcal{O}_1$ conditioned $\mathrm{Obs}^{\mathcal{O}_1}(1^\lambda, i, k_i) = \mathrm{Obs}^{\mathcal{O}_2}(1^\lambda, i, k_i)$. We then swap the sampling of $\mathcal{O}_2$ and k_i, as they are independent:

1. Sample $\mathcal{O}_2 \leftarrow U_\lambda^O$
2. Sample $k_i \leftarrow \mathsf{Sim}_i(1^\lambda)$
3. Sample $\mathcal{O}_1 \leftarrow U_\lambda^O$ conditioned on $\mathrm{Obs}^{\mathcal{O}_1}(1^\lambda, i, k_i) = \mathrm{Obs}^{\mathcal{O}_2}(1^\lambda, i, k_i)$
4. Sample $\mathcal{O}_3, k_{1-i} \leftarrow \mathrm{KS}^{\mathcal{O}_1}(1^\lambda, 1, k_i, f)$

Note that step 3 queries $2^n t_e$ values of $\mathcal{O}_2$, and step 4 doesn't use $\mathcal{O}_2$, we can apply Lemma 15 again to see that A cannot distinguish better than $\text{poly}(\lambda)(t_e 2^{n-\lambda})^{1/c}$ between the previous distribution and:

1. Sample $\mathcal{O}_2 \leftarrow U_\lambda^O$
2. Sample $k_i \leftarrow \mathsf{Gen}^{\mathcal{O}_2}(1^\lambda, f)_i$
3. Sample $\mathcal{O}_1 \leftarrow U_\lambda^O$ conditioned on $\text{Obs}^{\mathcal{O}_1}(1^\lambda, i, k_i) = \text{Obs}^{\mathcal{O}_2}(1^\lambda, i, k_i)$
4. Sample $\mathcal{O}_3, k_{1-i} \leftarrow \text{KS}^{\mathcal{O}_1}(1^\lambda, 1, k_i, f)$

For here we will apply a long sequence of information-theoretical transformation, which don't change the distribution at all. To begin with, we only care about $\text{Obs}^{\mathcal{O}_2}(1^\lambda, i, k_i)$ so we can fold it into the sampling in step 2:

1. Sample $k_i \leftarrow \mathsf{Gen}^{\mathcal{O}}(1^\lambda, f)_i, o \leftarrow \text{Obs}^{\mathcal{O}}(1^\lambda, i, k_i)$ for a random oracle $\mathcal{O}$.
2. Sample $\mathcal{O}_1 \leftarrow U_\lambda^O$ conditioned on $\text{Obs}^{\mathcal{O}_1}(1^\lambda, i, k_i) = o$
3. Sample $\mathcal{O}_3, k_{1-i} \leftarrow \text{KS}^{\mathcal{O}_1}(1^\lambda, 1, k_i, f)$

Now let us unfold the definition of KS:

1. Sample $k_i \leftarrow \mathsf{Gen}^{\mathcal{O}}(1^\lambda, f)_i, o \leftarrow \text{Obs}^{\mathcal{O}}(1^\lambda, i, k_i)$ for a random oracle $\mathcal{O}$.
2. Sample $\mathcal{O}_1 \leftarrow U_\lambda^O$ conditioned on $\text{Obs}^{\mathcal{O}_1}(1^\lambda, i, k_i) = o$
3. Sample $\mathcal{O}_2 \leftarrow U_\lambda^O$ and $k'_0, k'_1, tr \leftarrow \mathsf{Gen}'^{\mathcal{O}_2}(1^\lambda, f)$ conditioned on $k'_i = k_i$ and $\text{Obs}^{\mathcal{O}_2}(1^\lambda, i, k_i) = o$
4. Define $\mathcal{O}_3(x) := \begin{cases} tr(x) & tr(x) \text{ exists}, \\ \mathcal{O}_1(x) & \text{otherwise}. \end{cases}$

We can swap 2 and 3 as they are independent:

1. Sample $k_i \leftarrow \mathsf{Gen}^{\mathcal{O}}(1^\lambda, f)_i, o \leftarrow \text{Obs}^{\mathcal{O}}(1^\lambda, i, k_i)$ for a random oracle $\mathcal{O}$.
2. Sample $\mathcal{O}_2 \leftarrow U_\lambda^O$ and $k'_i, k'_1, tr \leftarrow \mathsf{Gen}'^{\mathcal{O}_2}(1^\lambda, f)$ conditioned on $k'_i = k_i$ and $\text{Obs}^{\mathcal{O}_2}(1^\lambda, i, k_i) = o$
3. Sample $\mathcal{O}_1 \leftarrow U_\lambda^O$ conditioned on $\text{Obs}^{\mathcal{O}_1}(1^\lambda, i, k_i) = o$
4. Define $\mathcal{O}_3(x) := \begin{cases} tr(x) & tr(x) \text{ exists}, \\ \mathcal{O}_1(x) & \text{otherwise}. \end{cases}$

And then combine 3 and 4 together:

1. Sample $k_i \leftarrow \mathsf{Gen}^{\mathcal{O}}(1^\lambda, f)_i, o \leftarrow \text{Obs}^{\mathcal{O}}(1^\lambda, i, k_i)$ for a random oracle $\mathcal{O}$.
2. Sample $\mathcal{O}_2 \leftarrow U_\lambda^O$ and $k'_0, k'_1, tr \leftarrow \mathsf{Gen}'^{\mathcal{O}_2}(1^\lambda, f)$ conditioned on $k'_i = k_i$ and $\text{Obs}^{\mathcal{O}_2}(1^\lambda, i, k_i) = o$
3. Sample $\mathcal{O}_3(x) := \begin{cases} tr(x) & tr(x) \text{ exists}, \\ o(x) & o(x) \text{ exists}, \\ U_\lambda & \text{otherwise}. \end{cases}$

We will now merge steps 1 and 2 together. We can think of them together as sampling from quintuples $(\mathcal{O}, k_0, k_1, tr, o)$. The first step samples k_i, o from the correct marginal distribution, and then the second step samples the rest of the value conditioned on the values sampled in the previous step, so we can merge them to sample a quintuple directly. Because $\mathcal{O}_2$ is not used in step 3 we will remove it and only sample (k_0, k_1, tr, o):

1. Sample $k_0, k_1, tr \leftarrow \mathsf{Gen}'^{\mathcal{O}}(1^\lambda, f)$, $o \leftarrow \text{Obs}^{\mathcal{O}}(1^\lambda, i, k_i)$ for a random oracle $\mathcal{O}$.
2. Sample $\mathcal{O}_3(x) := \begin{cases} tr(x) & tr(x) \text{ exists}, \\ o(x) & o(x) \text{ exists}, \\ U_\lambda & \text{otherwise}. \end{cases}$

Finally, we can regard these two steps again as sampling from quintuples $(\mathcal{O}, k_0, k_1, tr, o)$. The first step samples (k_0, k_1, tr, o) from their marginal distribution, and we see that $\mathcal{O}_3$ is sampled from the correct distribution, that is all oracles which agree with tr, o are equally likely, because no relevant execution of $\mathsf{Gen}, \mathrm{Obs}$ can query any locations outside of tr, o (precisely by the definition of tr and o). This means that we simply sample from the quintuples, which we can also do by first sampling the oracle. This gives the desired distribution:

1. Sample $\mathcal{O}_3(x) \leftarrow U_\lambda^O$
2. Sample $k_0, k_1, tr \leftarrow \mathsf{Gen}'^{\mathcal{O}_3}(1^\lambda, f)$ ◀

For the data structure itself we won't use the full KS, but instead a reduced version KS-Red. Instead of returning the full oracle, the reduced version will return the changed locations tr.

The data structure mostly functions by using the side-switching algorithm to do a small change to the oracle and key and change the result of the evaluation by the function f.

FSS data structure
$\mathcal{O} : \{0,1\}^\lambda \to \{0,1\}^\lambda$: oracle
k_0: key, represented using t_k λ-bit cells
$\mathcal{O}_{init} : \{0,1\}^\lambda \to \{0,1\}^\lambda$: initial oracle
k_{init}: initial key, represented using t_k λ-bit cells

$\mathsf{Init}()$:
1: Sample $\mathcal{O}_{init} \leftarrow U_\lambda^O$
2: Set $\mathcal{O} = \mathcal{O}_{init}$
3: Sample $k_{init} \leftarrow \mathsf{Gen}^{\mathcal{O}_{init}}(1^\lambda, \mathbf{0})_0$
4: Set $k_0 = k_{init}$

$\mathsf{Update}(f)$:
1: Sample $(tr, k_1) \leftarrow \text{KS-Red}^{\mathcal{O}}(1^\lambda, 0, k_0, \mathbf{0})$. If it returns $\perp$ **return** $\perp$
2: **for** $x \in \mathrm{Dom}(tr)$ **do**
3: Update $\mathcal{O}(x) \leftarrow tr(x)$
4: **end for**
5: Sample $(tr_2, k_0') \leftarrow \text{KS-Red}^{\mathcal{O}}(1^\lambda, 1, k_1, f)$. If it returns $\perp$ **return** $\perp$
6: **for** $x \in \mathrm{Dom}(tr_2)$ **do**
7: Update $\mathcal{O}(x) \leftarrow tr_2(x)$
8: **end for**
9: Update $k_0 \leftarrow k_0'$

$\mathsf{Query}(x)$:
1: **return** $\mathsf{Eval}^{\mathcal{O}}(1^\lambda, 0, k_0, x) - \mathsf{Eval}^{\mathcal{O}_{init}}(1^\lambda, 0, k_{init}, x)$

Figure 1 FSS write probe model algorithm.

▶ **Theorem 18.** *Let $\mathcal{F}$ be a function family with an* FSS *with key size $t_k = t_k(\lambda, n)$, generation time $t_g = t_g(\lambda, n)$ and evaluation time $t_e = t_e(\lambda, n)$ for input size n and security parameter $\lambda = \lambda(n)$ which is fully black-box reducible to PRF. Let m be the number of desired queries, and suppose $\lambda \geq n + \log(mt_e) + c(\log(m) + \kappa)$*

Then Algorithm 1 is a data structure for $\mathcal{F}$'s range counting problem in the write probe model with worst case $O(t_k + t_g)$ writes per update and $O(t_e)$ probes per query with λ-bit cells which is correct for m updates with probability $\geq 1 - \mathrm{poly}(\lambda)2^{-\kappa}$.

Proof. We assume WLOG that 0 is one of the functions in $\mathcal{F}$ using Section 3.2 of [11].

If the data structure doesn't abort during an update, we show that the queries will be answered correctly: Call the initial oracle $\mathcal{O}$, the oracle after step 4 $\mathcal{O}'$, and the oracle after step 8 $\mathcal{O}''$. Since $\mathrm{Obs}^{\mathcal{O}}(1^\lambda, 0, k_0) = \mathrm{Obs}^{\mathcal{O}'}(1^\lambda, 0, k_0)$ by the definition of KS, we have

$\mathsf{EvalAll}^{\mathcal{O}'}(1^\lambda, 0, k_0) = \mathsf{EvalAll}^{\mathcal{O}}(1^\lambda, 0, k_0)$, and because (k_0, k_1) come from $\mathsf{Gen}(1^\lambda, 0)$ with tr, and so are a possible output of Gen for $\mathcal{O}'$, we have $\mathsf{EvalAll}^{\mathcal{O}'}(1^\lambda, 0, k_0) + \mathsf{EvalAll}^{\mathcal{O}'}(1^\lambda, 1, k_1) = \mathbf{0}$. Combining the last two equations we have

$$\mathsf{EvalAll}^{\mathcal{O}'}(1^\lambda, 1, k_1) = -\mathsf{EvalAll}^{\mathcal{O}}(1^\lambda, 0, k_0).$$

Similarly for $\mathcal{O}''$ we have $\mathsf{EvalAll}^{\mathcal{O}''}(1^\lambda, 1, k_1) = \mathsf{EvalAll}^{\mathcal{O}'}(1^\lambda, 1, k_1)$, and $\mathsf{EvalAll}^{\mathcal{O}''}(1^\lambda, 0, k_0') + \mathsf{EvalAll}^{\mathcal{O}''}(1^\lambda, 1, k_1) = f$, so

$$\mathsf{EvalAll}^{\mathcal{O}''}(1^\lambda, 0, k_0') = f - \mathsf{EvalAll}^{\mathcal{O}'}(1^\lambda, 1, k_1) = f + \mathsf{EvalAll}^{\mathcal{O}}(1^\lambda, 0, k_0).$$

This shows that $\mathsf{EvalAll}^{\mathcal{O}}(1^\lambda, 0, k_0)$ increases by f. In Query we output the x-th element of $\mathsf{EvalAll}^{\mathcal{O}}(1^\lambda, 0, k_0) - \mathsf{EvalAll}^{\mathcal{O}_{init}}(1^\lambda, 0, k_{init})$, which starts at 0 and is increased by f with $\mathsf{EvalAll}^{\mathcal{O}}(1^\lambda, 0, k_0)$.

Left to show is an upper bound on the probability of returning $\perp$ during an update sequence. We will ignore $\mathcal{O}_{init}$ and k_{init} here, as these are only used for answering queries.

We will use a sequence of $3m + 1$ hybrids to bound the success probability for an update sequence $f_1, f_2, \ldots, f_m$, defined in Figure 2:

Hybrid number $3i - 2$ for $i = 1, 2, \ldots, m, m + 1$:
1: Sample $\mathcal{O} \leftarrow U_\lambda^O$
2: Sample $k_0 \leftarrow \mathsf{Gen}^{\mathcal{O}}(1^\lambda, \mathbf{0})_0$
3: **for** $i \le j \le m$ **do**
4: Apply $\mathsf{Update}(f_j)$ to the data structure $(\mathcal{O}, k_0)$. If it returns $\perp$ **return** False
5: **end for**
6: **return** True

Hybrid number $3i - 1$ for $i = 1, 2, \ldots, m$:
1: Sample $\mathcal{O} \leftarrow U_\lambda^O$
2: Sample $k_1 \leftarrow \mathsf{Gen}^{\mathcal{O}}(1^\lambda, \mathbf{0})_1$
3: Sample $(tr_2, k_0') \leftarrow \text{KS-Red}^{\mathcal{O}}(1^\lambda, 1, k_1, f_i)$. If KS returns $\perp$ **return** $\perp$
4: **for** $x \in \mathrm{Dom}(tr_2)$ **do**
5: Update $\mathcal{O}(x) \leftarrow tr_2(x)$
6: **end for**
7: Set $k_0 \leftarrow k_0'$
8: **for** $i < j \le m$ **do**
9: Apply $\mathsf{Update}(f_j)$ to the data structure $(\mathcal{O}, k_0)$. If it returns $\perp$ **return** False
10: **end for**
11: **return** True

Hybrid number $3i$ for $i = 1, 2, \ldots, m$:
1: Sample $\mathcal{O} \leftarrow U_\lambda^O$
2: Sample $k_0 \leftarrow \mathsf{Gen}^{\mathcal{O}}(1^\lambda, f)_0$
3: **for** $i < j \le m$ **do**
4: Apply $\mathsf{Update}(f_j)$ to the data structure $(\mathcal{O}, k_0)$. If it returns $\perp$ **return** False
5: **end for**
6: **return** True

Figure 2 Hybrids number $3i - 2, 3i - 1, 3i$.

Consider the difference in probability between hybrid $3i - 2$ and $3i - 1$. We unfold $\mathsf{Update}(f_i)$ in hybrid $3i - 2$ and use Lemma 17. The number of oracle queries made by the rest of the hybrid is $O(m2^n t_e)$, so by the lemma the distinguishing probability is bounded by $\mathrm{poly}(\lambda)(m2^{n-\lambda}t_e + 2^{n-\lambda}t_e)^{\frac{1}{c}} = \mathrm{poly}(\lambda)2^{-\kappa}/m$.

Now consider the difference between $3i-1$ and $3i$. Again we can apply Lemma 17, and get a bound on the difference of $\text{poly}(\lambda)2^{-\kappa}/m$.

Finally, for the difference between hybrid number $3i$ and hybrid $3i+1 = 3(i+1)-2$ we can apply Corollary 16 to get a bound of $\text{poly}(\lambda)2^{-\kappa}/m$.

Now from the triangle inequality

$$|\Pr[\text{hybrid 1 outputs True}] - \Pr[\text{hybrid } 3m+1 \text{ outputs True}]| \leq$$
$$\sum_{i=1}^{3m} |\Pr[\text{hybrid } i \text{ outputs True}] - \Pr[\text{hybrid } i+1 \text{ outputs True}]| \leq$$
$$\sum_{i=1}^{3m} \text{poly}(\lambda)2^{-\kappa}/q \leq \text{poly}(\lambda)2^{-\kappa}.$$

And since $\Pr[\text{hybrid } 3m+1 \text{ outputs True}] = 1$ we are finished. ◀

Setting $m = 2^n$ and $\kappa = 3n$ we get error probability $o(\frac{1}{2^{2n}})$, so we apply Proposition 8 to get

▶ **Corollary 19.** *Let $\mathcal{F}$ be a function family with an* FSS *with key size $t_k = t_k(\lambda, n)$, generation time $t_g = t_g(\lambda, n)$ and evaluation time $t_e = t_e(\lambda, n)$ for input size n and security parameter λ which is fully black-box reducible to PRF. Suppose $\lambda \geq Cn$, for some constant C depending on the FSS and all functions in $\mathcal{F}_n$ can be described with* $\text{poly}(\lambda)$ *bits.*

Then there exists a data structure for $\mathcal{F}$'s range counting problem in the write probe model with worst case $O(t_k + t_g)$ writes per update and expected query time $O(t_e)$ which is correct for 2^n updates.

Specializing to use Corollary 10, we get

▶ **Corollary 20.** *Let $\mathcal{F}^2_{box}$ be the function family of functions which assign a constant group element β for points in a given 2D rectangle and 0 to other points, with a cyclic group $\log|\mathbb{G}| = (1+\varepsilon)n$, and suppose we have an FSS scheme fully black-box reducible to PRF where the key size and* Gen *time are polynomial for $\lambda = O(n)$, then the efficiency of* Eval *must be $t_e = \Omega((\frac{n}{\log n})^2)$.*

Proof. By Corollary 19 we have a data structure in the write probe model for the rectangle stabbing problem with weights in $\mathbb{G}$. Assuming $\log|\mathbb{G}| \geq n + \varepsilon n$, we can solve also the rectangle stabbing problem with εn-bit integer weights, because the result fits in $\mathbb{G}$. The bound now follows from Corollary 10. ◀

▶ Remark 21. The above bound also holds in any constant dimension larger than 2, as we can simply solve the 2D problem in the first two dimensions and ignore the rest.

Alternatively, specializing to apply Corollary 14:

▶ **Corollary 22.** *Let $\mathcal{F}^2_{box}$ be the function family of functions which assigns a 1 for points in a given 2D rectangle and 0 to other points, with output in $\mathbb{F}_2$, and suppose we have an FSS scheme fully black-box reducible to PRF where the key size and* Gen *time are polynomial for $\lambda = O(n)$, then the efficiency of* Eval *must be $t_e = \Omega(\frac{n^{3/2}}{\log^3 n})$.*

And lastly, applying Theorem 12 for $B = n^{\omega(1)}$, $B \leq 2^{n/4}$,

▶ **Corollary 23.** *Let $\mathcal{F}_{mono}$ be the function family of all monomials $ax^b \in \mathbb{F}_{2^n}[X]$ such that $b \leq B$, and suppose we have an FSS scheme fully black-box reducible to PRF where the key size and* Gen *time are polynomial for $\lambda = O(n)$, then the efficiency of* Eval *must be $t_e = \Omega(\frac{n\sqrt{\log(B)}}{\log^2 n})$.*

5 Data Structure to FSS

We also have a reduction from a data structure to an FSS secure from OWF, assuming the data structure is in the oblivious group model. Recall that a data structure in the oblivious group model can be represented as a factorization of the matrix $M \in \mathbb{Z}^{|\mathcal{F}|\times 2^n}$ representing the function family to an update matrix $\mathcal{U} \in \mathbb{Z}^{|\mathcal{F}|\times S}$ and a query matrix $\mathcal{Q} \in \mathbb{Z}^{S\times 2^n}$, such that each row of $\mathcal{U}$ has at most t_u non-zero values, and each column of $\mathcal{Q}$ has at most t_q non-zero values.

The reduction will use the DMPF to share the appropriate row of $\mathcal{U}$, and during Eval will run Eval on the DMPF based on the column of the $\mathcal{Q}$ corresponding to the query, computing $\sum_{\mathcal{Q}_{j,x}\neq 0} \mathcal{Q}_{j,x}\mathsf{DMPF.Eval}(1^\lambda, i, k_i, j)$. The sum of the two shares will give $\sum_{\mathcal{Q}_{j,x}\neq 0} \mathcal{Q}_{j,x}\mathcal{U}_{f,j} = (\mathcal{U}\mathcal{Q})_{f,x} = M_{f,x} = f(x)$.

This reduction using plain DMPF incurs, for many problems, an extra factor of n compared to building the FSS directly. The source of this factor is that DMPF costs $O(nB\lambda)$ for storing B points with n bits. We can use a tree memory model to alleviate this problem, where we have an infinite binary tree with values in the nodes, and we can access these values by walking in the tree. This memory model can be implemented more efficiently by FSS constructions based on the GGM tree [13]. Simulating random access with S cells in this memory structure (like what's done by DMPF) inherently costs $\log(S)$ per operation, but for some problems we can use the tree memory better directly.

For example, consider the problem of prefix DPF, where the function we want to share is 1 for all prefixes of some binary string x and 0 otherwise. Using the tree memory model, we can just write ones on the path from the root to x, and in evaluation access the node corresponding to the query point. This costs $O(n\lambda)$ for query and evaluation. If we use a flat structure instead, however, we still need to store n points, which will cost a key size of $O(n^2\lambda)$ using DMPF.

To abstract this, we define the *tree cost* for updates and queries in the oblivious group model.

For a given vector v, the tree size of it is $|v|_{tree} = \left|\left\{\left\lfloor \frac{i}{2^j}\right\rfloor \mid v_i \neq 0, j \in \mathbb{N}, \left\lfloor \frac{i}{2^j}\right\rfloor \neq 0\right\}\right|$. If we label the root 1, its children 2, 3, etc, $|v|_{tree}$ is the number of nodes we need to access to access all non-zero values of v.

We can now define the tree update cost t_u^{tree} and tree query cost t_q^{tree} of a data structure in the oblivious group model as the maximum tree size of a row in $\mathcal{U}$ and column in $\mathcal{Q}$ respectively.

We also define incremental DMPF, which is a variant of DMPF which can take advantage of this structure. This is a merge of DMPF and incremental DPF [6]. In incremental DPF Gen shares a single path in the infinite binary tree and values along it (this can also be thought of as sharing a binary string and a value for each prefix). There is an EvalNext function which takes the key and the current state as well as a single bit, and moves in that direction, returning the new state as well as the value stored in the current node.

For incremental DMPF Gen shares values on a binary tree, filling the rest of the infinite binary tree with zeros, instead of sharing only a single path in the infinite binary tree. EvalNext functions in the same way as in incremental DPF.

The formal construction of the incremental DMPF will be given in the full version of this paper and gives us the theorem:

▶ **Theorem 24.** *Let $(\mathcal{U}, \mathcal{Q})$ be a data structure for $\mathcal{F}$'s range searching problem in the oblivious group model. Then there exists an FSS for $\mathcal{F}$ with a length-doubling PRG oracle $\mathcal{O} : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$, key size $t_k = O(t_u^{tree})$ or $O(\lambda t_u^{tree})$ bits, $t_g = O(t_u^{tree})$ executions of $\mathcal{O}$ in Gen and $t_e = O(t_q^{tree})$ executions of $\mathcal{O}$ in Eval.*

From the trivial bounds $t_u^{tree} \leq t_u, t_q^{tree} \leq t_q$ and the lower bound $t_u t_q = \Omega(n^{d-1})$ for $\mathcal{F}_{box}^d$ [25], we get the corollary:

▶ **Corollary 25.** *If* (Gen, Eval) *is an FSS for* $\mathcal{F}_{box}^d$ *constructed using Theorem 24 with evaluation time* t_e *and key size* t_k *then* $t_k t_e = \Omega(n^{d-1})$.

References

1 Pankaj K Agarwal. Range searching. In *Handbook of discrete and computational geometry*, pages 1057–1092. Chapman and Hall/CRC, 2017. doi:10.1201/9781315119601.

2 Paul Baecher, Christina Brzuska, and Marc Fischlin. Notions of black-box reductions, revisited. In Kazue Sako and Palash Sarkar, editors, *Advances in Cryptology - ASIACRYPT 2013*, pages 296–315, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. doi:10.1007/978-3-642-42033-7_16.

3 Amos Beimel. Secret-sharing schemes: A survey. In *International conference on coding and cryptology*, pages 11–46. Springer, 2011. doi:10.1007/978-3-642-20901-7_2.

4 Jon Louis Bentley and Jerome H Friedman. Data structures for range searching. *ACM Computing Surveys (CSUR)*, 11(4):397–409, 1979. doi:10.1145/356789.356797.

5 Alexander Bienstock, Sarvar Patel, Joon Young Seo, and Kevin Yeo. Near-Optimal oblivious Key-Value stores for efficient PSI, PSU and Volume-Hiding Multi-Maps. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 301–318, Anaheim, CA, August 2023. USENIX Association. URL: https://www.usenix.org/conference/usenixsecurity23/presentation/bienstock.

6 Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. Lightweight techniques for private heavy hitters. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 762–776, 2021. doi:10.1109/SP40001.2021.00048.

7 Elette Boyle, Nishanth Chandran, Niv Gilboa, Divya Gupta, Yuval Ishai, Nishant Kumar, and Mayank Rathee. Function secret sharing for mixed-mode and fixed-point secure computation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 871–900, 2021. doi:10.1007/978-3-030-77886-6_30.

8 Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: Silent ot extension and more. In *Advances in Cryptology–CRYPTO 2019: 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2019, Proceedings, Part III 39*, pages 489–518. Springer, 2019. doi:10.1007/978-3-030-26954-8_16.

9 Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators from ring-lpn. In *Annual International Cryptology Conference*, pages 387–416. Springer, 2020. doi:10.1007/978-3-030-56880-1_14.

10 Elette Boyle, Niv Gilboa, Matan Hamilis, Yuval Ishai, and Yaxin Tu. Improved Constructions for Distributed Multi-Point Functions . In *2025 IEEE Symposium on Security and Privacy (SP)*, pages 2414–2432, Los Alamitos, CA, USA, May 2025. IEEE Computer Society. doi:10.1109/SP61157.2025.00044.

11 Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 337–367. Springer, 2015. doi:10.1007/978-3-662-46803-6_12.

12 Elette Boyle, Niv Gilboa, and Yuval Ishai. Breaking the circuit size barrier for secure computation under ddh. In *Annual International Cryptology Conference*, pages 509–539. Springer, 2016. doi:10.1007/978-3-662-53018-4_19.

13 Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 1292–1303. ACM, 2016. doi:10.1145/2976749.2978429.

14 Elette Boyle, Niv Gilboa, and Yuval Ishai. Secure computation with preprocessing via function secret sharing. In *Theory of Cryptography: 17th International Conference, TCC 2019, Nuremberg, Germany, December 1–5, 2019, Proceedings, Part I 17*, pages 341–371. Springer, 2019. doi:10.1007/978-3-030-36030-6_14.

15 Elette Boyle, Lisa Kohl, and Peter Scholl. Homomorphic secret sharing from lattices without fhe. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 3–33. Springer, 2019. doi:10.1007/978-3-030-17656-3_1.

16 Bernard Chazelle. Lower bounds for orthogonal range searching: I. the reporting case. *Journal of the ACM (JACM)*, 37(2):200–212, 1990. doi:10.1145/77600.77614.

17 Bernard Chazelle. Lower bounds for orthogonal range searching II. the arithmetic model. *J. ACM*, 37(3):439–463, July 1990. doi:10.1145/79147.79149.

18 Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private information retrieval. *Journal of the ACM (JACM)*, 45(6):965–981, 1998. doi:10.1145/293347.293350.

19 Henry Corrigan-Gibbs, Dan Boneh, and David Mazières. Riposte: An anonymous messaging system handling millions of users. In *2015 IEEE Symposium on Security and Privacy*, pages 321–338. IEEE, 2015. doi:10.1109/SP.2015.27.

20 Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Orthogonal Range Searching*, pages 95–120. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. doi:10.1007/978-3-540-77974-2_5.

21 Yevgeniy Dodis, Shai Halevi, Ron D Rothblum, and Daniel Wichs. Spooky encryption and its applications. In *Annual International Cryptology Conference*, pages 93–122. Springer, 2016. doi:10.1007/978-3-662-53015-3_4.

22 Michael Fredman and Michael Saks. The cell probe complexity of dynamic data structures. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 345–354, 1989. doi:10.1145/73007.73040.

23 Michael L Fredman. The complexity of maintaining an array and computing its partial sums. *Journal of the ACM (JACM)*, 29(1):250–260, 1982. doi:10.1145/322290.322305.

24 Kasper Green Larsen. The cell probe complexity of dynamic range counting. In Howard J. Karloff and Toniann Pitassi, editors, *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, STOC '12, pages 85–94, New York, NY, USA, 2012. ACM. doi:10.1145/2213977.2213987.

25 Kasper Green Larsen. On range searching in the group model and combinatorial discrepancy. *SIAM Journal on Computing*, 43(2):673–686, 2014. doi:10.1137/120865240.

26 Kasper Green Larsen, Omri Weinstein, and Huacheng Yu. Crossing the logarithmic barrier for dynamic boolean data structure lower bounds. *SIAM Journal on Computing*, 49(5):STOC18–323–STOC18–367, 2020. doi:10.1137/18M1198429.

27 Zhe Li, Chaoping Xing, Yizhou Yao, and Chen Yuan. Efficient pseudorandom correlation generators for any finite field. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 145–175. Springer, 2025. doi:10.1007/978-3-031-91092-0_6.

28 Claudio Orlandi, Peter Scholl, and Sophia Yakoubov. The rise of Paillier: Homomorphic secret sharing and public-key silent ot. In *Advances in Cryptology–EUROCRYPT 2021: 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17–21, 2021, Proceedings, Part I 40*, pages 678–708. Springer, 2021. doi:10.1007/978-3-030-77870-5_24.

29 Mihai Patrascu. Lower bounds for 2-dimensional range counting. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 40–46, 2007. doi:10.1145/1250790.1250797.

30 Mihai Patrascu and Erik D Demaine. Logarithmic lower bounds in the cell-probe model. *SIAM Journal on Computing*, 35(4):932–963, 2006. doi:10.1137/S0097539705447256.

31 Omer Reingold, Luca Trevisan, and Salil P. Vadhan. Notions of reducibility between cryptographic primitives. In *Theory of Cryptography, First Theory of Cryptography Conference, TCC 2004, Cambridge, MA, USA, February 19-21, 2004, Proceedings*, volume 2951 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2004. doi:10.1007/978-3-540-24638-1_1.

32 Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979. doi:10.1145/359168.359176.

33 Andrew Chi-Chih Yao. Should tables be sorted? *Journal of the ACM (JACM)*, 28(3):615–628, 1981. doi:10.1145/322261.322274.