SLAMP-FSS: Two-Party Multi-Point Function Secret Sharing from Simple Linear Algebra

Erki Külaots[1], Toomas Krips[1], Hendrik Eerikson[2], Darya Pisetskaya[1] and Pille Pullonen-Raudvere[2]

[1] University of Tartu, Institute of Computer Science, Tartu, Estonia
[2] Cybernetica AS, Information Security Research Institute, Tallinn, Estonia

Abstract. Multi-point function secret sharing (FSS) is a building block for pseudorandom correlation generators used in novel silent correlation generation methods for various secure multi-party computation applications. However, the main construction used so far is the naive approach to combining several point functions. In this paper, we propose an efficient and natural generalisation of the point function FSS scheme of Boyle et al. 2016 [BGI16] using a tree structure, a pseudorandom generator and systems of linear equations. We propose a new notion of distributed random multi-point function. Our construction splits the distributed multi-point function scheme into a random multi-point function scheme and an algorithm to transform a random output into the desired output value. The resulting scheme, which we call SLAMP-FSS, improves upon the state of the art in terms of calls to a pseudorandom generator (PRG).

Keywords: Function secret sharing ; Secure multi-party computation ; Distributed multi-point function.

1 Versions of SLAMP-FSS

We note that the main algorithm given in this version of the paper is different from the algorithm given in the initial version in the ePrint [KKEPR24]. There are two key differences. First, in the previous version, the pseudorandom function with input k output $v + 1$ values of length k. Previous reviewers pointed out that it is unfair to compare such a PRF with a PRF that doubles the length of the input. Thus in the current version, the length of the output of the pseudorandom function in this paper is $v + k$, which is more similar to a doubling-the-length PRF, see later in the paper for discussion. In addition, this allows us to replace solving systems of linear equations with coefficients from $\mathbb{F}_{2^k}$ by solving systems of linear equations with coefficients from $\mathbb{F}_2$ which is much more efficient.

Secondly, in the initial version of the paper we gave two versions of the main algorithm, where the second version was more efficient but gave randomized outputs. In this version we give one algorithm, but we modularize it. The first part of the algorithm gives an output that is zero everywhere except for the predetermined points, where the generating phase chooses those values randomly. These random values are passed on to the second phase of the algorithm, which compiles it to a DMPF with desired outputs. The second part is also universal in that it compiles any distributed-random-output-multipoint-function to a distributed-multipoint-function with desired outputs.

E-mail: erki.kulaots@ut.ee (Erki Külaots), toomas.krips@ut.ee (Toomas Krips), hendrik.eerikson@cyber.ee (Hendrik Eerikson), darya.pisetskaya@ut.ee (Darya Pisetskaya), pille.pullonen-raudvere@cyber.ee (Pille Pullonen-Raudvere)

This work is licensed under a "CC BY 4.0" license.
Date of this document: 2026-05-05.

In this version, we have added information about a prototype implementation and cleared up the writing. This version is the same as the one published in IACR Communications in Cryptology Volume 3, Issue 1 [KKE+26].

2 Introduction

Function secret sharing scheme (FSS) [BGI15] is a cryptographic tool for sharing a private function between parties so that the function remains unknown, but the parties can collectively evaluate the function. Moreover, the evaluation itself does not require communication. FSS schemes have been proposed for various classes of functions, including point functions [GI14] and multi-point functions [dCP22]. Point functions are functions that evaluate to a predetermined value β at the predetermined special point α and evaluate to 0 everywhere else. Multi-point functions have some constant t number of nonzero points. This paper proposes an FSS scheme for multi-point functions using a tree structure that has been previously utilised for point functions. Note that a two-party FSS scheme for (multi-)point functions is also known as *distributed (multi-)point function* (DPF or DMPF, respectively). These classes of functions are of specific interest as FSS schemes for them are used to compress additive secret sharing of vectors with 1 or t nonzero elements. This results in a communication-efficient way to securely share sparse vectors.

FSS was initially proposed in [BGI15] for querying and updating distributed databases. In general, FSS definition applies to all kinds of functions, but mainly efficient constructions are known for point functions. FSS for point functions has been used for contact tracing application [DIL+20], ORAM [DS17, BKKO20], anonymous messaging [CBM15, ECZB21, NSD22], implementing non-linear functions using look-up tables in privacy-preserving machine learning [Wag22, GJM+24], secure search [FLO19, DFL+20, DRPS22], comparison functions [SVLH23, GGM24] and private subset-histograms [BBC+21]. The notion can also be extended to the case of programmable distributed point functions [BGIK22] where one key can be used to share many point functions. There are also notions of verifiable distributed point functions [BBC+21, dCP22] where parties can check that their share is correct and distributed point functions with constraints [JZZR24] where parties can check that the dealer was authorised to share this function. Multi-point FSS can be used in the constructions of vector oblivious linear-function evaluation (VOLE) [BCGI18], random oblivious transfer [BCG+19], comparison functions [KPM24], private set intersection [Zha23], and pseudorandom correlation generators (PCGs) [BCG+19]. Correlated randomness is a precious resource in cryptographic applications. For example, identical random strings distributed among the parties can be used for perfectly secure encryption by using one-time pad encryption. Other examples of more useful types of random correlation with respect to MPC are random oblivious transfer (OT) correlations, oblivious linear-function correlations (OLE), multiplication triples (also known as Beaver triples [Bea91]) and one-time truth tables [BCGI18, BCG+19].

2.1 Our contributions

We propose a new scheme for multi-point FSS. Such schemes consist of key generation and evaluation algorithms. Key generation takes the function description and gives the keys (or shares) for the two parties. During evaluation, both parties use their keys and a common input to locally compute a share of the function output. Often, the function is evaluated at all possible inputs, this is called *full evaluation*. Previous schemes rely on many pseudorandom generator (PRG) evaluations which are the most expensive part of the protocol, and a large number of PRG calls are needed in the protocol. Therefore, our goal is to reduce the number of PRG calls in the evaluation. Our main scheme is called Simple Linear Algebra Multi-Point Function Secret Sharing (SLAMP-FSS). SLAMP-FSS has

similar key size and time complexity to other multi-point FSS schemes and its advantages are simple mathematical constructions and efficiency of evaluating the shared function at some small constant number of points. More comparison appears in Section 7.

For constructing PCGs, commonly, t applications of a distributed point functions are needed. For PCGs based on Learning Parity with Noise (LPN), the t depends on the amount of noise needed for the LPN assumption to hold. The goal is to have a small amount of noise while still ensuring that the assumption holds, for example, in [BCG+19] for length 2^{16} the $t = 32$ for 80-bit security. Hence, t is about the approximate order of magnitude of $2^5 - 2^{10}$. So far, the most common approach to full evaluation of multi-point functions has been using the distributed point function from [BGI16] as a black box and using various approaches, for example, using combinatorial objects such as batch codes [BCGI18] and probabilistic batch codes [ACLS18, SGRR19] to reduce the total evaluation cost. However, batch codes are technically complicated and obtaining good specific parameters is difficult, and modifications such as probabilistic batch codes have their own trade-offs.

We propose a natural generalisation of [BGI16] for multi-point functions. SLAMP-FSS uses simple linear algebra and is conceptually simple. Our construction is based on a binary tree with n levels and the bit representation of the input value is used to traverse the tree and the leaves contain the output values. The tree has t nonzero leaves, we call these *active* leaves and nodes on the paths leading to them from the root as *active* nodes; the other nodes we call *inactive* nodes. Evaluating at point x corresponds to traversing the tree to the corresponding leaf. The parties have a vector of secret-shared values with the invariant that the values there are zero if and only if we are currently at an inactive node. Thus, it is important to construct SLAMP-FSS in such a way that if a node is inactive, its children will automatically be inactive. There can be at most t active nodes at any level of the binary tree. Active nodes of a layer will have twice the number of children, some of which will be active and some will be inactive. We use simple linear algebra to ensure that the invariant mentioned before will be satisfied at all levels.

We modularize our approach by first building a scheme that outputs random values at the predetermined inputs (Simple Linear Algebra Multi-Point Random Function Secret Sharing a.k.a SLAMPR-FSS), and then an algorithm that compiles a random-output scheme into a multi-point FSS. This algorithm can compile any random-output scheme to a multi-point FSS scheme with a fixed output value and is of independent interest, as well as being an integral part of SLAMP-FSS. We note that SLAMP-FSS can be used to evaluate the distributed point function on any number of points, including single input or full evaluation. It is most efficient for a moderate number of points, such as $t \leq 64$ and could be used in some PCG constructions.

We implement and benchmark a prototype of SLAMP-FSS for a concrete set of parameters and compare our scheme with the state of the art. We additionally provide benchmarks for the primitive operations that compose SLAMP-FSS. The source code of the prototype and benchmarks has been published on GitHub[1].

2.2 Paper Outline

In Section 3, we give a short description of prior work regarding distributed multi-point functions. Section 4 lists the notation we use and the underlying definitions required by the rest of the paper. It also contains some helper lemmas. The construction of SLAMP-FSS appears in Section 5. Section 6 lists and proves various properties of SLAMP-FSS, eventually proving that it is indeed a secure function secret sharing scheme for multi-point functions. In Section 7, we discuss the efficiency of SLAMP-FSS and SLAMPR-FSS and compare them with prior works. Finally, Section 8 concludes this paper.

[1] https://github.com/jrmngndr/slamp-fss

3 Related work

Our work expands the distributed point function construction from [BGI16]. The overall idea of the construction is to consider the evaluation as a binary tree where each edge corresponds to an input bit value. Evaluating the function means traversing the tree according to the input and reading the value at the leaf as an output. In an FSS scheme, both parties receive the seed for the root and correction words for each layer of the tree. During evaluation, they expand the seed using a pseudorandom generator (PRG) in the current node and modify the output using the correction word to learn the seeds for the next layer in the tree. The correction words are crafted to ensure that if the input should evaluate to 0 then the parties end up with the same seed, but if the output is nonzero, then the seed and the generated output differ. In the former case, the parties' values XOR to zero, giving a XOR-sharing of the value 0 as needed. In the latter case, the values of the separate parties are likely to give a nonzero output when XOR-ed together. The final layer of correction words can then be used at the leaves to modify the nonzero secret shared output to the secret shared output of the desired function value while leaving the 0 outputs as zeroes.

Concurrently with our work, [BGH+25] proposed two methods for distributed multi-point functions (DMPF) based on the tree structure from [BGI16]. The main difference from [BGI16] is that for a multi-point function, there are t paths from the root to the leaves that give a nonzero output. The overall approach remains the same, but more computation is needed to compute the correction words and the initial keys to ensure correct evaluation. They propose two approaches for this, one using oblivious key-value stores (OKVS) for the correction words and another for big-state using longer modifiers to determine how to use correction words in each layer. They also propose an overall template that specifies how the tree is traversed. This template allows the concrete proposed schemes to focus on how the seeds and correction words are computed and applied and could be used by future schemes as a blueprint. Note that [BGH+25] cites the initial ePrint version of SLAMP-FSS [KKEPR24], considering it as a special case of their OKVS-based construction. However, in the current version of the paper we have made several modifications to the scheme, such as replacing field elements by bits in systems of linear equations, and using tricks to omit PRG-calls in the last round and reducing PRG output size, which make our current version more efficient than the initial proposal for SLAMP-FSS.

The naive approach to DMPF is the sum of t separate distributed point functions that is sometimes called the sum of point functions [BCGI18]. Similar approach with more efficient DPF is to partition the domain of the function into t smaller sections and to apply the original DPF construction to each shorter section separately which is suitable if the points are regularly distributed. The sum of point functions is the main approach that has been used in the applications requiring DMPF and it is used and optimised for various cases, for example [BCGI18, SGRR19, BCG+20, BBC+24].

The second approach proposed in [BCGI18] uses batch codes and can only be used for full evaluation. A probabilistic batch code encodes the length n vector to m codewords. For a DMPF, the vector describing the function only has t non-zero elements and the properties of the code can be chosen such that each of the m codewords is either all zero or has one non-zero element. Hence, distributed point functions can then used to share the codewords. This approach requires m DPF instances where the length of the DPF input depends on the length of the respective codeword. The full evaluation first reconstructs the codewords using full evaluation of the point functions and then decodes the code to learn the t non-zero entries. Probabilistic batch codes were also used for FSS constructions by [ACLS18, SGRR19, dCP22, BBC+24, ARR25]. Note that [ACLS18, SGRR19] implement a weaker primitive called Known-Index FSS, where one of the parties learns the position of the non-zero elements.

According to [BGH+25], the batch code based DMPFs are efficient for large instances where $t > 10^4$. The OKVS-based DMPF is useful for mid-sized DMPF where $t < 10^4$ while their big-state scheme is efficient for small values of t. Our scheme is also most efficient for the small values of t as we have to solve systems of linear equations where the size depends on t.

There is a separate line of work considering DMPF with information theoretic security. Note that achieving information-theoretic security requires more than two parties and limits how many of them could be corrupted. Hence, this is a significantly different setting than the computationally secure two-party setting considered in this paper. Nevertheless, a nice scheme with a good overview of other information-theoretic schemes can be found in [KDK+24].

This paper considers the FSS key generation for the case where the sharer knows the function. We expect that a distributed setup can be achieved analogously to the ideas in [BGH+25] that extend the distributed point function setup from [DS17]. However, this is left as future work. The core idea is that the share generation algorithm can be executed using secure multi-party computation techniques. The main difficulties in the setup are evaluating a PRG and solving a system of t linear equations. For PRGs, it is likely that a solution like the setup algorithm in [DS17] can also be adapted to our case to evaluate all PRGs locally. Secondly, if we wish to solve a system of linear equations $[A]\vec{x} = [\vec{b}]$ over a large field where the secret shared A has t rows and v columns with $t < v$, then there exists the following approach. One can fix the first $v - t$ values of $\vec{x}$, in which case solving the rest of the system is equivalent to solving a related system $[A']\vec{x'} = [\vec{b'}]$ where A' is with an overwhelming probability a random square matrix, and $[A']$ and $[\vec{b'}]$ can be computed with no communication (i.e. quickly). This can be solved by computing $[A'^{-1}]$ and computing $[A'^{-1}][\vec{b'}]$. Computing $[A'^{-1}]$ can in turn be solved by two secure matrix multiplications [BB89] which can be done in secure multi-party computation.

4 Preliminaries

This section gives the main necessary definitions, most are commonly known standard terms. However, this section also gives a novel definition of distributed random-t-multi-point functions in Definition 14, which we will use as a building block for SLAMP-FSS. Some supporting lemmas can be found in Section 4.4.

4.1 Notation

For $\ell \in \mathbb{N}$, $\{0,1\}^\ell$ denotes the bitstrings of length ℓ and $\{0,1\}^*$ is the set of all bitstrings. For $x \in \{0,1\}^*$ and integer $i \in \mathbb{N}$, x^i denotes the i-th bit of x, for $x \in \{0,1\}^*$ and $y \in \{0,1\}^*$, $x\|y$ denotes the concatenation of x and y. For bits $x^1, x^2 \in \{0,1\}$, x^1x^2 denotes the concatenation of x^1 and x^2. Generally, when speaking of a bitstring x, we use $x\|b$ to denote x with the bit $b \in \{0,1\}$ appended to it. For bitstrings $x, y \in \{0,1\}^\ell$, $x^1x^2 \dots x^i = y^1y^2 \dots y^i$ denotes that the first i bits of x coincide with the first i bits of y.

For values x, y, $[x \stackrel{?}{=} y]$ denotes the equality check that outputs 1 if they are equal. $x \xleftarrow{\$} D$ denotes that x is chosen uniformly at random from D. $\mathbb{F}_{2^k}$ denotes the finite field of size 2^k. For $x \in \mathbb{F}_{2^k}$, $\mathsf{Bits}(x)$ denotes the canonical length k bit representation of x. For a vector $\vec{c} \in (\mathbb{F}_{2^k})^e$, c_i denotes the i-th element and $\mathsf{rank}_2(\vec{c})$ denotes the rank of the $k \times e$ matrix with entries from the binary field $\mathbb{F}_2$ where the i-th row of the matrix is $\mathsf{Bits}(c_i)$. For two vectors $\vec{c}, \vec{g}$ of equal length s, their scalar product is denoted as $\langle \vec{c}, \vec{g} \rangle := \sum_{i=1}^{s} c_i g_i$. For a matrix A, we denote by $A^{(i)}$ the i-th row of A. We use $A \leftarrow \mathsf{Mat}_{s,l}(0)$ to denote that A is set to be an $s \times l$ matrix consisting of zeroes. We use $\vec{c} \leftarrow \mathsf{Vec}_s(0)$ to denote that $\vec{c}$ is set to be a vector of length s consisting of zeroes. We use $\{x\}$ to denote the set containing

x and $\{x_i\}_{i=1}^{\ell}$ as the set containing x_1 to x_ℓ. For a value x, $[x]$ denotes its secret shares where party $\mathcal{P}_i$ knows $[x]_i$ such that $x = [x]_1 + [x]_2$, see Section 4.2.4. We denote the $k \times k$ unit matrix by I_k.

In a sequence of games, we sometimes abbreviate the algorithms to focus on the parts that have changed. In this case, we use ... to denote that there are some unchanged lines of code in the game, and we highlight the changes in coloured lines. In addition, we occasionally use the shorthand $|\text{Game } A - \text{Game } B| \leq \varepsilon$ to denote

$$|\Pr[\text{Game } A(\mathcal{A}) = 1] - \Pr[\text{Game } B(\mathcal{A}) = 1]| \leq \varepsilon,$$

where $\mathcal{A}$ is any PPT adversary. Similarly, we use Game A = Game B to denote that the probability is zero.

For sets A and B, $\mathcal{F}(A \to B)$ is the family of all functions $\varphi : A \to B$. We denote point function as $P_{a,b}$ (see Definition 9) and multi-point functions as $P_{\vec{\alpha},\vec{\beta}}$ (see Definition 11). In our constructions, $f\colon \mathbb{F}_{2^k} \to \{0,1\}^v \times \mathbb{F}_{2^k}$, $f'\colon \mathbb{F}_{2^k} \to \{0,1\}^{\bar{v}} \times \mathbb{F}_{2^{\bar{v}}}$ are $\varepsilon_{\mathsf{PRG}}$-PRGs (see Definition 5). The set R_i is the set of active nodes in layer i of the binary tree, t is the number of nonzero points in the multi-point function, n is the length of the function input, κ is a statistical security parameter, and ϵ is an empty bitstring. We require $k, n, t, v, \bar{v} \in \mathbb{N}$, $v > t + \kappa$, and $\bar{v} > k + t + \kappa + 2$. To each node r in the tree, we associate a bitstring $X_r \in \{0,1\}^v$ (for the final layer, $X_r \in \{0,1\}^{\bar{v}}$) and a field element $\tau_r \in \mathbb{F}_{2^k}$ (or $\tau_r \in \mathbb{F}_{2^{\bar{v}}}$ for the last layer). In the evaluation algorithms, these are denoted as $\bar{X}_r$ and $\bar{\tau}_r$. For $t \geq y$ and for $v \geq 0$, we denote with the expression $\xi(y,t,v) := 1 - \frac{\prod_{i=0}^{t-1}(1-2^{i-v})}{\prod_{i=0}^{y-1}(1-2^{i-v})}$.

4.2 Cryptographic definitions

The terms probabilistic polynomial time, pseudorandom generator (PRG), (additive) secret sharing, function secret sharing, ideal cipher model, probability ensemble, statistical distance, ε-indistinguishability, and ε-closeness are defined in the usual way in the following.

4.2.1 Probabilistic polynomial time (PPT) algorithm

Probabilistic polynomial time algorithms are algorithms that run for a polynomial number of steps and that can use some internal randomness to make choices. Polynomial number of steps means that there exists some polynomial t_{poly} such that for all inputs $x \in \{0,1\}^*$ and all internal randomness the time it takes for the algorithm to halt is less than $t_{poly}(|x|)$ [Gol01].

4.2.2 Statistical and computational distance

This section gives standard definitions of statistical distance as in [Gol01] and computational distance as in [KL08].

Definition 1 (Probability Ensemble)**.** Let I be a countable index set. An **ensemble indexed by** I is a sequence of random variables indexed by I. Namely, any $X = \{X_i\}_{i \in I}$, where each X_i is a random variable, is an ensemble indexed by I.

Definition 2 (Statistical distance)**.** Let $X := \{X_i\}_{i \in \mathbb{N}}$ and $Y := \{Y_i\}_{i \in \mathbb{N}}$ be ensembles. The statistical distance between X and Y is defined as

$$SD(X,Y) = \frac{1}{2} \cdot \sum_{\alpha} |\Pr[X_n = \alpha] - \Pr[Y_n = \alpha]|.$$

Definition 3 (ε-indistinguishable)**.** Let $X := \{X_i\}_{i \in \mathbb{N}}$ and $Y := \{Y_i\}_{i \in \mathbb{N}}$ be ensembles. We say X and Y are ε-indistinguishable if for every PPT distinguisher D:

$$|\Pr[D(X_n) = 1] - \Pr[D(Y_n) = 1]| \leq \varepsilon,$$

where $D(X_n)$ means that x is chosen according to the distribution of X_n and then $D(x)$ is executed.

Definition 4 (ε-**close**)**.** We call two cryptographic games Game 0 and Game 1 ε-close, if for all PPT algorithms $\mathcal{A}$

$$|\Pr[\text{Game } 0(\mathcal{A}) = 1] - \Pr[\text{Game } 1(\mathcal{A}) = 1]| \leq \varepsilon,$$

where $\mathcal{G}_0(\mathcal{A})$ denotes that $\mathcal{A}$ is used as a subroutine in $\mathcal{G}_0$ and the probability is taken over the randomness of the games and the randomness of $\mathcal{A}$.
Note: The definition above relates to computational distance. Two ensembles can also be called ε-**close**, if the statistical distance between them is less or equal than ε.

4.2.3 Pseudorandom generator (PRG)

A pseudorandom generator is a polynomial time deterministic algorithm that gets a small uniform seed as its input and outputs a large pseudorandom value [KL08]. Pseudorandom here means that no bounded adversary can distinguish between this and a uniform value.

Definition 5 (PRG [KL08])**.** Let X and Y be sets such that $|X| < |Y|$. We say that a function $f\colon X \to Y$ is a $\varepsilon_{\mathsf{PRG}}$-pseudorandom generator, if for every PPT adversary $\mathcal{A}$

$$\left|\Pr\left[\mathcal{A}(f(x)) = 1 \mid x \xleftarrow{\$} X\right] - \Pr\left[\mathcal{A}(y) = 1 \mid y \xleftarrow{\$} Y\right]\right| \leq \varepsilon_{\mathsf{PRG}}.$$

4.2.4 Secret Sharing

Secret sharing is the process of sharing a piece of data between multiple parties such that a malicious subset of parties cannot access the data without cooperating with at least a set threshold of the parties [Sha79, KL23]. A piece of data held by one party is called a *secret share*. For the purpose of this paper, we focus on the case of two parties, such that the set of parties is $\mathcal{P} := \{\mathcal{P}_1, \mathcal{P}_2\}$. For any value x, the shared version is denoted as $[x]$ such that $x = [x]_1 + [x]_2$ and party $\mathcal{P}_i$ knows $[x]_i$.

Definition 6 (Secret sharing scheme [KL23])**.** A 2-party p_{triv}-private secret sharing scheme for a set of secrets S and a set of parties $\mathcal{P} := \{\mathcal{P}_1, \mathcal{P}_2\}$ consists of a pair of PPT algorithms (share, $\mathsf{reconstruct}$), where

- share: On input $s \in S$ outputs shares $\sigma_1, \sigma_2 \in \{0,1\}^*$;
- $\mathsf{reconstruct}$: On inputs (i, σ_i) from parties $\mathcal{P}_i \in \mathcal{P}$, outputs the secret $s' \in S$ or the error symbol $\perp$.

This scheme must satisfy the following properties:

- **Completeness**: For every $s \in S$, we have

$$\Pr[s' = s \mid (\sigma_1, \sigma_2) \leftarrow \mathsf{share}(s); s' \leftarrow \mathsf{reconstruct}(\sigma_1, \sigma_2)] = 1,$$

- **Privacy**: For every $s \in S$, every subset of parties $\hat{\mathcal{P}} \subset \mathcal{P}$, that satisfy $\left|\hat{\mathcal{P}}\right| < 2$, and every PPT algorithm $\mathcal{A}$

$$\Pr\left[s' = s \mid (\sigma_1, \ldots, \sigma_n) \leftarrow \mathsf{share}(s); s' \leftarrow \mathcal{A}(\{\sigma_i : \mathcal{P}_i \in \hat{\mathcal{P}}\})\right] \leq p_{triv}.$$

The output $\perp$ means that the inputs were inconsistent and no valid output could be given. For reasonable cases, the probability p_{triv} is close to $\frac{1}{|S|}$. Notably, for a perfect secret sharing scheme $p_{triv} = \frac{1}{|S|}$ and the adversary $\mathcal{A}$ is unbounded.

Definition 7 (Additive secret sharing)**.** Let $\mathbb{G}$ be an abelian group. Additive 2-party secret sharing is a perfect secret sharing scheme in which

- share: On input $s \in \mathbb{G}$, samples $\sigma_1 \xleftarrow{\$} \mathbb{G}$. Then computes $\sigma_2 = s - \sigma_1$ and outputs shares σ_1, σ_2.
- $\mathsf{reconstruct}$: On input σ_1, σ_2, outputs $\sum_{i=1}^{n} \sigma_i$. On any other subset of input shares outputs $\perp$.

For a secret shared value τ from some abelian group $\mathbb{G}$ and a two-party additive secret sharing scheme, we denote the share of $\mathcal{P}_i$ as $[\tau]_i$ such that $[\tau]_1 + [\tau]_2 = \tau$. We denote by $[\tau]$ the tuple $([\tau]_1, [\tau]_2)$ and call it a secret-sharing of τ. For an abelian group $\mathbb{G}$, a function $f\colon \mathbb{G} \to \mathbb{G}$, a value $\tau \in \mathbb{G}$, and $[\tau] = ([\tau]_1, [\tau]_2)$, we use the notation $f([\tau])$ to denote the tuple $(f([\tau]_1), f([\tau]_2))$. Thus, if in some algorithm we write $[z] \leftarrow f([\tau])$, it means that $[z]_1 \leftarrow f([\tau]_1)$ and $[z]_2 \leftarrow f([\tau]_2)$.

4.2.5 Function secret sharing (FSS)

Function secret sharing is a type of secret sharing, where the data shared by the parties is a description of some function φ from a predefined function family $\mathcal{F}$. This description can be used to calculate the secret share of $\varphi(x)$ for any x in the domain of φ [BGI15].

Definition 8 (Function secret sharing [BGI15])**.** A 2-party function secret sharing (FSS) scheme with respect to function class $\mathcal{F}(X \to Y)$ and secret sharing scheme SS is a pair of PPT algorithms $(\mathsf{Gen}, \mathsf{Eval})$, where

- Gen: Gets the description of $\varphi \in \mathcal{F}$ and security parameter κ as its input and it outputs 2 keys $\mathsf{sk}_1, \mathsf{sk}_2 \in \{0,1\}^*$.
- Eval: On input (i, k_i, x), the algorithm outputs the i-th party's secret share of $\varphi(x)$ with respect to SS, where $i \in \{1,2\}$, $\mathsf{sk}_i \in \{0,1\}^*$ and $x \in X$.

The scheme must satisfy the following properties:

- **Correctness**: For all $\varphi \in \mathcal{F}(X \to Y)$, $x \in X$,

$$\Pr[\mathsf{reconstruct}(\mathsf{Eval}(1, \mathsf{sk}_1, x), \mathsf{Eval}(2, \mathsf{sk}_2, x)) = \varphi(x) | \mathsf{sk}_1, \mathsf{sk}_2 \leftarrow \mathsf{Gen}(1^\kappa, \varphi)] = 1.$$

- **Security**: We call an FSS scheme secure, if for corrupted party $T \in \{1,2\}$ and for all PPT algorithms $\mathcal{A}$ there exists a negligible function $\varepsilon_{\mathsf{FSS}}$, such that:

$$\Pr[c = \tilde{c}] - \frac{1}{2} \leq \varepsilon_{\mathsf{FSS}}(\kappa),$$

  where c and $\tilde{c}$ come from the following experiment:

  - The adversary outputs $(\varphi_1, \varphi_2, \sigma) \leftarrow \mathcal{A}(1^\kappa)$, where $\varphi_1, \varphi_2 \in \mathcal{F}(X \to Y)$ are descriptions of two functions from function family $\mathcal{F}$ and $\sigma \in \{0,1\}^*$ is the state of $\mathcal{A}$.
  - The challenger selects $c \xleftarrow{\$} \{0,1\}$ and calculates $(\mathsf{sk}_1, \mathsf{sk}_2) \leftarrow \mathsf{Gen}(1^\kappa, \varphi_b)$.
  - The adversary gets the corrupted key and calculates $\tilde{c} \leftarrow \mathcal{A}(\mathsf{sk}_T, \sigma)$.

Note that in the rest of this paper, we use syntactic sugar to denote $\mathsf{Eval}(i, k_i, x)$ as $\mathsf{Eval}^i(k_i, x)$. Note that Definition 8 is using the game-based approach mirroring the ideas of semantic security. However, an equivalent definition can also be given using simulation-based security as discussed in [BGI15]. In the simulation-based variant, the security property requires the adversary to distinguish between a key generated correctly from the one generated by a simulator without access to the input function.

4.2.6 Distributed point and multi-point functions

The main classes of functions for which FSS schemes exist are point and multi-point functions. The respective FSS schemes are called distributed point or multi-point functions. In addition to these, we also define a distributed random multi-point function. This primitive behaves quite like a distributed multi-point function except that the nonzero output values of the function are determined during key generation and not defined by the input party.

Definition 9 (Point function)**.** Let X be a set and let $\mathbb{G}$ be an abelian group. For $\alpha \in X$ and $\beta \in \mathbb{G}$ the point function $P_{\alpha,\beta} : X \to \mathbb{G}$ is defined by $P_{\alpha,\beta}(\alpha) = \beta$ and $P_{\alpha,\beta}(x) = 0$ for all $x \neq \alpha$.

Definition 10 (Distributed point function (DPF) [BGI15], [GI14])**.** A distributed point function (DPF) is an FSS scheme with respect to the family of point functions over $X = \{0,1\}^n$ and $\mathbb{G} = \mathbb{F}_{2^m}$ where the group operation is defined as the pointwise XOR.

Definition 11 (t-multi-point function [BCGI18])**.** Let X be a set and let $\mathbb{G}$ be an abelian group. For $\vec{\alpha} \in X^t$ and $\vec{\beta} \in \mathbb{G}^t$ the t-multi-point function $P_{\vec{\alpha},\vec{\beta}}$ is defined by

$$P_{\vec{\alpha},\vec{\beta}}(x) = \begin{cases} b_i, \text{ if } x = \alpha_i, \\ 0, \text{ otherwise,} \end{cases}$$

where α_i is the i-th element of $\vec{\alpha}$ and β_i is the i-th element of $\vec{\beta}$.

Definition 12 (Distributed multi-point function (DMPF))**.** A distributed multi-point function (DMPF) is an FSS scheme with respect to the family of multi-point functions over $X = \{0,1\}^n$ and $\mathbb{G} = \mathbb{F}_{2^m}$ where the group operation is defined to be as the pointwise XOR.

Definition 13 (Random-t-multi-point function)**.** Let X be a set and let $\mathbb{G}$ be an abelian group. For $\vec{\alpha} \in X^t$, the random-t-multi-point function $P_{\vec{\alpha}}$ is defined by

$$P_{\vec{\alpha}}(x) = \begin{cases} \text{randomly distributed element of } \mathbb{G}, & \text{if } x = \alpha_i, \\ 0, & \text{otherwise,} \end{cases}$$

where α_i is the i-th element of $\vec{\alpha}$ and β_i is the i-th element of $\vec{\beta}$.

Definition 14 (Distributed random-t-multi-point function)**.** Let x be a set and let $\mathbb{G}$ be an abelian group. Distributed random-t-multi-point function is a 2-party FSS scheme $(\mathsf{GenRand}, \mathsf{EvalRand})$ with respect to the family of random-t-multi-point functions from $X = \{0,1\}^n$ to $\mathbb{G} = \mathbb{F}_{2^m}$ where the group operation is defined to be as the pointwise XOR and the value of the output is fixed during $\mathsf{GenRand}$ execution. Distributed random-t-multi-point function is (τ, ε)-*close to random* if for any τ-time adversary $\mathcal{A}$, there exists a PPT $\mathsf{GenPrim}$ such that

$$|\Pr[\text{Game } 1(\mathcal{A}) = 1] - \Pr[\text{Game } 2(\mathcal{A}) = 1]| \leq \varepsilon,$$

where the games are defined in Figure 1.

In the games 1 and 2, the σ refers to the state of the adversary. Note that the idea is that when the FSS scheme is evaluated, the random elements $\vec{\beta}$ is fixed during the key generation $\mathsf{GenRand}$ and for the $\mathsf{EvalRand}$ the output for any α_i is deterministic. $\mathsf{GenRand}$ only takes the input $\vec{\alpha}$.

Game 1:
Input : Adversary $\mathcal{A}$
1 $(\alpha_1, \ldots, \alpha_t) \in X^t, \mu \in \{1,2\}, \sigma \leftarrow \mathcal{A}()$
2 $\bar{\mu} \leftarrow 3 - \mu$
3 $\mathsf{sk}_1, \mathsf{sk}_2 \leftarrow \mathsf{Gen}((\alpha_1, \ldots, \alpha_t), f)$
4 **for** $i = 1$ **to** t **do**
5 $\quad [z_i]_{\bar{\mu}} \leftarrow \mathsf{EvalRand}^{\bar{\mu}}(\mathsf{sk}_{\bar{\mu}}, f, \alpha_i)$
6 **return** $\mathcal{A}(\mathsf{sk}_\mu, \{[z_i]_{\bar{\mu}}\}_{i=1}^t, \sigma)$

Game 2:
Input : Adversary $\mathcal{A}$
1 $(\alpha_1, \ldots, \alpha_t) \in X^t, \mu \in \{1,2\}, \sigma \leftarrow \mathcal{A}()$
2 $\bar{\mu} \leftarrow 3 - \mu$
3 $\mathsf{sk}_1, \mathsf{sk}_2 \leftarrow \mathsf{GenPrim}(f)$
4 **for** $i = 1$ **to** t **do**
5 $\quad [z_i]_{\bar{\mu}} \xleftarrow{\$} \mathbb{G}$
6 **return** $\mathcal{A}(\mathsf{sk}_\mu, \{[z_i]_{\bar{\mu}}\}_{i=1}^t, \sigma)$

Figure 1. Games defining distributed random-t-multi-point function.

4.2.7 Ideal Cipher Model

The Ideal Cipher Model [CPS08] essentially assumes that the output of a block cipher with a known key is randomly distributed. Note that we use the model only for showing the correctness of the model, that is, we use it in the proofs where there is no adversary who tries to break the model but we merely use it to assume that the output of a block cipher is approximately evenly distributed with regard to some subspaces. We consider this a reasonable assumption to make, as this assumption is typically made in this line of work.

Definition 15 (Block cipher [KL08])**.** Let $\gamma, \kappa \in \mathbb{N}$ and $F : \{0,1\}^\gamma \times \{0,1\}^\kappa \to \{0,1\}^\gamma$ be an efficient function. We call F a **keyed permutation** or a block cipher if for all $k \in \{0,1\}^\kappa$ the function $F(\cdot, k)$ is bijective.

Definition 16 (Ideal Cipher Model (ICM)[CPS08])**.** The Ideal Cipher Model (ICM) is an idealised model where a publicly accessible block cipher exists. This block cipher takes in κ-bit keys and γ-bit inputs and returns γ bit outputs and is chosen uniformly from all such block ciphers. This is equivalent to choosing a function from 2^κ independent random permutations.

Note that in this paper, we use the input to our function as the key to the PRG that is evaluated at a fixed point, thus achieving that the output will be longer than the input. This approach was introduced by Guo et al. [GKWY20] and is commonly used. We give more details when using this property in Section 6.

We also note that we use the model to simplify the proofs of Propositions 1 and 2. These results show that the probability of the key generation algorithm outputting $\perp$ is low. We comment in Remark 1 how the Ideal Cipher Model could be omitted from this work.

4.3 Algebraic definitions

The terms kernel, rank and nullity are defined the usual way. Now let $\mathbb{F}$ be a field and $k, n \in \mathbb{N}$. The following definitions and theorems are based on [AR91]. We give our definition for rank_2.

Theorem 1. *A system of linear equations $Ax = b$ is solvable if and only if the rank of the coefficient matrix A is the same as the rank of the augmented matrix $(A \mid b)$.*

Theorem 2 (Dimension theorem of matrices)**.** *If A is a matrix with n columns then*

$$\mathsf{rank}(A) + \mathsf{nullity}(A) = n.$$

We note that we are working with $\mathbb{F}_{2^k}$ and that the set $\mathbb{F}_{2^k}$ can also be considered as a vector space over $\mathbb{F}_2$ of dimension k.

We also need to consider the rank and solvability of linear equations of elements of $(\mathbb{F}_2)^k$. For an element $a \in \mathbb{F}_{2^k}$ we define $\mathsf{Bits}(a)$ to be the canonical bitstring of length k.

Definition 17 (Rank$_2$)**.** Let $Z \in \mathbb{F}_{2^k}^{t \times v}$, with $Z = (z_{i,j})_{i=1,j=1}^{t,v}$ such that $z_{i,j} \in \{0,1\}$. Let the row vectors of Z be $\{\vec{z_i}\}_{i=1}^{t}$. We say that $\mathsf{rank}_2(Z)$ is the rank of the matrix Z over the field $\mathbb{F}_2$. Let $\vec{\beta} = \{\beta_1, \ldots, \beta_t\}$ with all $\beta_i \in \mathbb{F}_{2^k}$. Let B be the matrix with the row vectors $\mathsf{Bits}(\beta_1), \ldots, \mathsf{Bits}(\beta_t)$. We define $\mathsf{rank}_2(B)$ (or $\mathsf{rank}_2(\vec{\beta})$) to be the rank of the matrix B over the field $\mathbb{F}_2$. We define $\mathsf{rank}_2(Z|\vec{\beta})$ to be the rank of the binary matrix $Z|B$ over $\mathbb{F}_2$.

4.4 Additional lemmata

This section proves some helper theorems for systems of linear equations over binary fields, needed to analyze the efficiency of SLAMP-FSS.

Lemma 1. *Let $\vec{d} \in \mathbb{F}_{2^k}^v$ where $v \geq k$. If $\mathsf{rank}_2(\vec{d}) = k$, then for every $\rho \in \mathbb{F}_{2^k}$, the number of binary vectors $X \in (\mathbb{F}_2)^v$ such that $\left\langle X, \vec{d} \right\rangle = \rho$ is 2^{v-k}.*

Proof. Define for any $\rho \in \mathbb{F}_{2^k}$, the set $S_\rho := \{X \in (\mathbb{F}_2)^v | \left\langle X, \vec{d} \right\rangle = \rho\}$. Now, we see that for any $\rho \in \mathbb{F}_{2^k}$ for which $S_\rho \neq \emptyset$, if $X_0 \in S_\rho$ and $X_1 \in S_0$, then $X_0 + X_1 \in S_\rho$. Indeed, $\left\langle X_0 + X_1, \vec{d} \right\rangle = \left\langle X_0, \vec{d} \right\rangle + \left\langle X_1, \vec{d} \right\rangle = \rho + 0 = \rho$.

Similarly, if for some $X_2, X_3 \in (\mathbb{F}_2)^v$, $\left\langle X_2, \vec{d} \right\rangle = \rho = \left\langle X_3, \vec{d} \right\rangle$, then $\left\langle X_2 - X_3, \vec{d} \right\rangle = \left\langle X_2, \vec{d} \right\rangle - \left\langle X_3, \vec{d} \right\rangle = \rho - \rho = 0$. Hence, if $X_2, X_3 \in S_\rho$ for some $\rho \in \mathbb{F}_{2^k}$, then $X_2 - X_3 \in S_0$.

Thus, we can partition the space $(\mathbb{F}_2)^v$ into classes S_ρ.

First note that no class S_ρ is empty. Indeed, this follows from the fact that $\vec{d}$ is full-rank. (Given some value ρ, one can express it as a sequence of bits $\rho_1, \ldots, \rho_k$. One can now express the condition as the following matrix equation over $\mathbb{F}_2$: $(x_1, \ldots, x_k)D = (\rho_1, \ldots, \rho_k)$ where $D = \mathsf{Bits}(\vec{d})$. If $v = k$, one can for any ρ compute $(x_1, \ldots, x_k) \leftarrow (\rho_1, \ldots, \rho_k)D^{-1}$. If $v > k$, one can choose k elements of $\vec{d}$ that are linearly independent, form a matrix D' from them and similarly compute $(x_1, \ldots, x_k) \leftarrow (\rho_1, \ldots, \rho_k)D'^{-1}$ and for all elements of $\vec{d}$ that are not in that linearly independent set, set the corresponding values of x to be zeroes.)

We note that any class S_ρ has the same size as S_0. Indeed, take any $X' \in S_\rho$, which we can do as S_ρ is non-empty. Then define the function $f\colon S_0 \to S_\rho$ by $f(X) = X + X'$. By the arguments above, it is correctly defined and surjective. It is also clearly injective. Thus, the sets S_0 and S_ρ have the same size.

Hence, all the S_ρ are the same size. As the number of the ρ is 2^k, it follows that $|S_\rho| = \frac{2^v}{2^k} = 2^{v-k}$. Hence, the claim is proven. □

The following is well known.

Lemma 2 ([MP13])**.** *Let A be a random $m \times n$ matrix over $\mathbb{F}_2$ where $m \geq n$, where all elements are set to either 0 or 1 with an equal probability. Then the probability that A has full rank is $\prod_{i=0}^{n-1}(1 - 2^{i-m})$.*

The following generalisation of Lemma 2 follows quite easily.

Lemma 3. *Let A be a random $m \times n$ matrix over $\mathbb{F}_{2^k}$ where $m \geq n$, where all elements are set to either 0 or 1 with an equal probability. Then the probability that A has full rank is $\prod_{i=0}^{n-1}(1 - 2^{i-m})$*

For the next lemma we will need the following definition:

Definition 18 (generalised inverse)**.** For $A \in \mathbb{F}_2^{t\times v}$, it's generalised inverse is a matrix $A^+ \in \mathbb{F}_2^{v\times t}$ that satisfies the following properties:

- $AA^+A = A$.
- $A^+AA^+ = A^+$.

Lemma 4. *Let $A, B \in \mathbb{F}_2^{t\times v}$ where $t < v$ and let A, B have full rank. Let $AX = B$ be a matrix equation where $X \in \mathbb{F}_2^{v\times v}$. Then, the equation is solvable for X and the probability that $\mathsf{rank}_2(X) = v$ is $\prod_{i=0}^{v-t-1}(1-2^{i-v+t})$. Also, $\prod_{i=0}^{v-t-1}(1-2^{i-v+t}) > e^{\frac{-4}{3}}$.*

Proof. The existence of the solution follows directly from Lemma 5. The probability proof uses the generalised inverse. It is well known that for a system of equations $AX = B$, when solving for X, if a solution exists, then $AA^+B = B$ and all the solutions D can be characterised as

$$D = A^+B + (I_v - A^+A)W,$$

where W is arbitrary.

Let us do a proof sketch for this claim. Indeed, first note that $A^+(AA^+B - B) = A^+AA^+B - A^+B = A^+B - A^+B = 0$. Thus $(AA^+B - B)$ belongs to the right kernel of A^+. However, since A^+ has full rank and has more rows than columns, then its right kernel must be just zero, hence $AA^+B - B = 0$, i.e $AA^+B = B$. Secondly, suppose that $AA^+B = B$ and $D = A^+B + (I_v - A^+A)W$, for some arbitrary W. Then $AD = AA^+B + A(I_v - A^+A)W = B + AW - AA^+AW = B$. Thirdly, suppose that for some D, $AD = B$ and that $D = A^+B + C$. Then $B = AD = AA^+B + AC = B + AC$ and hence $AC = 0$. Since the kernel of A can be expressed as $(I_v - A^+A)W$ for arbitrary W, one can see that thus $D = AA^+B + A(I_v - A^+A)W$ for some W.

Now let us continue with the proof of the main lemma.

Note that since $\mathsf{rank}(A) = t$, we can represent $A = \begin{pmatrix} A' & 0 \end{pmatrix} G$ for some invertible G and full-rank $A' \in \mathbb{F}_2^{t\times t}$. Now we can take $A^+ = G^{-1}\begin{pmatrix} A'^{-1} \\ 0 \end{pmatrix}$.

Indeed, in this case $AA^+ = \begin{pmatrix} A' & 0 \end{pmatrix} GG^{-1}\begin{pmatrix} A'^{-1} \\ 0 \end{pmatrix} = I_t$, and from this it follows that it is a generalised inverse.

Naturally, $\mathsf{rank}(A^+) = t$.

Note that

$$A^+A = G^{-1}\begin{pmatrix} A'^{-1} \\ 0 \end{pmatrix}\begin{pmatrix} A' & 0 \end{pmatrix} G = G^{-1}\begin{pmatrix} I_t & 0 \\ 0 & 0 \end{pmatrix} G.$$

We can similarly represent B as $B = \begin{pmatrix} B' & 0 \end{pmatrix} H$.

Now,

$$A^+B = G^{-1}\begin{pmatrix} A'^{-1} \\ 0 \end{pmatrix}\begin{pmatrix} B' & 0 \end{pmatrix} H = G^{-1}\begin{pmatrix} A'^{-1}B' & 0 \\ 0 & 0 \end{pmatrix} H.$$

Since $A'^{-1}B'$ has rank t and multiplying with invertible matrices does not change rank, $\mathsf{rank}(A^+B) = t$.

On the other hand,

$$I_v - A^+A = G^{-1}G - G^{-1}\begin{pmatrix} I_t & 0 \\ 0 & 0 \end{pmatrix} G \quad = G^{-1}\left(I_v - \begin{pmatrix} I_t & 0 \\ 0 & 0 \end{pmatrix}\right) G = G^{-1}\begin{pmatrix} 0 & 0 \\ 0 & I_{v-t} \end{pmatrix} G.$$

Now let us pick at random $W_4 \xleftarrow{\$} \mathbb{F}_2^{v-t\times v-t}$. By Lemma 3,

$$\Pr[\mathsf{rank}(W_4) = v - t] = \prod_{i=0}^{v-t-1}(1-2^{i-v+t}).$$

Also pick $W_1 \xleftarrow{\$} \mathbb{F}_2^{t\times t}, W_2 \xleftarrow{\$} \mathbb{F}_2^{t\times v-t}$ and $W_3 \xleftarrow{\$} \mathbb{F}_2^{v-t\times t}$.

Let

$$W' := \begin{pmatrix} W_1 & W_2 \\ W_3 & W_4 \end{pmatrix}.$$

Our goal is to do a construction from which it follows that the general solution $D = A^+B + (I_v - A^+A)W$, where we will define $W = G^{-1}W'H$, is invertible if and only if W_4 is invertible, and from that to conclude the bound.

Define $W'' = G^{-1}W'$. W'' is also distributed uniformly at random in $\mathbb{F}_2^{t\times v}$. Now let us define a matrix D' and do some arithmetic with it:

$$D' := \begin{pmatrix} {A'}^{-1}B' & 0 \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 0 & I_{v-t} \end{pmatrix} GW'' = \begin{pmatrix} {A'}^{-1}B' & 0 \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 0 & I_{v-t} \end{pmatrix} \begin{pmatrix} W_1 & W_2 \\ W_3 & W_4 \end{pmatrix} =$$

$$= \begin{pmatrix} {A'}^{-1}B' & 0 \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ W_3 & W_4 \end{pmatrix} = \begin{pmatrix} {A'}^{-1}B' & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} I_t & 0 \\ W_3 & W_4 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 0 & I_{v-t} \end{pmatrix} \begin{pmatrix} I_t & 0 \\ W_3 & W_4 \end{pmatrix} =$$

$$= \begin{pmatrix} {A'}^{-1}B' & 0 \\ 0 & I_{v-t} \end{pmatrix} \begin{pmatrix} I_t & 0 \\ W_3 & W_4 \end{pmatrix}.$$

Since $\begin{pmatrix} {A'}^{-1}B' & 0 \\ 0 & I_{v-t} \end{pmatrix}$ is full-rank, D' is full-rank if and only if W_4 is full-rank.

We note that D' is full-rank iff $D := G^{-1}D'H$ has full rank. Define $W := W''H$ and note that this is also distributed uniformly at random in $\mathbb{F}_2^{t\times v}$.

However,

$$D = G^{-1}D'H = G^{-1}\begin{pmatrix} {A'}^{-1}B' & 0 \\ 0 & 0 \end{pmatrix} H + G^{-1}\begin{pmatrix} 0 & 0 \\ 0 & I_{v-t} \end{pmatrix} GW''H = A^+B + (I_v - A^+A)W,$$

where W is distributed uniformly at random in $\mathbb{F}_2^{t\times v}$.

We note that this is exactly the case for the general solution D. Thus, this D is invertible if and only if W_4 is invertible. Hence, the main claim of the lemma holds.

Plugging in Corollary 1, we obtain the other claim of the lemma, as $\prod_{i=0}^{v-t-1}(1 - 2^{i-v+t}) = (1-2^{-1})\dots(1-2^{-(v-t)}) > e^{\frac{-4}{3}}$.

□

Lemma 5. *Let $Z \in \mathbb{F}_{2^k}^{t\times v}$, with $Z = (z_{i,j})_{i=1,j=1}^{t,v}$ where all $z_{i,j} \in \{0,1\}$. Let $\vec{b} = \{b_1,\dots,b_t\}$ with all $b_i \in \mathbb{F}_{2^k}$. Then, $\mathsf{rank}_2(Z) = \mathsf{rank}_2(Z|\vec{b})$, if and only if, the linear system of equations $Z\vec{g} = \vec{b}$ over $\mathbb{F}_{2^k}$, where $\vec{g}$ are the unknowns, has at least one solution.*

Moreover, if the system has a solution, it has exactly $(2^k)^{v-\mathsf{rank}(Z)}$ solutions.

Proof. Suppose that $\mathsf{rank}_2(Z) = \mathsf{rank}_2(Z|\vec{b}) = r$. Let $B = \mathsf{Bits}(\vec{b})$. We note that multiplying the matrices Z and $Z|B$ by an invertible binary matrix does not change their respective $\mathsf{rank}_2()$. We also note that we can apply binary row transformations to Z to obtain a matrix where the last $t-r$ rows are zero and the first r rows are nonzero, more specifically, each column has just one 1, and in all of the first r rows, there is at least one 1. These binary row transformations can be represented as a binary $t\times t$ matrix Y. Additionally, let us have a $v\times v$ permutation matrix W such that the upper left $r\times r$ submatrix of YZW is the identity matrix. Let W' be a $(v+k)\times(v+k)$ matrix where the upper left $v\times v$ submatrix is W, the lower right $k\times k$ submatrix is the identity, and the other elements are zeroes, that is, $W' = \begin{pmatrix} W & 0 \\ 0 & I_k \end{pmatrix}$.

We now note that it follows that $Y(Z|B)W'$ must also have that the last $t-r$ rows are zero. Indeed, if this were not the case, one could first do elementary column transformations and make the first r rows of the elements corresponding to B into zeroes by adding the i-th

column to every column where B has a 1 on i-th row. These transformations would not affect the last $t-r$ rows as these are all zeroes. Now, if the last $t-r$ rows of $Y(Z|B)W'$ were not zeroes, then the resulting matrix would have rank at least $r+1$, which would be a contradiction.

Opening $Y(Z|B)W'$ we get $YZW|YB$. Now, we have a system of equations

$$YZW\vec{g'} = YB.$$

We can also interpret YB as $Y\vec{b}$ as a vector of elements of $\mathbb{F}_{2^k}$, and here as well the last $t-r$ elements are zero. Let $\vec{b_Y} \in \mathbb{F}_{2^k}^r$ be such that $Y\vec{b} = \begin{pmatrix} \vec{b_Y} \\ 0 \end{pmatrix} \mathbb{F}_{2^k}^t$.

Now set $\vec{g'}$ to be equal to $\begin{pmatrix} \vec{b_Y} \\ 0 \end{pmatrix} \in \mathbb{F}_{2^k}^v$. Denote by $(YZW)_{i,j}$ the element of the i-th row and j-th column of YZW. Now, let us verify that $YZW\vec{g'} = Y\vec{b}$. Let i be the index of the row in question. For $i \in [1,r]$ we have that $\sum_{j=1}^{v}(YZW)_{i,j}g'_j = \sum_{j=1}^{r}\delta_{i,j}g'_j + \sum_{j=r+1}^{v}(YZW)_{i,j}\cdot 0 = g'_i = (\vec{b_Y})_i = (YB)_i$, which we wanted to show. (Here $\delta_{i,j}$ is the Kronecker delta.) For $i > r$, we have that all $(YZW)_{i,j} = 0$ and that all $(YB)_i = 0$, thus the equalities are trivially satisfied.

Denote $W\vec{g'} =: \vec{g''}$. We have that $YZ\vec{g''} = Y\vec{b}$. Multiplying on the left by Y^{-1}, we get that $Z\vec{g''} = \vec{b}$, and, hence, we have a solution.

For the converse, suppose that there exists a solution $\vec{\alpha}$ that satisfies the given system of linear equations, that is

$$Z\vec{\alpha} = \vec{b}.$$

Suppose for the contradiction that $\mathsf{rank}_2(Z) \neq \mathsf{rank}_2(Z|\vec{b})$. Obviously, adding columns to a matrix can not diminish the rank, thus it must be that $\mathsf{rank}_2(Z) < \mathsf{rank}_2(Z|\vec{b})$. Denote $\mathsf{rank}_2(Z) = r$.

Similarly to above, define Y, W and W'. Define $\vec{\alpha}' = W^{-1}\vec{\alpha}$.

We have that $YB = YZ\vec{\alpha} = YZWW^{-1}\vec{\alpha} = YZW\vec{\alpha}'$. We have that the last $t-r$ rows of YZW are zeroes. However, since $\mathsf{rank}_2(Z|B) > r$, then also $\mathsf{rank}_2(YZ|YB) > r$. Thus, one of the last $t-r$ rows of $Y\vec{b}$ must contain a nonzero element. Let this row be the j-th row. We thus have that

$$0 = \sum_{j=1}^{v} 0\vec{\alpha}'_j = \sum_{j=1}^{v}(YZW)_{i,j}\vec{\alpha}'_j = (Y\vec{b})_i \neq 0,$$

a contradiction. Thus, the result holds.

Concerning the number of solutions, we note that we simply have a system of linear equations over $\mathbb{F}_{2^k}$. The number of its solutions is well-known to be $(2^k)^{v-\mathsf{rank}(Z)}$. Indeed, we can choose arbitrary values for the first $v - \mathsf{rank}(Z)$ values of the unknowns and the other $\mathsf{rank}(Z)$ values are uniquely fixed by them. □

The following results needed for SLAMP-FSS follow from Lemma 2 and Lemma 3.

Lemma 6. *Let A be a random $m \times n$ matrix over $\mathbb{F}_{2^k}$ where $m > n$, where all elements are set to either 0 or 1 with an equal probability. Then the probability that A has full rank is no less than $1 - 2^{n-m}$.*

Proof. From Lemma 2 we know that the probability is $\prod_{i=0}^{n-1}(1-2^{i-m})$. We note that when we expand this, we obtain

$$\prod_{i=0}^{n-1}(1-2^{i-m}) = 1 - 2^{-m} - 2^{1-m} - \cdots - 2^{n-1-m} + 2^{-m}2^{1-m} + \cdots =$$

$$= 1 - \sum_{i=0}^{n-1} 2^{i-m} + \sum_{k=1}^{n-1} \left((-1)^{k+1} \sum_{0 \le i_0 < \ldots < i_k \le n-1} \prod_{j=0}^{k} 2^{i_j - m} \right).$$

Here, we order the summands such that the summands with fewer terms are placed earlier. Note that the operator $\sum_{0 \le i_0 < \ldots < i_k \le n-1}$ picks k-combinations of indices. Let us split the second part into positive and negative summands and get

$$\sum_{\substack{k=1 \\ 2 \nmid k}}^{n-1} \left(\sum_{0 \le i_0 < \ldots < i_k \le n-1} \prod_{j=0}^{k} 2^{i_j - m} \right) - \sum_{\substack{k=1 \\ 2 \mid k}}^{n-1} \left(\sum_{0 \le i_0 < \ldots < i_k \le n-1} \prod_{j=0}^{k} 2^{i_j - m} \right).$$

We want to show that the previous expression is positive. For this, we now split the proof into two depending on whether $2 \mid n-1$. If $2 \mid n-1$, then we can pair each $2 \nmid k$ with $k+1$. If $2 \nmid n-1$, then the last summand is positive, which means we can forget about it, and we can pair up all the other summands in a similar manner. So, let us fix k such that $2 \nmid k$ and get

$$\sum_{0 \le i_0 < \ldots < i_k \le n-1} \prod_{j=0}^{k} 2^{i_j - m} - \sum_{0 \le i_0 < \ldots < i_{k+1} \le n-1} \prod_{j=0}^{k+1} 2^{i_j - m} =$$

$$= \sum_{0 \le i_0 < \ldots < i_k \le n-1} \prod_{j=0}^{k} 2^{i_j - m} - \sum_{0 \le i_0 < \ldots < i_k \le n-1} \prod_{j=0}^{k} 2^{i_j - m} \left(\sum_{i_{k+1} = i_k + 1}^{n-1} 2^{i_{k+1} - m} \right) =$$

$$= \sum_{0 \le i_0 < \ldots < i_k \le n-1} \prod_{j=0}^{k} 2^{i_j - m} \left(1 - \sum_{i_{k+1} = i_k + 1}^{n-1} 2^{i_{k+1} - m} \right).$$

If $i_k + 1 > n-1$, then $\sum_{i_{k+1}=i_k+1}^{n-1} 2^{i_{k+1}-m} = 0$ and the expression must be positive. If $i_k + 1 \le n-1$, then we get

$$\sum_{i_{k+1} = i_k + 1}^{n-1} 2^{i_{k+1} - m} = 2^{i_{k+1} - m} \cdot \frac{2^{n-1-i_{k+1}+1} - 1}{2-1} = 2^{n-m} - 2^{i_{k+1}-m}.$$

Since $2^{n-m} < 1$, then

$$1 - \sum_{i_{k+1} = i_k + 1}^{n-1} 2^{i_{k+1} - m} = 1 - 2^{n-m} + 2^{i_{k+1}-m} > 0$$

and this means

$$\sum_{0 \le i_0 < \ldots < i_k \le n-1} \prod_{j=0}^{k} 2^{i_j - m} \left(1 - \sum_{i_{k+1} = i_k + 1}^{n-1} 2^{i_{k+1} - m} \right) > 0$$

from which we can conclude

$$1 - \sum_{i=0}^{n-1} 2^{i-m} + \sum_{k=1}^{n-1} \left((-1)^{k+1} \sum_{0 \le i_0 < \ldots < i_k \le n-1} \prod_{j=0}^{k} 2^{i_j - m} \right) >$$

$$> 1 - \sum_{i=0}^{n-1} 2^{i-m} = 1 - 2^{n-m} + 2^{-m} > 1 - 2^{n-m}.$$

□

Lemma 7. *For any $n > 1$ it holds that*

$$2^{\frac{n(n+1)}{2}-2} < \prod_{i=0}^{n-1}(2^{i+1}-1) < 2^{\frac{n(n+1)}{2}}$$

Proof. For the upper bound, we see that

$$\prod_{i=0}^{n-1}(2^{i+1}-1) < \prod_{i=0}^{n-1} 2^{i+1} = 2^{\sum_{i=0}^{n-1}(i+1)} = 2^{\frac{n(n+1)}{2}}.$$

For the lower bound, we first note that

$$\prod_{i=0}^{n-1}(2^{i+1}-1) = \left(\prod_{i=0}^{n-1} 2^{i+1}\right)\prod_{i=0}^{n-1}\left(1-\frac{1}{2^{i+1}}\right) = 2^{\frac{n(n+1)}{2}}\prod_{i=0}^{n-1}\left(1-\frac{1}{2^{i+1}}\right) \geq$$
$$\geq 2^{\frac{n(n+1)}{2}}\prod_{i=0}^{\infty}\left(1-\frac{1}{2^{i+1}}\right).$$

The last element is well-defined, because the sequence $\prod_{i=0}^{n-1}\left(1-\frac{1}{2^{i+1}}\right)$ is strictly decreasing as $n \to \infty$ and is in $[0,1)$, and thus bounded, it must converge into a value in $[0,1)$.

To estimate $\prod_{i=0}^{\infty}\left(1-\frac{1}{2^{i+1}}\right)$, let us take logarithm of it.

We have that

$$\ln\left(\prod_{i=0}^{\infty}\left(1-\frac{1}{2^{i+1}}\right)\right) = \sum_{i=0}^{\infty}\ln\left(1-\frac{1}{2^{i+1}}\right) = \sum_{i=0}^{\infty}\sum_{j=1}^{\infty}\frac{-(2^{-(i+1)})^j}{j},$$

where the last equality comes from the Taylor expansion of $\ln(1-x)$.

Now note that for $i \geq 0$, we have

$$\sum_{j=1}^{\infty}\frac{-(2^{-(i+1)})^j}{j} \geq -(2^{-(i+1)}) + \sum_{j=2}^{\infty}-\frac{(2^{-(i+1)})^j}{2} =$$
$$= -(2^{-(i+1)}) - \frac{2^{-2i-2}}{2(1-2^{-i-1})} \geq -(2^{-(i+1)}) - 2^{-2(i+1)}.$$

Note that $\sum_{i=0}^{\infty} -2^{-(i+1)} = -1$, and $\sum_{i=0}^{\infty} -2^{-2(i+1)} = -\frac{1}{3}$. Hence,

$$\sum_{i=0}^{\infty}\sum_{j=1}^{\infty}\frac{-(2^{-(i+1)})^j}{j} > -\frac{4}{3}.$$

Since the logarithm and the exponent are strictly increasing functions, they preserve inequalities. Thus, we have that

$$\prod_{i=0}^{\infty}\left(1-\frac{1}{2^{i+1}}\right) > e^{-\frac{4}{3}} > 2^{-2}$$

and

$$\prod_{i=0}^{n-1}(2^{i+1}-1) > 2^{\frac{n(n+1)}{2}}\prod_{i=0}^{\infty}\left(1-\frac{1}{2^{i+1}}\right) > 2^{\frac{n(n+1)}{2}-2},$$

which we needed to show.

□

Corollary 1. *For any $n > 1$ it holds that*

$$2^{-2} < e^{\frac{-4}{3}} < \prod_{i=1}^{n}\left(1-\frac{1}{2^i}\right) < 1$$

Proof.

$$\prod_{i=1}^{n}\left(1-\frac{1}{2^i}\right) = \left(\prod_{i=1}^{n}2^{-i}\right)\prod_{i=1}^{n}(2^i-1) = 2^{-\frac{n(n+1)}{2}}\prod_{j=0}^{n-1}(2^{j+1}-1).$$

From Lemma 7 above we get that

$$2^{-2} = 2^{-\frac{n(n+1)}{2}}2^{\frac{n(n+1)}{2}-2} < 2^{-\frac{n(n+1)}{2}}\prod_{j=0}^{n-1}(2^{j+1}-1) < 2^{-\frac{n(n+1)}{2}}2^{\frac{n(n+1)}{2}} = 1.$$

□

Lemma 8. *Let $B \in (\mathbb{F}_2)^{k\times r}$. Let $\mathsf{rank}(B) = u$. Let B' be a random $v-k\times r$ binary matrix, such that $r < v$. The probability that $\begin{pmatrix}B\\B'\end{pmatrix}$ is full-rank is no less than*

$$\prod_{i=0}^{r-1}(1-2^{i-v+k-u}).$$

Proof. Let us denote the probability that the matrix $\begin{pmatrix}B\\B'\end{pmatrix}$ is full-rank by $P(B,u,r,v-k)$. We note that what matters in this probability are u, r and $v-k$. If there are two matrices $B_1 \in (\mathbb{F}_2)^{k_1\times r}$ and $B_2 \in (\mathbb{F}_2)^{k_2\times r}$ but their rank is the same, then, $P(B_1,u,r,v-k) = P(B_2,u,r,v-k)$. Indeed, in both cases we are looking the probability of when we add $v-k$ random binary vectors of length r to a subspace of size 2^u, whether we obtain the full space. We also note that by symmetry, the probability is the same for two matrices B_1 and B_2 of the same rank and size.

Now, take two matrices $B_1, B_2 \in (\mathbb{F}_2)^{k\times r}$, with $\mathsf{rank}(B_1) < \mathsf{rank}(B_2)$. Clearly $P(B_1,u,r,v-k) \leq P(B_2,u,r,v-k)$. Indeed, let B'' be a submatrix of B_2 formed by $\mathsf{rank}(B_1)$ rows of B_2. By the argument above, $P(B_1,u,r,v-k) = P(B'',u,r,v-k)$. Now, for any B' such that $\begin{pmatrix}B''\\B'\end{pmatrix}$ is full-rank, $\begin{pmatrix}B_2\\B'\end{pmatrix}$ is also full-rank, thus $P(B_2,u,r,v-k) \geq P(B'',u,r,v-k) = P(B_1,u,r,v-k)$.

Let us now count all the full-rank matrices of size $v\times r$. We know that there are $2^{vr}\prod_{i=0}^{r-1}(1-2^{i-v})$ full-rank matrices of such size.

Now let us classify those matrices according to the first k rows. Because of the inequality above, it follows that the classes that correspond to those where the first k rows are full-rank cannot have less members than those where the first k rows are not full-rank. Thus, we have that for every such B with rank k, there are at least $2^{-kr}(2^{vr}\prod_{i=0}^{r-1}(1-2^{i-v}))$ full-rank matrices. Since there are $2^{(v-k)r}$ options for B', in this case, the probability that the matrix is full-rank is no less than $\frac{2^{(v-k)r}\prod_{i=0}^{r-1}(1-2^{i-v})}{2^{(v-k)r}} = \prod_{i=0}^{r-1}(1-2^{i-v})$.

Now, if the rank of B is u which is smaller than k, then, because $P(B_1,u,r,v-k) = P(B_2,u,r,v-k)$, this probability is actually the same as the case where instead of k we have u, and instead of v we have $v-k+u$. We thus get the probability bound $\prod_{i=0}^{r-1}(1-2^{i-v+k-u})$. □

Lemma 9. *Let $A \in (\mathbb{F}_2)^{r\times v}$ be full-rank and $\vec{b} \in \mathbb{F}_{2^k}^r$ and let $\mathsf{rank}_2(\vec{b}) = u$. Let $v > r+k$. Let $\vec{d}$ be a random solution to the system of linear equations $Ax = \vec{b}$. Let $B = \mathsf{Bits}(\vec{b})$.*

Then the probability that $\vec{d}$ is full-rank is equal to the probability $\Pr[\mathsf{rank}(\begin{pmatrix} B \\ D_2 \end{pmatrix}) = k | D_2 \xleftarrow{\$} (\mathbb{F}_2)^{k \times v-r}]$*, which in turn is at least* $\prod_{i=0}^{r-1}(1-2^{i-v+k-u})$.

Proof. Let $B = \mathsf{Bits}(\vec{b})$. Let us count the binary matrices D such that $AD = B$ and D is full-rank. We note that we can express A as $A = \begin{pmatrix} I_r & 0 \end{pmatrix} Z_A$, where Z_A is some invertible matrix. Writing $Z_A D$ out as $\begin{pmatrix} D_1 \\ D_2 \end{pmatrix}$, we get that

$$AD = \begin{pmatrix} I_r & 0 \end{pmatrix} Z_A D = \begin{pmatrix} I_k & 0 \end{pmatrix} \begin{pmatrix} D_1 \\ D_2 \end{pmatrix} = D_1.$$

Thus, we have that D is a solution if and only if $Z_A D = \begin{pmatrix} B \\ D_2 \end{pmatrix}$, that is, $D = Z_A^{-1} \begin{pmatrix} B \\ D_2 \end{pmatrix}$ where D_2 is arbitrary. We note that D is full-rank if and only if $\begin{pmatrix} B \\ D_2 \end{pmatrix}$ is full-rank. By Lemma 8, the result follows. □

Lemma 10. *Let $v > k, r$. Let $A \in (\mathbb{F}_2)^{r \times v}$ be full-rank. Choose at random $\vec{b} \xleftarrow{\$} (\mathbb{F}_{2^k})^r$. Choose at random $\alpha \in (\mathbb{F}_{2^k})^v$ as a solution for the equation $Ax = \vec{b}$. Then the probability that $\mathsf{rank}_2(\alpha) = k$ is $\prod_{i=0}^{k-1}(1-2^{i-v})$, and all full-rank α are equally likely.*

Proof. For any particular choice of $\vec{b}$ with $B = \mathsf{Bits}(\vec{b})$, we can see that $A\alpha = \vec{b}$ if and only if $D = Z_A^{-1} \begin{pmatrix} B \\ D_2 \end{pmatrix}$ where D_2 is arbitrary. Indeed, writing $Z_A D$ out as $\begin{pmatrix} B \\ D_2 \end{pmatrix}$, we get that

$$AD = \begin{pmatrix} I_r & 0 \end{pmatrix} Z_A D = \begin{pmatrix} I_r & 0 \end{pmatrix} \begin{pmatrix} B \\ D_2 \end{pmatrix} = B.$$

This means that any D, that is α, is uniquely determined by first choosing the B and then the D_2. We note that D is full-rank if and only if the randomly chosen binary matrix $\begin{pmatrix} B \\ D_2 \end{pmatrix}$ is full-rank, which happens with probability $\prod_{i=0}^{k-1}(1-2^{i-v})$. For the other part of the claim, we note that every full-rank matrix $\begin{pmatrix} B \\ D_2 \end{pmatrix}$ is equally likely and to every $\begin{pmatrix} B \\ D_2 \end{pmatrix}$ corresponds exactly one α, since Z_A is invertible. □

5 Construction of SLAMP-FSS

A function secret sharing scheme consists of two algorithms – the key generation Gen and the evaluation Eval. Here we define these algorithms for SLAMP-FSS and ($\mathsf{GenRand}$, $\mathsf{EvalRand}$) for SLAMPR-FSS. SLAMP-FSS takes as an input a multi-point function $P_{\vec{\alpha},\vec{\beta}}$ where we require that for $i, j \in \{1, \ldots, t\}$ if $i \neq j$, then $\alpha_i \neq \alpha_j$. To evaluate $P_{\vec{\alpha},\vec{\beta}}$ on x, both parties traverse the binary trees defined by the keys given by Gen using the input bit-by-bit. This means that, if we are at some node and the next bit is 0, then we move to the left child and if the next bit is 1, then we move right. In the Eval and $\mathsf{EvalRand}$ algorithms, each such step except the last uses one PRG call and some other cheaper operations. The last step does not require any PRG calls.

5.1 Core ideas

This section explains the main ideas used to develop SLAMP-FSS. Our scheme is based on the tree-based approach to distributed point functions. Essentially, each function can

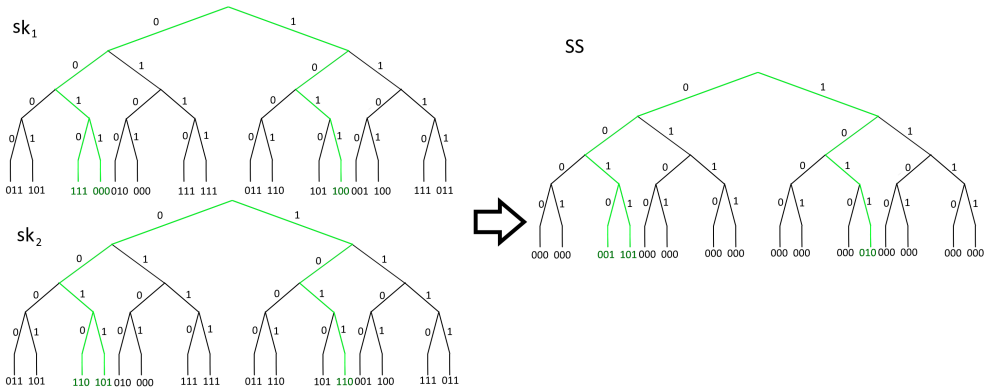


Figure 2. Example of SLAMP-FSS. sk_i shows the trees that can be evaluated using the given key where the value at the leaf becomes the output share of the respective party. SS shows the values encoded in the secret shared tree.

be described as a binary tree and function evaluation is then traversing the tree each time choosing the branch that corresponds to the next bit in the input value α. The leaves of the tree are labelled with the output values. For a distributed multi-point function, the leaves corresponding to traversing the tree based on α_i have the value β_i. The tree structure can be seen on the right in Figure 2 where the green shows the nonzero evaluation paths.

5.1.1 Active and inactive paths

If a path coincides with a path obtained by traversing the tree based on some α_j in $P_{\vec{\alpha},\vec{\beta}}$, then we call that path **an active path**. Every other path from root to leaf is called **an inactive path**. We call nodes that are on some active path **active nodes** and other nodes **inactive nodes**. The goal is to have a shared secret of 0 at the inactive nodes and a nonzero shared secret at active nodes. Our active nodes correspond to accepting paths and inactive nodes to the non-accepting paths in [BGH+25].

For example, in Figure 2, active paths and active nodes are coloured green and inactive nodes are coloured black. The 3-multi-point function from the function family $\mathcal{F}(\{0,1\}^4 \to \mathbb{F}_{2^3})$ is defined by points $(0010, 001)$, $(0011, 101)$ and $(1011, 010)$. The left part shows the full evaluation results for keys sk_1 and sk_2, hence, $\mathcal{P}_i$ could compute the values in the leaves using sk_i. However, during evaluation, the parties can not distinguish active paths from inactive paths. The true leaf values are only known when the two outputs are combined by XOR-ing the values that the parties have on their respective leaves. This output is shown as the tree labeled SS.

Note that every active node has to have at least one active child, but it can have two. Also note that there are at most t active nodes at each level of the binary tree. Inactive nodes cannot have any active children, therefore, our construction must preserve the inactiveness of nodes.

5.1.2 Key points of the construction

During the evaluation, both parties compute some values for each of the nodes that they traverse. Similarly to [BGI16], the key part we use in accomplishing that the children of inactive nodes are inactive is that we are working over a field of characteristic 2. The goal of $\mathsf{GenRand}$ is to generate the keys such that for inactive nodes, both parties compute the same values despite having different keys.

During evaluation, both parties apply the same deterministic operations. Thus, if for

some node, their values become equal (i.e., corresponding to a secret-sharing of 0), the values at all of the descendants of that node will also stay equal. In total, the leaf values will also become equal and correspond to the desired zero output.

The main effort in key generation in GenRand is directed at the active nodes and generating keys that distinguish the active and inactive children or active nodes. The goal is to ensure that the inactive children of the active nodes result in both parties learning the same value for the respective node and ensuring this in a way that does not leak which paths are active. These conditions can be described as a set of linear equations. We note that it suffices if the values at the leaf nodes are a collection of bits and one field element. This allows the output of the PRG to be small, thus obtaining a reasonable comparison with other techniques (see Section 7).

5.1.3 From random to desired output

We modularize our construction by first building a *random* multi-point distributed point function for $P_{\vec{\alpha}}$ that outputs values in $\mathbb{F}_{2^{\bar{v}}}$ as a helper function. This makes up SLAMPR-FSS. After that we give helper functions DeRand in Algorithm 2 and DeRandEval in Algorithm 7 that take the output of a random-DMPF and compile it into the usual DMPF for desired output values $(\beta_1, \ldots, \beta_t)$. This tool can also be used for other constructions of random-DMPF not just SLAMPR-FSS. We note that, although in our current version, the output of a random-DMPF has to be longer than the output of the DMPF, if the desired output vector $\vec{\beta}$ has full rank when viewed as a binary matrix, then the output of a random-DMPF does not have to be longer. We note that vector $\vec{\beta}$ has full-rank with an overwhelming probability for some applications of the DMPF. For example, in PCG constructions, as there the values of active nodes hold parts of correlated randomness — since the nonzero values of one instance of a DMPF are just uniformly random, they are full-rank with an overwhelming probability.

We take the modular approach because the last step of the algorithm requires slightly weaker properties than the other steps. However, in the full evaluation, since the number of operations doubles at every step (layer of the binary tree), the last step dominates the cost of the evaluation, hence, one should make it as cheap as possible. This split into two allows us to omit a PRG call in the last step, thus halving the total number of PRG calls. On the other hand, the output of the PRG has to be slightly larger in the penultimate round.

5.2 Key generation algorithm

This section gives the details of the key generation algorithms of SLAMP-FSS and SLAMPR-FSS in Algorithm 3 and Algorithm 1, respectively. These construction use two versions of PRGs: $f\colon \mathbb{F}_{2^k} \to \{0,1\}^v \times \mathbb{F}_{2^k}$ and $f'\colon \mathbb{F}_{2^k} \to \{0,1\}^{\bar{v}} \times \mathbb{F}_{2^{\bar{v}}}$. The set R_i is the set of active nodes in layer i of the binary tree, with at most t nodes at each layer for t-multi-point function. We also use a statistical security parameter κ. In the security proofs, we require $v > t + \kappa$ and $\bar{v} > k + t + \kappa + 2$ where $k \geq \kappa$.

Our construction in Algorithm 1 starts from generating secret shared values for the root of the tree — a shared binary vector $[X_\epsilon]$ and $[\tau_\epsilon] \in \mathbb{F}_{2^k}$. This is the core part of the key that the parties learn.

The rest of the key is containing the correction values for the levels of the tree to differentiate between the active and inactive children of the active nodes. For every layer j, we construct two local operators $F_{j,0}$ and $F_{j,1}$. The parties evaluate SLAMP-FSS on input $r = r^1 r^2 \ldots r^m$ by computing $[X_r], [\tau_r] = F_{m,r^m}(F_{m-1,r^{m-1}}(\ldots F_{1,r^1}([X_\epsilon], [\tau_\epsilon]) \ldots))$. We are keeping the invariant that at active nodes r, X_r, τ_r are nonzero but at inactive nodes, X_r, τ_r are zero. The operators are built in such a way that if $[X_r]_1 = [X_r]_2$ and $[\tau_r]_1 = [\tau_r]_2$, then also all of their descendants are equal for both evaluators. This is

obtained by ensuring that all the operators deterministic and the same for both parties. Consequently, if the shares become the same value, they will stay the same, and over fields of characteristic 2, the shares being the same is equivalent to the shared value being 0.

Our main effort goes to keeping track of the at most t active nodes at each layer of the tree. They have at most $2t$ children, of which, at most t are alive. The operators are constructed in such a way that map at most t nonzero values $[X_r], [\tau_r]$ to zero while mapping other values to nonzeroes.

More specifically, the operators $F_{j,b}([X_r]_\mathtt{p}, [\tau_r]_\mathtt{p})$ are defined as $f\left(\left\langle [X_r]_\mathtt{p}, \vec{d}_{j-1}\right\rangle + [\tau_r]_\mathtt{p} w_{j,b}\right)$, for $\mathtt{p} \in \{1, 2\}$. For every layer j, there are $t' \leq t$ active nodes $r_{j,1}, \ldots, r_{j,t'}$. We construct the $\vec{d}_j$ and $w_{j,0}$, $w_{j,1}$ in such a way that for all inactive children of active nodes, that we denote as $r_{j,i}\|b$, it holds that $\left\langle [X_{r_{j,i}}], \vec{d}_{j-1}\right\rangle + [\tau_{r_{j,i}}] w_{j,b} = 0$, but that for the active children of active nodes $r_{j,i}\|b$, it holds that $\left\langle [X_{r_{j,i}}], \vec{d}_{j-1}\right\rangle + [\tau_{r_{j,i}}] w_{j,b} \neq 0$. Finding $\vec{d}_j$ and $w_{j,0}$, $w_{j,1}$ that satisfy these restrictions is equivalent to solving a system of linear equations where the coefficients are binary. Note that the last layer is slightly different for efficiency and modularity reasons.

More specifically, the key generation algorithm Gen (formalised in Algorithm 3, using Algorithm 1, Algorithm 4 and Algorithm 2) works as follows:

1. Sample X_ϵ from $\{0,1\}^v \setminus \{(0, 0, \ldots, 0)\}$ and τ_ϵ from $\mathbb{F}_{2^k} \setminus \{0\}$. Interpret X_ϵ as a vector of elements from $\mathbb{F}_{2^k}$ and sample shares of X_ϵ over $\{0,1\}^v \subset \mathbb{F}_{2^k}^v$ and τ_ϵ over $\mathbb{F}_{2^k}$ for $\mathcal{P}_1$ and $\mathcal{P}_2$. We thus obtain a secret-shared tuple of total bitlength $v + k$ with the shares of the first v elements being over $\mathbb{F}_2$ and the shares of the last element being over $\mathbb{F}_{2^k}$. Add the empty string, denoted by ϵ, to the set R_0 as the root is always active.

2. For each layer $i \in \{1, \ldots, n-1\}$ of the binary tree, do the following:

   (a) Randomly sample two different values $w_{i,0}$ and $w_{i,1}$ from $\mathbb{F}_{2^k}$. Their difference ensures that active nodes with exactly one inactive child can deactivate only the inactive child. These are analogues of the correction words.

   (b) Iterate over the set R_{i-1} of active nodes from the previous layer and separate its nodes' children $r\|b$ into active nodes R_i and inactive nodes R'_i. Here $r \in \{0,1\}^{i-1}$ and $b \in \{0,1\}$. Collect r with two active children into $\widehat{R}_{i-1}$.

   (c) For each inactive node $r\|b$ in R'_i, add a constraint $\left\langle X_r, \vec{d}_{i-1}\right\rangle = \tau_r \cdot w_{i,b}$ to a system of linear equations. This deactivates this inactive node (that is, makes $X_{r\|b} = 0$ and $\tau_{r\|b} = 0$ in the evaluation, and thus the respective shares of the parties equal). We note that here $\vec{d}_{i-1}$ denotes the variables in the linear system of equations that we obtain.

   (d) For each active node r in $\widehat{R}_{i-1}$, add a constraint $\left\langle X_r, \vec{d}_{i-1}\right\rangle = \tau_r \cdot w_{r,2}$ to the system of linear equations, where $w_{r,2} \in \mathbb{F}_{2^k}$ is not equal to $w_{i,0}$ nor $w_{i,1}$. This ensures that, with high probability, active nodes that have an active sibling are not deactivated by accident. (Note that $|R'_i| + |\widehat{R}_{i-1}| = |R_i|$.)
   In fact, if this constraint were omitted, then there would be some (small) probability that for an active node r with two active children it would hold that $\left\langle X_r, \vec{d}_{i-1}\right\rangle = \tau_r \cdot w_{i,b}$ for some $b \in \{0,1\}$. Then $r\|b$ will be unintentionally deactivated, as this is the exact condition used to deactivate nodes. Omitting this constraint will save some time but will increase the probability of error and will require changing the error probability in Item 2f.

(e) Test, whether the matrix describing the system of equations is full-rank, and, if not, return $\perp$. This ensures that the system of equations is solvable. We note that by the proof of Proposition 1, the probability that this system is not solvable is at most $2^{-\kappa}$, so if $\kappa < v - t$ is large enough, this test can be omitted.

(f) In addition to that, with probability $\xi(|R_{i-1}|, t, v) := 1 - \frac{\prod_{i=0}^{t-1}(1-2^{i-v})}{\prod_{i=0}^{|R_{i-1}|-1}(1-2^{i-v})}$, return $\perp$ and abort.

The goal of this is that the abort probability would not depend on the $|R_{i-1}|$ and thus on the topology of the tree, which could leak some information about the $\vec{\alpha}$. Especially, the size of R_{i-1} contains the active nodes which gives information about how the values in α are distributed (e.g. where do we branch in the tree).

(g) Solve the system of linear equations for $\vec{d}_{i-1} \in \mathbb{F}_{2^k}^v$. Since there are at most t rows in the system of linear equations and $v > t$ variables, and the matrix corresponding to the system of equations is full-rank, then we know there are a lot of solutions. Choose a $\vec{d}_{i-1}$ that is full-rank. By Lemma 10, we know that the majority of the solutions are full-rank, so the expected number of tries is very close to 1.

(h) For each active node $r\|b$ in R_i, if $i < n-1$, set the secret shares

$$([X_{r\|b}]_\mathtt{p}, [\tau_{r\|b}]_\mathtt{p}) \leftarrow f\left(\left\langle [X_r]_\mathtt{p}, \vec{d}_{i-1}\right\rangle + [\tau_r]_\mathtt{p} \cdot w_{i,b}\right), \quad \mathtt{p} \in \{1,2\},$$
$$(X_{r\|b}, \tau_{r\|b}) \leftarrow ([X_{r\|b}]_1, [\tau_{r\|b}]_1) + ([X_{r\|b}]_2, [\tau_{r\|b}]_2).$$

If $\tau_{r\|b} = 0$, output $\perp$. This is because, in this case, the active child would also be deactivated in the next level. Note that we do not have to do this operation for inactive nodes, because at each iteration we are only using the secret shares of the values in the active nodes. The construction guarantees that the children of inactive nodes are automatically inactive. For $i = n-1$, we do almost the same, but instead of f, we use a different PRG f'. Note that $f\colon \mathbb{F}_{2^k} \to \{0,1\}^v \times \mathbb{F}_{2^k}$ and $f'\colon \mathbb{F}_{2^k} \to \{0,1\}^{\bar{v}} \times \mathbb{F}_{2^{\bar{v}}}$.

3. For $i = n$, perform the steps 2a-2g as above, but in $\mathbb{F}_{2^{\bar{v}}}$ instead of $\mathbb{F}_{2^k}$ and the length of X_r is $\bar{v}$ instead of v; if $\mathsf{rank}_2(\vec{b}) < t$, return $\perp$; we sample $\vec{d}_{n-1} \in \mathbb{F}_{2^{\bar{v}}}^{\bar{v}}$ on the condition that $\mathsf{rank}_2(\vec{d}_{n-1}) = \bar{v}$; and instead of step 2h, we set

$$[z_{r\|b}]_\mathtt{p} \leftarrow \left\langle [X_r]_\mathtt{p}, \vec{d}_{i-1}\right\rangle + [\tau_r]_\mathtt{p} \cdot w_{i,b}, \quad \mathtt{p} \in \{1,2\}$$
$$z_{r\|b} \leftarrow [z_{r\|b}]_1 + [z_{r\|b}]_2$$

If $\mathsf{rank}_2(A) = \mathsf{rank}_2(\vec{b}) = t$, then the proportion of $\vec{d}_{n-1}$ where $A\vec{d}_{n-1} = \vec{b}$ and $\mathsf{rank}_2(\vec{d}_{n-1}) = \bar{v}$ is at least $e^{-\frac{4}{3}} > \frac{1}{4}$ among all the solutions (as shown in the Lemma 4). Hence, we will always find a full-rank solution for $\vec{d}_{n-1}$ for any full-rank pair $A, \vec{b}$ and it takes, in expectation, no more than 4 tries.

4. Set $([X_\epsilon]_1, [\tau_\epsilon]_1, \{w_{i,0}\}_{i=1}^n, \{w_{i,1}\}_{i=1}^n, \{\vec{d}_i\}_{i=0}^{n-1})$ as the first key sk_1 and $([X_\epsilon]_2, [\tau_\epsilon]_2, \{w_{i,0}\}_{i=1}^n, \{w_{i,1}\}_{i=1}^n, \{\vec{d}_i\}_{i=0}^{n-1})$ as the second key sk_2.

5. In layer n, we store the intermediate values $\{\alpha_i\}_{i=1}^t$, $\{[z_{\alpha_i}]_1\}_{i=1}^t$, $\{[z_{\alpha_i}]_2\}_{i=1}^t$, $\{z_{\alpha_i}\}_{i=1}^t$ in a container called state. Note that R_n contains exactly $\{\alpha_i\}_{i=1}^t$. This is the final state for the key generation.

6. At this point we have $\mathsf{GenRand}$ (Algorithm 1) for a random-DMPF. For SLAMPR-FSS, it is sufficient to stop here and to give sk_i to $\mathcal{P}_i$.

Algorithm 1: GenRand – Key generator for SLAMPR-FSS

Input : $\vec{\alpha} = (\alpha_1, \ldots, \alpha_t)$ where $\alpha_i \in \{0,1\}^n$

$[X_\epsilon]_1 \xleftarrow{\$} \{0,1\}^v$, here interpret $[X_\epsilon]_1 \in \mathbb{F}_{2^k}^v$

$[X_\epsilon]_2 \xleftarrow{\$} \{0,1\}^v \setminus \{[X_\epsilon]_1\}$, here interpret $[X_\epsilon]_2 \in \mathbb{F}_{2^k}^v$

$X_\epsilon \leftarrow [X_\epsilon]_1 + [X_\epsilon]_2$

$[\tau_\epsilon]_1 \xleftarrow{\$} \mathbb{F}_{2^k}$

$[\tau_\epsilon]_2 \xleftarrow{\$} \mathbb{F}_{2^k} \setminus \{[\tau_\epsilon]_1\}$

$\tau_\epsilon \leftarrow [\tau_\epsilon]_1 + [\tau_\epsilon]_2$

$R_0 \leftarrow \{\epsilon\}$, where ϵ is an empty string

$\mathsf{state} \leftarrow \{R_0, [X_\epsilon]_1, [X_\epsilon]_2, X_\epsilon, [\tau_\epsilon]_1, [\tau_\epsilon]_2, \tau_\epsilon\}$

for $i = 1$ **to** n **do**

  $w_{i,0}, w_{i,1}, \vec{d}_{i-1}, \mathsf{state} \leftarrow \mathsf{ProcessLayer}(i, \vec{\alpha}, \mathsf{state})$

$\mathsf{sk}_1 \leftarrow [X_\epsilon]_1, [\tau_\epsilon]_1, \{w_{i,0}\}_{i=1}^n, \{w_{i,1}\}_{i=1}^n, \{\vec{d}_i\}_{i=0}^{n-1}$

$\mathsf{sk}_2 \leftarrow [X_\epsilon]_2, [\tau_\epsilon]_2, \{w_{i,0}\}_{i=1}^n, \{w_{i,1}\}_{i=1}^n, \{\vec{d}_i\}_{i=0}^{n-1}$

return $\mathsf{state}, \mathsf{sk}_1, \mathsf{sk}_2$

Algorithm 2: DeRand – Derandomiser of Gen

Input : $\{[z_r]_1\}_{r \in R_n}, \{[z_r]_2\}_{r \in R_n}, \vec{\alpha} = (\alpha_1, \ldots, \alpha_t), \vec{\beta} = (\beta_1, \ldots, \beta_t)$ where $|R_n| = t$, $[z_r]_\mathsf{p} \in \{0,1\}^{\bar{v}}$ for $\mathsf{p} \in \{1,2\}$, $\alpha_i \in \{0,1\}^n$ and $\beta_i \in \mathbb{F}_{2^m}$

for $i = 1$ **to** t **do**

  $\{[z_{i,j}]_1\}_{j=1}^{\bar{v}} \leftarrow \mathsf{Bits}([z_{\alpha_i}]_1)$

  $\{[z_{i,j}]_2\}_{j=1}^{\bar{v}} \leftarrow \mathsf{Bits}([z_{\alpha_i}]_2)$

for $i = 1$ **to** t **do**

  for $j = 1$ **to** $\bar{v}$ **do**

    $z_{i,j} \leftarrow [z_{i,j}]_1 + [z_{i,j}]_2$

$Z \leftarrow \{z_{i,j}\}_{i=1,j=1}^{t,\bar{v}}$

if $\mathsf{rank}_2(Z) < t$ **then**

  return $\perp$

Solve $Z\vec{g} = \vec{\beta}$ and sample $\vec{g}$ from the solution space

return $\vec{g}$

7. For SLAMP-FSS we want fixed outputs $\vec{\beta} = (\beta_1, \ldots, \beta_t)$ at inputs $(\alpha_1, \ldots, \alpha_t)$. Toward this goal, one can proceed as is described in Gen for SLAMP-FSS (Algorithm 3) and call DeRand (Algorithm 2). In this case, we convert all the random $[z_r]_1$ and $[z_r]_2$ to bits for all $r \in (\alpha_1, \ldots, \alpha_t)$. We set $Z \leftarrow \{[z_{i,j}]_1 + [z_{i,j}]_2\}_{i=1,j=1}^{t,\bar{v}}$. The values $[z_r]_1$ and $[z_r]_2$ are passed to Algorithm 2 inside the state.

8. Now we solve the system of equations $Z\vec{g} = \vec{\beta}$ where $\vec{g}$ is the unknown and randomly a select a solution if $\mathsf{rank}_2(Z) < t$, otherwise return $\perp$. We note that, by Lemma 5, if $\mathsf{rank}_2(Z) = t$ (which also means $t = \mathsf{rank}_2(Z|\vec{\beta})$), then there exist solutions to the system of equations $Z\vec{g} = \vec{\beta}$.

9. If at any point the algorithm returns $\perp$, we restart the algorithm.

10. Now we give to party $\mathcal{P}_i$ the value sk_i along with $\vec{g}$.

Algorithm 3: Gen – Key generation for SLAMP-FSS
Input : $(\alpha_1, \beta_1), \ldots, (\alpha_t, \beta_t)$ where $\alpha_i \in \{0,1\}^n$ and $\beta_i \in \mathbb{F}_{2^m}$
$\mathsf{state}, \mathsf{sk}_1, \mathsf{sk}_2 \leftarrow \mathsf{GenRand}((\alpha_1, \ldots, \alpha_t))$
Parse state as $R_n, \{[z_r]_1\}_{r \in R_n}, \{[z_r]_2\}_{r \in R_n}, \{z_r\}_{r \in R_n}$
$\vec{g} \leftarrow \mathsf{DeRand}(\{[z_r]_1\}_{r \in R_n}, \{[z_r]_2\}_{r \in R_n}, (\alpha_1, \ldots, \alpha_t), (\beta_1, \ldots, \beta_t))$
$\mathsf{sk}_1 \leftarrow \mathsf{sk}_1, \vec{g}$
$\mathsf{sk}_2 \leftarrow \mathsf{sk}_2, \vec{g}$
If $\perp$ was output by $\mathsf{GenRand}$ or DeRand, start from the beginning
return sk_1, sk_2

5.3 Evaluation algorithm

During evaluation, both parties independently use their key sk_p and a common input x to get secret shared outputs for $P_{\vec{\alpha},\vec{\beta}}(x)$ or $P_{\vec{\alpha}}(x)$. In short, for each layer, they expand their seed using a PRG, then pick the correction value according to the input bit and apply it to obtain the seed for the next level.

To evaluate SLAMPR-FSS at a bitstring $x = x^1 x^2 \ldots x^n$, we compute $[X_x], [\tau_x] = F_{n,x^n}(F_{n-1,x^{n-1}}(\ldots F_{1,x^1}([X_\epsilon]_\mathsf{p}, [\tau_\epsilon]_\mathsf{p}) \ldots))$. The operators $F_{j,b}$ for $b \in \{0,1\}$ are as described in the previous section and the concrete details of the evaluation $\mathsf{EvalRand}$ are in Algorithm 6. To obtain the evaluation phase of SLAMP-FSS, $\mathsf{EvalRand}$ outputs are passed to $\mathsf{DeRandEval}$ (Algorithm 7 which maps the random outputs of $\mathsf{EvalRand}$ to the desired output values using some linear algebra as detailed in Eval in Algorithm 5. This means that active nodes r, the values that the party $\mathcal{P}$ obtains are exactly the same as the $[X_r]_\mathsf{p}, [\tau_r]_\mathsf{p}$ that were computed in Gen, since exactly the same operators are applied to the same inputs. Inactive nodes are such that the shares of the parties are equal by construction.

More specifically, the evaluation for party $\mathsf{p} \in \{1,2\}$ (Algorithm 5) on evaluation input $x = x^1 x^2 \ldots x^n \in \{0,1\}^n$ and key sk_p works as follows.

1. Set $([\bar{X}_0]_\mathsf{p}, [\bar{\tau}_0]_\mathsf{p}) \leftarrow ([X_\epsilon]_\mathsf{p}, [\tau_\epsilon]_\mathsf{p})$ where the latter is part of sk_p.

2. Iterate over $i \in \{1, \ldots, n\}$ to consider all inputs bits. Do the following:

   (a) Calculate $z_i \leftarrow \left\langle [\bar{X}_{i-1}]_\mathsf{p}, \vec{d}_{i-1} \right\rangle + [\bar{\tau}_{i-1}]_\mathsf{p} \cdot w_{i,x^i}$.

   (b) If $i < n-1$, evaluate the PRG f as $([\bar{X}_i]_\mathsf{p}, [\bar{\tau}_i]_\mathsf{p}) \leftarrow f([z_i]_\mathsf{p})$.
   If $i = n-1$, evaluate the PRG f' as $([\bar{X}_i]_\mathsf{p}, [\bar{\tau}_i]_\mathsf{p}) \leftarrow f'([z_i]_\mathsf{p})$.
   If $i = n$, then no PRG is needed, so set $[z]_\mathsf{p} \leftarrow z_n$.
   Note that, if $r \| x^i := x^1 x^2 \ldots x^i$ is active, then $([\bar{X}_i]_\mathsf{p}, [\bar{\tau}_i]_\mathsf{p}) = ([X_{r\|x^i}]_\mathsf{p}, [\tau_{r\|x^i}]_\mathsf{p})$ where $[X_{r\|x^i}]_\mathsf{p}, [\tau_{r\|x^i}]_\mathsf{p}$ were generated in the key generation algorithm. If $r\|x^i$ is an inactive node, then r is either active or inactive. If r is active then from $\left\langle X_r, \vec{d}_{i-1} \right\rangle = \tau_r \cdot w_{i,x^i}$, we can deduce that $\left\langle [X_r]_1, \vec{d}_{i-1} \right\rangle + [\tau_r]_1 \cdot w_{i,x^i} = \left\langle [X_r]_2, \vec{d}_{i-1} \right\rangle + [\tau_r]_2 \cdot w_{i,x^i}$. This means both parties use the same argument in f (or f'). As f (or f') is deterministic, both parties have the same value as their secret share. Hence, the shared secret is 0 as required for correctness. Both secret shares will stay the same until the end of the protocol, as the evaluation algorithm continues exactly the same for both parties. If r is inactive, then we know that some ancestor of r must be active and it is already ensured that parties have the same shared for r.
   This concludes evaluation $\mathsf{EvalRand}$ for SLAMPR-FSS in Algorithm 6.

Algorithm 4: ProcessLayer – Subroutine for processing layers in the binary tree of GenRand (Algorithm 1)

Input : $i \in \{1,\dots,n\}$, $\vec{\alpha} = (\alpha_1,\dots,\alpha_t)$ where $\alpha_j \in \{0,1\}^n$, $\mathsf{state} = \{R_{i-1}, \{([X_r]_1, [X_r]_2, X_r, [\tau_r]_1, [\tau_r]_2, \tau_r) \mid r \in R_{i-1}\}\}$ where $R_0 = \{\epsilon\}$ and $R_{i-1} \subset \{0,1\}^{i-1}$ for $i > 1$, $[X_r]_\mathtt{p} \in \{0,1\}^v \subset \mathbb{F}_{2^k}^v$, $X_r \in \{0,1\}^v \subset \mathbb{F}_{2^k}^v$, $\tau_r \in \mathbb{F}_{2^k}$ and $[\tau_r]_\mathtt{p} \in \mathbb{F}_{2^k}$ for $\mathtt{p} \in \{1,2\}$ (for $i = n$, $[X_r]_\mathtt{p} \in \{0,1\}^{\bar{v}} \subset \mathbb{F}_{2^k}^{\bar{v}}$, $X_r \in \{0,1\}^{\bar{v}} \subset \mathbb{F}_{2^k}^{\bar{v}}$)

if $i < n$ **then** $\mathbb{F} \leftarrow \mathbb{F}_{2^k}$
else $\mathbb{F} \leftarrow \mathbb{F}_{2^{\bar{v}}}$
$w_{i,0} \xleftarrow{\$} \mathbb{F} \setminus \{0\}$, $w_{i,1} \xleftarrow{\$} \mathbb{F} \setminus \{0, w_{i,0}\}$
$R_i, R_i', \widehat{R}_{i-1} \leftarrow \emptyset$
for $r \in R_{i-1}$ **do**
  for $b \in \{0,1\}$ **do**
    if $\exists \alpha_j \in \vec{\alpha}$ *such that* α_j *starts with* $r\|b$ **then** $R_i \leftarrow R_i \cup \{r\|b\}$
    else $R_i' \leftarrow R_i' \cup \{r\|b\}$
  if $r\|0 \in R_i$ *and* $r\|1 \in R_i$ **then** $\widehat{R}_{i-1} \leftarrow \widehat{R}_{i-1} \cup \{r\}$
$A \leftarrow \mathsf{Mat}_{R_{i-1},v}(0)(0)$, $\vec{b} \leftarrow \mathsf{Vec}_{|R_{i-1}|}(0)$, and $j_i \leftarrow 1$
for $r\|b \in R_i'$ **do**
  $A^{(j_i)} \leftarrow X_r$
  $\vec{b}_j \leftarrow \tau_r \cdot w_{i,b}$
  $j_i \leftarrow j_i + 1$
for $r \in \widehat{R}_{i-1}$ **do**
  $w_{r,2} \xleftarrow{\$} \mathbb{F} \setminus \{w_{i,0}, w_{i,1}, 0\}$
  $A^{(j_i)} \leftarrow X_r$
  $\vec{b}_j \leftarrow \tau_r \cdot w_{r,2}$
  $j_i \leftarrow j_i + 1$
If $\mathsf{rank}(A) < |R_{i-1}|$, **return** $\perp$
With probability $\xi(|R_{i-1}|, t, v)$, **return** $\perp$
Solve $A\vec{d}_{i-1} = \vec{b}$ and sample full-rank $\vec{d}_{i-1}$ from the solution space
If $i = n$, if $\mathsf{rank}_2(\vec{b}) < t$, return $\perp$, else, choose a solution such that $\mathsf{rank}_2(\vec{d}_{i-1}) = \bar{v}$
if $i < n$ **then**
  for $r\|b \in R_i$ **do**
    if $i < n-1$ **then**
      $([X_{r\|b}]_\mathtt{p}, [\tau_{r\|b}]_\mathtt{p}) \leftarrow f\left(\left\langle [X_r]_\mathtt{p}, \vec{d}_{i-1} \right\rangle + [\tau_r]_\mathtt{p} \cdot w_{i,b}\right)$ for $\mathtt{p} \in \{1,2\}$
    else
      $([X_{r\|b}]_\mathtt{p}, [\tau_{r\|b}]_\mathtt{p}) \leftarrow f'\left(\left\langle [X_r]_\mathtt{p}, \vec{d}_{i-1} \right\rangle + [\tau_r]_\mathtt{p} \cdot w_{i,b}\right)$ for $\mathtt{p} \in \{1,2\}$
    $X_{r\|b} \leftarrow [X_{r\|b}]_1 + [X_{r\|b}]_2$
    $\tau_{r\|b} \leftarrow [\tau_{r\|b}]_1 + [\tau_{r\|b}]_2$
    If $\tau_{r\|b} = 0$, **return** $\perp$
  $\mathsf{state} \leftarrow \{R_i, \{([X_r]_1, [X_r]_2, X_r, [\tau_r]_1, [\tau_r]_2, \tau_r) \mid r \in R_i\}\}$
  return $w_{i,0}, w_{i,1}, \vec{d}_{i-1}, \mathsf{state}$;
else
  for $r\|b \in R_i$ **do**
    $[z_{r\|b}]_\mathtt{p} \leftarrow \left\langle [X_r]_\mathtt{p}, \vec{d}_{i-1} \right\rangle + [\tau_r]_\mathtt{p} \cdot w_{i,b}$ for $\mathtt{p} \in \{1,2\}$
    $z_{r\|b} \leftarrow [z_{r\|b}]_1 + [z_{r\|b}]_2$
  $\mathsf{state} \leftarrow R_i, \{[z_r]_1\}_{r\in R_i}, \{[z_r]_2\}_{r\in R_i}, \{z_r\}_{r\in R_i}$
  return $w_{i,0}, w_{i,1}, \vec{d}_{i-1}, \mathsf{state}$;

Algorithm 5: $\mathsf{Eval}^{\mathsf{p}}$ – Evaluation of SLAMP-FSS for party $\mathcal{P}_{\mathsf{p}}$

Input : $\mathsf{sk}_{\mathsf{p}} = ([X_\epsilon]_{\mathsf{p}}, [\tau_\epsilon]_{\mathsf{p}}, \{w_{i,0}\}_{i=1}^n, \{w_{i,1}\}_{i=1}^n, \{\vec{d}_i\}_{i=0}^{n-1}, \vec{g})$, evaluation input $x = x^1 \dots x^n$ where $x^i \in \{0,1\}$ and $[X_\epsilon]_{\mathsf{p}} \in \{0,1\}^v \subset \mathbb{F}_{2^k}^v$, $[\tau_\epsilon]_{\mathsf{p}} \in \mathbb{F}_{2^k}$, $w_{i,b} \in \mathbb{F}_{2^k}$ for $i \leq n$ and $w_{n,b} \in \mathbb{F}_{2^{\bar{v}}}$ for $b \in \{0,1\}$, $\vec{g} = (g_1, \dots, g_{\bar{v}})$ where $g_i \in \mathbb{F}_{2^m}$ and $\vec{d}_i \in \mathbb{F}_{2^k}^v$ for $i < n$, and $\vec{d}_n \in \mathbb{F}_{2^{\bar{v}}}^{\bar{v}}$.

Parse sk_{p} as $(\mathsf{sk}'_{\mathsf{p}}, \vec{g})$
$\{z_i\}_{i=1}^{\bar{v}} \leftarrow \mathsf{EvalRand}^{\mathsf{p}}(\mathsf{sk}'_{\mathsf{p}}, x = x^1 \dots x^n)$
$y \leftarrow \mathsf{DeRandEval}^{\mathsf{p}}(\{z_i\}_{i=1}^{\bar{v}}, \vec{g})$
return y

Algorithm 6: $\mathsf{EvalRand}^{\mathsf{p}}$ – Evaluation of SLAMPR-FSS for party $\mathcal{P}_{\mathsf{p}}$

Input : $\mathsf{sk}_{\mathsf{p}} = ([X_\epsilon]_{\mathsf{p}}, [\tau_\epsilon]_{\mathsf{p}}, \{w_{i,0}\}_{i=1}^n, \{w_{i,1}\}_{i=1}^n, \{\vec{d}_i\}_{i=0}^{n-1})$, evaluation input $x = x^1 \dots x^n$ where $x^i \in \{0,1\}$ $[X_\epsilon]_{\mathsf{p}} \in \{0,1\}^v \subset \mathbb{F}_{2^k}^v$, $[\tau_\epsilon]_{\mathsf{p}} \in \mathbb{F}_{2^k}$, $w_{i,b} \in \mathbb{F}_{2^k}$ for $i \leq n$ and $w_{n,b} \in \mathbb{F}_{2^{\bar{v}}}$ for $b \in \{0,1\}$ and $\vec{d}_i \in \mathbb{F}_{2^k}^v$ for $i < n-1$, and $\vec{d}_{n-1} \in \mathbb{F}_{2^{\bar{v}}}^{\bar{v}}$.

$[X_0]_{\mathsf{p}}, [\tau_0]_{\mathsf{p}}, \{w_{i,0}\}_{i=1}^n, \{w_{i,1}\}_{i=1}^n, \{\vec{d}_i\}_{i=0}^{n-1} \leftarrow \mathsf{sk}_{\mathsf{p}}$
$([\bar{X}_0]_{\mathsf{p}}, [\bar{\tau}_0]_{\mathsf{p}}) \leftarrow ([X_0]_{\mathsf{p}}, [\tau_0]_{\mathsf{p}})$
for $i = 1$ **to** $n-2$ **do**
  $([\bar{X}_i]_{\mathsf{p}}, [\bar{\tau}_i]_{\mathsf{p}}) \leftarrow f\left(\left\langle [\bar{X}_{i-1}]_{\mathsf{p}}, \vec{d}_{i-1} \right\rangle + [\bar{\tau}_{i-1}]_{\mathsf{p}} \cdot w_{i,x^i}\right)$
$([\bar{X}_{n-1}]_{\mathsf{p}}, [\bar{\tau}_{n-1}]_{\mathsf{p}}) \leftarrow f'\left(\left\langle [\bar{X}_{n-2}]_{\mathsf{p}}, \vec{d}_{n-2} \right\rangle + [\bar{\tau}_{n-2}]_{\mathsf{p}} \cdot w_{n-1,x^{n-1}}\right)$
$[z]_{\mathsf{p}} \leftarrow \left\langle [\bar{X}_{n-1}]_{\mathsf{p}}, \vec{d}_{n-1} \right\rangle + [\bar{\tau}_{n-1}]_{\mathsf{p}} \cdot w_{n,x^n}$
return $[z]_{\mathsf{p}}$

3. For evaluation Eval of SLAMP-FSS, we need to replace the random outputs with the desired outputs $\vec{\beta}$. This is achieved by $\mathsf{DeRandEval}^{\mathsf{p}}$ in Algorithm 7. The party takes the bits of their random output and multiplies them with g_i that are part of sk_{p}. This gives the correct result as together the parties take z and compute $\sum_{i=1}^{\bar{v}} z_i g_i$ as the output. If $z \in \mathbb{F}_{2^{\bar{v}}}$ is equal to zero, then also all of its bits are zeroes, and thus $\sum_{i=1}^{\bar{v}} z_i g_i$ is also zero. For nonzero z, the $\vec{g}$ was chosen in DeRand so that the final output would be $\vec{\beta}$.

This algorithm can be easily extended to the full evaluation on all possible inputs. For $\mathsf{FullEval}$, the main thing to notice is that all inputs and therefore the evaluation of the whole tree can be done in parallel, evaluating each level of the tree once.

6 Properties of SLAMP-FSS

In the construction, we associated six values to each node r: $[X_r]_1, [X_r]_2, X_r = [X_r]_1 + [X_r]_2, [\tau_r]_1$, $[\tau_r]_2, \tau_r = [\tau_r]_1 + [\tau_r]_2$. For all active nodes, these are exactly the values

Algorithm 7: $\mathsf{DeRandEval}^{\mathsf{p}}$ – Evaluation derandomization algorithm

Input : $[z]_{\mathsf{p}}, \vec{g} = (g_1, \dots, g_{\bar{v}})$ where $z \in \{0,1\}^{\bar{v}}$ and $g_i \in \mathbb{F}_{2^m}$

$\{z_i\}_{i=1}^{\bar{v}} \leftarrow \mathsf{Bits}([z]_{\mathsf{p}})$
$y \leftarrow \sum_{i=1}^{\bar{v}} z_i g_i$
return y

that are computed in ProcessLayer Algorithm 4 and kept in state. By construction of EvalRand and Eval, we also have that for every active node r party $\mathcal{P}_\mathsf{p}$ has the exact same values $[X_r]_\mathsf{p}, [\tau_r]_\mathsf{p}$ during evaluation, since they performed exactly the same deterministic operations on the same input as in the generation algorithm.

For the proofs, consider the following notation. For $i < n$, denote the value $(X_{r^1r^2\dots r^{i-1}}, \tau_{r^1r^2\dots r^{i-1}})$ as $M_i(r)$. If $i = n$, then $M_i(r) = z_{r^1r^2\dots r^n}$ where z_r is the respective output of GenRand. In a sense, this captures the value at the node and can be modified to the desired output at the leaf. Throughout the construction, $M_i(r)$ is nonzero at precisely the active nodes with an overwhelming probability. For the sake of brevity, we will assume that $M_i(r) = 0$ means $M_i(r) = (\vec{0}, 0)$ for $i < n$ and $M_i(r) = 0$ for $i = n$.

The following properties help us to prove the correctness and security of the DMPF scheme. Each of the properties is formalised and proven in the referenced Proposition or Theorem. Items 1 to 8 are needed to show correctness and Items 9 to 11 relate to the security of SLAMP-FSS.

1. Algorithm 4 ProcessLayer for processing layers in the binary tree has a low probability of error, assuming its inputs are correct, by which we mean that we only consider the correctness if the ProcessLayer algorithm is used so that inputs to the ProcessLayer are as defined by the GenRand algorithm. We do not consider what happens if the ProcessLayer is executed with arbitrary inputs. (Proposition 1)

2. Algorithm 3 Gen has a low probability of outputting $\perp$ assuming its inputs are correct, by which we mean $\vec{\alpha} \in (\{0,1\}^n)^t$ and $\vec{\beta} \in \mathbb{F}_{2^m}^t$. (Proposition 2)

3. If the secrets shared by the parties $\mathcal{P}_1$ and $\mathcal{P}_2$ are [0], that is, $M_i(x) = 0$, then after processing the next symbol of the input using the evaluation algorithm Eval Algorithm 5, the secrets are still [0], that is, $M_{i+1}(x\|0) = M_{i+1}(x\|1) = 0$ (Proposition 3). This ensures that descendants of inactive nodes are inactive.

4. If $M_i(x) \neq 0$, then $x^1x^2\dots x^i = \alpha_j^1\alpha_j^2\dots\alpha_j^i$ for at least one of the α_j in $\vec{\alpha}$ (Proposition 4). This ensures that nonzero nodes are on active paths.

5. For an honest evaluation of the GenRand algorithm Algorithm 1 that outputs $\mathsf{state} = (R, \{[z_r]_1\}_{r\in R}, \{[z_r]_2\}_{r\in R}, \{z_r\}_{r\in R})$ and $\mathsf{sk}_1, \mathsf{sk}_2$; and for honest evaluation of EvalRand Algorithm 6 with input $x \in R$, we have that for $\mathsf{p} \in \{1,2\}$, we have that $\mathsf{EvalRand}^\mathsf{p}(\mathsf{sk}_\mathsf{p}, x) = [z_x]_\mathsf{p}$ and that $[z_x]_1 + [z_x]_2 = z_x$ (Proposition 5). Hence, SLAMPR-FSS is correct for the inputs defined by $P_{\vec{\alpha}}$.

6. If there exists $j \in \{1,\dots,t\}$ such that $x = \alpha_j$, then executing Gen Algorithm 3 and Eval Algorithm 5 produces a shared secret of β_j (Proposition 6). Hence,SLAMP-FSS is correct for the inputs defined by $P_{\vec{\alpha},\vec{\beta}}$.

7. If there does not exist $j \in \{1,\dots,t\}$ such that $x = \alpha_j$, then executing Gen Algorithms 3 and EvalRand Algorithm 6 produces a shared secret of 0 (Proposition 7).

8. SLAMP-FSS and SLAMPR-FSS are correct FSS schemes for multi-point and random multi-point functions, respectively (Theorem 3).

9. SLAMPR-FSS defined by GenRand in Algorithm 1 and EvalRand in Algorithm 6 is $t \cdot n \cdot \varepsilon_{\mathsf{PRG}} + n2^{-k-\kappa}$-close to random (Theorem 4). Hence, SLAMPR-FSS is secure.

10. Combining a ε_r-close to random random-DMPF scheme with our DeRand and DeRandEval algorithms result in a secure FSS scheme (Theorem 5).

11. It is not possible to tell any information about α_j or β_j, for any $j \in \{1,\dots,t\}$, from the viewpoint of one party, that is, SLAMP-FSS is a secure FSS for multi-point functions (Theorem 6).

In order to prove these properties, we assume that we have the ideal block cipher $F : \{0,1\}^{(\delta+1)k} \times \{0,1\}^k \to \{0,1\}^{(\delta+1)k}$. Then we sample a random key $key \in \{0,1\}^{(\delta+1)k}$, fix it, and define an ideal PRG $\hat{f}$ as $F(key, \cdot)$. Since F is sampled randomly from all permutations and also x is uniform, then $\hat{f}$ produces uniformly random elements from $\{0,1\}^{(\delta+1)k}$. We can interpret these as elements of $\{0,1\}^v \times \mathbb{F}_{2^k}$. In the following proofs, we assume that our PRG f behaves like $\hat{f}$. Hence, we model the PRGs in the Ideal Cipher Model (see Section 4.2.7). For our construction, we need that $\bar{v} > t + k + \kappa + 2$ and $v > t + \kappa$. These ensure that the schemes are close to random and that we have reasonable probabilities that the algorithms succeed. If we use 256-bit AES for the block cipher, then $k = 256$ and the block length must be a multiple of 128. Then, for $\kappa \leq 128$ and $t \leq 126$, we can have $v = 256$, with the corresponding $\delta + 1 = 4$, and $\bar{v} = 372$ and the corresponding $\delta + 1 = 5$.

6.1 Correctness

In this subsection, we show SLAMP-FSS is correct, which means that Gen does not abort (or return $\perp$) with overwhelming probability, and after running Eval the parties have secret shared results for the right multi-point function. As SLAMP-FSS uses SLAMPR-FSS as a component, we also show the correctness of SLAMPR-FSS.

Note that the problem with solving systems of linear equations in our construction is that they might not be solvable. Therefore, there is a small chance that ProcessLayer (Algorithm 4) will terminate with an error. Hence, to show correctness, we first show that the probability of an error output in Algorithm 4 is indeed low. For this, we bound the probability that the system of linear equations is not solvable. Note that the final layers are slightly different, hence, we consider these as separate cases in the following theorem.

Proposition 1. *If the values* ProcessLayer *uses are correctly defined and for all active nodes r of the previous level: $(X_r, \tau_r) \neq 0$, then the probability that Algorithm 4 will end with an error is*

$$\Pr[\textit{Algorithm 4 returns } \perp] \leq \frac{1}{2^{v-t}} + t2^{-k}$$

for levels $i < n - 1$,

$$\Pr[\textit{Algorithm 4 returns } \perp] \leq \frac{1}{2^{v-t}} + t2^{-\bar{v}}$$

for $i = n - 1$, and

$$\Pr[\textit{Algorithm 4 returns } \perp] \leq \frac{1}{2^{\bar{v}-t-1}}$$

for $i = n$.

Proof. We note that on levels $i < n$, Algorithm 4 returns $\perp$ if $\mathsf{rank}(A) < |R_{i-1}|$, or if $\tau_{r\|b} = 0$, or if we abort on line 21. We assumed that A and $\tau_{r\|b}$ are given as outputs of an ideal cipher, thus we assume that they are distributed uniformly randomly.

By Lemma 3 the probability that A is full-rank is equal to $\prod_{i=0}^{|R_{i-1}|-1}(1 - 2^{i-v})$. Thus, the probability that A is not full-rank is $1 - \prod_{i=0}^{|R_{i-1}|-1}(1 - 2^{i-v})$. Together with the line 21 the probability of returning $\perp$ becomes then

$$1 - \prod_{i=0}^{|R_{i-1}|-1}(1 - 2^{i-v}) + \prod_{i=0}^{|R_{i-1}|-1}(1 - 2^{i-v}) \cdot \xi(|R_{i-1}|, t, v) =$$
$$= 1 - \prod_{i=0}^{|R_{i-1}|-1}(1 - 2^{i-v}) + \prod_{i=0}^{|R_{i-1}|-1}(1 - 2^{i-v}) \cdot \left(1 - \frac{\prod_{i=0}^{t-1}(1 - 2^{i-v})}{\prod_{i=0}^{|R_{i-1}|-1}(1 - 2^{i-v})}\right) =$$

$$= 1 - \prod_{i=0}^{t-1}(1 - 2^{i-v}),$$

which can be upper-bounded by 2^{t-v} by Lemma 6. The probability that at least one of $|R_i|$ randomly drawn elements from $\mathbb{F}_{2^k}$ is 0, is upper-bounded by $|R_i|\, 2^{-k} \leq t2^{-k}$. On level $i = n-1$, this value is $t2^{-\bar{v}}$, because of the use of f'.

For $i = n$, there is the additional chance of returning $\perp$ if $\mathsf{rank}_2(\vec{b}) < t$. Since $\vec{b}$ is constructed as an output of an ideal cipher multiplied by some nonzero constants, we can assume it is also distributed uniformly at random. On the other hand, both matrices are now $\bar{v} \times t$ binary matrices. Thus, analogously to the case of A, we can upper-bound the probability of the algorithm outputting $\perp$ due to $\vec{b}$ not being full-rank by $2^{t-\bar{v}}$. Thus, in total the bound becomes $2^{t-\bar{v}} + 2^{t-\bar{v}} = 2^{t+1-\bar{v}}$. □

From this error in ProcessLayer we can calculate the total error probability of Gen (Algorithm 3).

Proposition 2. *The probability of* Gen *returning* $\perp$ *is negligible in* $\min\{k, \kappa\}$.

Proof. We note that the algorithm Gen outputs $\perp$ either in ProcessLayer or DeRand. By Proposition 1,

$$\Pr[\text{Algorithm 4 returns } \perp \text{ on any layer}] \leq (n-2)\left(2^{t-v} + t2^{-k}\right) + 2^{t-v} + \\ + t2^{-\bar{v}} + 2^{t+1-\bar{v}} = (n-1)2^{t-v} + (n-2)t2^{-k} + t2^{-\bar{v}} + 2^{t+1-\bar{v}}.$$

In addition, DeRand outputs $\perp$ if $\mathsf{rank}_2(Z) < t$. By the ideal cipher model, we can assume that Z is distributed uniformly at random, and thus, we need to find the probability that a random $\bar{v} \times t$ matrix is not full-rank. By Lemma 6, this can be bounded by $2^{-\bar{v}+t}$.

Thus, the total probability of outputting $\perp$ can be upper-bounded by

$$(n-1)2^{t-v} + (n-2)t2^{-k} + t2^{-\bar{v}} + 3 \cdot 2^{t-\bar{v}}.$$

We note that, by our assumptions about our parameters, namely that $v > t + \kappa$, and $\bar{v} > k + t + \kappa + 2$, this is a negligible probability. Thus, optionally, one can omit the checks where $\perp$ is output, and with an overwhelming probability, this will not affect the result. □

Remark 1. We can omit the Ideal Cipher Model and add $t \cdot n \cdot \varepsilon_{\mathsf{PRG}} + 2^{-\kappa-k}$ to the probabilities displayed in Propositions 1 and 2. Indeed, if we follow the game jumps in the proof of Theorem 4, then we get a key generation algorithm whose output is computationally $t \cdot n \cdot \varepsilon_{\mathsf{PRG}} + 2^{-\kappa-k}$-close to the real key generation algorithm Gen. This new key generation algorithm behaves exactly as we assumed Gen would work under the Ideal Cipher Model.

Next, we show that if we have successfully deactivated an inactive node, then its children will also be inactive. Recall that for inactive nodes, the parties always share a zero value between them when evaluating. Recall that for $i < n$, we denote by $M_i(r)$ the value $(X_{r^1 r^2 \ldots r^{i-1}}, \tau_{r^1 r^2 \ldots r^{i-1}})$ and that if $i = n$, we denote by $M_i(r)$ the value $z_{r^1 r^2 \ldots r^n}$.

Proposition 3. *If, for a node* r, $M_i(r) = 0$, *then for both it's children* $r\|0$ *and* $r\|1$, $M_{i+1}(r\|0) = M_{i+1}(r\|1) = 0$.

Proof. If $X_r = [X_r]_1 + [X_r]_2 = 0$, then, as we are operating in a field with characteristic 2, $[X_r]_1 = [X_r]_2$. Likewise, if $\tau_r = [\tau_r]_1 + [\tau_r]_2 = 0$, then $[\tau_r]_1 = [\tau_r]_2$. Thus, since both parties apply the same deterministic functions to their values to compute the next shares of $M_{i+1}(r\|0)$ and $M_{i+1}(r\|1)$, all their future output shares are also equal. Hence, the secret shared values are also zero. (For $i = n$, recall that for $i = n$ we denote $M_i(r\|b) = z_{r\|b}$ where $z_{r\|b}$ is the respective output of GenRand.) □

Using the previous result, we show that, if $M_i(r) \neq 0$, then the node r is on a path to some active leaf α_j.

Proposition 4. *For all $i \in \{0, \ldots, n\}$, and for all $r \in \{0,1\}^i$, if $M_i(r) \neq 0$, then there exists $j \in \{1, \ldots, t\}$ such that $r^1 r^2 \ldots r^i = \alpha_j^1 \alpha_j^2 \ldots \alpha_j^i$ i.e. the node r is active.*

Proof. Assume, by contradiction, that there exists an inactive node r such that $M_i(r) \neq 0$. If there exists an inactive node r such that $M_i(r) \neq 0$, then there must exist one such node with the smallest i, w.l.o.g, let the value r be such a node with the smallest i. Note that $i > 1$, because the root is always active. Let the parent of r be p. If p is inactive and $M_{i-1}(p) \neq 0$, then this is a contradiction against the minimality of i. If p is inactive and $M_{i-1}(p) = 0$, then from Proposition 3 we get that $M_i(r) = 0$, which contradicts $M_i(r) \neq 0$.

Therefore, p must be active. Hence, in the i-th call to ProcessLayer, there must be a row in the matrix $A\vec{d}_{i-1} = \vec{b}$ corresponding to the parent. This row is, however, chosen exactly the way to make $M_i(r) = 0$. Hence, if $M_i(r) \neq 0$, then r is active (i.e. on a path to some active leaf).

□

Next, we argue that state contains the results of correct evaluation, the evaluation of an active path produces correct output, and the evaluation of an inactive path produces a shared secret of 0. This gives us that SLAMPR-FSS is correct.

Proposition 5. *For a honest evaluation of the* GenRand *algorithm that outputs* $\mathsf{state} = (R, \{[z_r]_1\}_{r\in R}, \{[z_r]_2\}_{r\in R}, \{z_r\}_{r\in R})$ *and* $\mathsf{sk}_1, \mathsf{sk}_2$*; and for honest evaluation of any $x \in R$, we have that for $\mathsf{p} \in \{1,2\}$,* $\mathsf{EvalRand}^{\mathsf{p}}(\mathsf{sk}_{\mathsf{p}}, x) = [z_x]_{\mathsf{p}}$ *and that $[z_x]_1 + [z_x]_2 = z_x$. By definition, $R = (\alpha_1, \ldots, \alpha_t)$.*

Proof. Recall that $M_n(r) = z_{r^1 r^2 \ldots r^n}$ where z_r is the respective output of GenRand. Note that, by definition, R contains all active nodes and for the final layer, this is exactly the active leaves in $\vec{\alpha}$. Hence, $x \in R$ is equivalent to, there exists j such that $x = \alpha_j$. This follows directly from how state is computed and how the function EvalRand is defined. □

Next, we consider SLAMP-FSS evaluation for one of the inputs that should give a nonzero output.

Proposition 6. *For all $x \in \{0,1\}^n$, if there exists $j \in \{1, \ldots, t\}$ such that $x = \alpha_j$, then $[\beta_j]_1 + [\beta_j]_2 = \beta_j$, where $[\beta_j]_1$ is the output of* Eval^1 *and $[\beta_j]_2$ is the output of* Eval^2*, provided that* Gen *succeeds.*

Proof. Let us fix $x \in \{0,1\}^n$ and assume there exists $j \in \{1, \ldots, t\}$ such that $x = \alpha_j$ and that Gen, Eval^1 and Eval^2 succeed. Let $[\beta_j]_1$ be the output of Eval^1 and $[\beta_j]_2$ be the output of Eval^2. By definition in DeRand and DeRandEval,

$$[\beta_j]_1 = \langle [X_x]_1, \vec{g} \rangle + [\tau_x]_1,$$
$$[\beta_j]_2 = \langle [X_x]_2, \vec{g} \rangle + [\tau_x]_2.$$

Since the scalar product is linear,

$$\beta_j = \langle X_x, \vec{g} \rangle + \tau_x = \langle [X_x]_1 + [X_x]_2, \vec{g} \rangle + [\tau_x]_1 + [\tau_x]_2 =$$
$$= \langle [X_x]_1, \vec{g} \rangle + [\tau_x]_1 + \langle [X_x]_2, \vec{g} \rangle + [\tau_x]_2 = [\beta_j]_1 + [\beta_j]_2.$$

Therefore, if the protocol succeeds, then $\mathcal{P}_1$ and $\mathcal{P}_2$ have a shared secret of β_j. □

As a final piece, we need to ensure that the evaluation on inputs other than $\vec{\alpha}$ gives zero as output. We show this for SLAMPR-FSS, from which we can derive this for SLAMP-FSS as well.

Proposition 7. *For all $x \in \{0,1\}^n$, if there does not exists $j \in \{1,\dots,t\}$ such that $x = \alpha_j$, then $[z]_1 + [z]_2 = 0$, where $[z]_1$ is the output of* $\mathsf{EvalRand}^1$ *and $[z]_2$ is the output of* $\mathsf{EvalRand}^2$*, provided that* GenRand *succeeds.*

Proof. Let us fix $x \in \{0,1\}^n$ and assume there does not exists $j \in \{1,\dots,t\}$ such that $x = \alpha_j$ and that GenRand, $\mathsf{EvalRand}^1$ and $\mathsf{EvalRand}^2$ succeed. Let $[z_j]_1$ be the output of $\mathsf{EvalRand}^1$ and $[z_j]_2$ be the output of $\mathsf{EvalRand}^2$. From the contrapositive of Proposition 4, we get that $M_n(x) = 0$. Recall that $[z_{r\|b}]_\mathsf{p} \leftarrow \left\langle [X_r]_\mathsf{p}, \vec{d}_{i-1} \right\rangle + [\tau_r]_\mathsf{p} \cdot w_{i,b}$. This means that $[X_x]_1 = [X_x]_2$ and $[\tau_x]_1 = [\tau_x]_2$, and thus

$$\begin{aligned}[z_j]_1 + [z_j]_2 &= \langle [X_x]_1, \vec{g}\rangle + [\tau_x]_1 + \langle [X_x]_2, \vec{g}\rangle + [\tau_x]_2 = \\ &= \langle [X_x]_1, \vec{g}\rangle + [\tau_x]_1 + \langle [X_x]_1, \vec{g}\rangle + [\tau_x]_1 = 0,\end{aligned}$$

as required. □

We have from Proposition 5 that for any $x \in R$, $\mathsf{EvalRand}(\mathsf{sk}_\mathsf{p}, x) = [z_x]_\mathsf{p}$. Thus, by construction of Algorithm 2 DeRand, and Algorithm 7 DeRandEval, we get that if the generation algorithm does not output $\perp$, the algorithm is correct. Hence, it remains to conclude that SLAMP-FSS is correct.

Theorem 3. *SLAMP-FSS defined by* Gen *and* Eval *is correct. That is, for all $\vec{\alpha} \in (\{0,1\}^n)^t$, $\vec{\beta} \in \mathbb{F}_{2^k}^t$, any input $x \in \{0,1\}^n$, if φ is t-multi-point function defined by $\vec{\alpha}$ and $\vec{\beta}$, then*

$$\Pr\left[\mathsf{Eval}^1(\mathsf{sk}_1, x) + \mathsf{Eval}^2(\mathsf{sk}_2, x) = \varphi(x) \mid \mathsf{sk}_1, \mathsf{sk}_2 \leftarrow \mathsf{Gen}(\vec{\alpha}, \vec{\beta})\right] = 1,$$

assuming Gen *succeeds.*

Proof. By Proposition 7, we get that if $x \notin \{\alpha_1,\dots,\alpha_t\}$, then evaluating EvalRand at x gives a sharing of 0 as an output. Thus, in DeRandEval, calling Bits on $[z]$ gives a shared vector of $\{z_i\}_{i=1}^t$ where all the z_i are zeroes. Then $y \leftarrow \sum_{i=1}^{\bar{v}} z_i g_i = \sum_{i=1}^{\bar{v}} 0 g_i = 0$. Thus, the function evaluates to zero for $x \notin \{\alpha_1,\dots,\alpha_t\}$. If $x \in \{\alpha_1,\dots,\alpha_t\}$, then the correctness comes directly from Proposition 6. □

6.2 Security

This section proves the security of SLAMPR-FSS and SLAMP-FSS. We show security in three steps. First, Theorem 4 shows that SLAMPR-FSS is a random DMPF (Definition 14). Secondly, Theorem 5 proves that if there is a random DMPF and we use the DeRand and DeRandEval algorithms, then it does not leak information about the multi-point function parameters, making it a secure distributed multi-point function. Finally, Theorem 6 proves that SLAMP-FSS is a secure distributed multi-point function due to Theorems 4 and 5.

First, we show that SLAMPR-FSS, as it is given in Algorithms 1 and 6, gives a random-DMPF scheme that is close to random. The proof is provided by a sequence of games that begins with the games defined in Definition 14. We then apply the security of the PRG-s and argue that the inputs to the PRG-s are distributed uniformly randomly (Game 3, Game 3.1, …, Game 3.t', Game 7). We then further simplify the algorithm by replacing various subsequent parts of the algorithm with uniformly random variables. We solve a random system of linear equations (Game 8, Game 9, Game 10), replace this system of linear equations with sampling a random vector (Game 13 and Game 14 using subgames Game 11 and Game 12), consider the last calls to ProcessLayer separately (Game 15), and simplify until we reach the desired output of random variables (Game 16, Game 4 and Game 4). The relationship between these games can be seen in Figure 3.

$$\mathcal{G}_3 \overset{\varepsilon_{\mathsf{PRG}}}{\approx} \mathcal{G}_{3.1} \overset{\varepsilon_{\mathsf{PRG}}}{\approx} \ldots \overset{\varepsilon_{\mathsf{PRG}}}{\approx} \mathcal{G}_{3.t'} \equiv \mathcal{G}_7 \equiv \mathcal{G}_8 \equiv \mathcal{G}_9 \equiv$$
$$\equiv \mathcal{G}_{10} \equiv \mathcal{G}_{13} \equiv \mathcal{G}_{14} \overset{2^{-k-\kappa}}{\approx} \mathcal{G}_{15} \equiv \mathcal{G}_{16} \equiv \mathcal{G}_{17} \equiv \mathcal{G}_4$$

Figure 3. Relations between the games of Theorem 4. $\mathcal{G}_i$ corresponds to Game i, $\mathcal{G}_i \equiv \mathcal{G}_j$ means that games i and j are equivalent, and $\mathcal{G}_i \overset{\varepsilon}{\approx} \mathcal{G}_j$ means that they are ε-close.

Game 3:
Input : Adversary $\mathcal{A}$
1 $(\alpha_1, \ldots, \alpha_t), \mu, \sigma \leftarrow \mathcal{A}()$
2 $\bar{\mu} \leftarrow 3 - \mu$
3 $\mathsf{state} = (\{[z_i]_1\}_{i=1}^t, \{[z_i]_2\}_{i=1}^t, \{z_i\}_{i=1}^t), \mathsf{sk}_1, \mathsf{sk}_2 \leftarrow \mathsf{GenRand}(\alpha_1, \ldots, \alpha_t)$
4 **return** $\mathcal{A}(\mathsf{sk}_\mu, \{[z_i]_{\bar{\mu}}\}_{i=1}^t, \sigma)$

Theorem 4. *Let $v > t + \kappa$, $\bar{v} > k + t + \kappa + 2$. Let $f\colon \mathbb{F}_{2^k} \to \{0,1\}^v \times \mathbb{F}_{2^k}$, $f'\colon \mathbb{F}_{2^k} \to \{0,1\}^{\bar{v}} \times \mathbb{F}_{2^{\bar{v}}}$ be $\varepsilon_{\mathsf{PRG}}$-PRGs.*
SLAMPR-FSS is $t \cdot n \cdot \varepsilon_{\mathsf{PRG}} + 2^{-\kappa-k}$-close to random.

Proof. In this proof, we follow a sequence of games. For each consecutive game, we use the magenta colour to distinguish where the game differs from the prior game. We note that, as the original algorithm was split into Algorithms 1 and 4, we needed to store the computed variables in the container state. In the proof, we are working with the whole algorithm written out, therefore, we do not use state and rather use all the variables therein directly. By Definition 14, we need to show that

$$|\Pr[\text{Game } 3(\mathcal{A}) = 1] - \Pr[\text{Game } 4(\mathcal{A}) = 1]| \leq t \cdot n \cdot \varepsilon_{\mathsf{PRG}} + 2^{-\kappa-k}.$$

This is sufficient, as Game 3 is just Game 1 from Definition 14 restated in the context of SLAMP-FSS, and Game 4 is Game 2 restated in the context of SLAMP-FSS, where GenPrim consists of the rows 2-8 and 14 of Game 4. Without loss of generality, we assume that $\mathcal{A}$ is corrupting $\mathcal{P}_\mu$, the other party is denoted as $\mathcal{P}_{\bar{\mu}}$.

By assumption, we have $\varepsilon_{\mathsf{PRG}}$-PRG $f\colon \mathbb{F}_{2^k} \to \{0,1\}^v \times \mathbb{F}_{2^k}$. By PRG security definition, for every PPT adversary $\mathcal{B}$:

$$|\Pr[\text{Game } 5(\mathcal{B}) = 1] - \Pr[\text{Game } 6(\mathcal{B}) = 1]| \leq \varepsilon_{\mathsf{PRG}}.$$

As shorthand, we use the following notation to denote the same as the line before

$$|\text{Game } 5 - \text{Game } 6| \leq \varepsilon_{\mathsf{PRG}}.$$

Now let us look at GenRand Algorithm 4 inside Game 3 and the first occurrence of the line (1) where $i = 1$

$$([X_{r\|b}]_{\bar{\mu}}, [\tau_{r\|b}]_{\bar{\mu}}) \leftarrow f\left(\left\langle [X_r]_{\bar{\mu}}, \vec{d}_{i-1} \right\rangle + [\tau_r]_{\bar{\mu}} \cdot w_{i,b}\right). \tag{1}$$

Since the root is always active, in other words, $\epsilon \in R_0$, then at least one of the corresponding node's children must be in R_1. Let us denote this child by b and its sibling by $\bar{b}$.

If $\bar{b} \notin R_1$, then $\vec{d}_0$ satisfies the condition that

$$\left\langle [X_\epsilon]_{\bar{\mu}}, \vec{d}_0 \right\rangle + [\tau_\epsilon]_{\bar{\mu}} \cdot w_{1,\bar{b}} = \left\langle [X_\epsilon]_\mu, \vec{d}_0 \right\rangle + [\tau_\epsilon]_\mu \cdot w_{1,\bar{b}}.$$

Game 4:
Input : Adversary $\mathcal{A}$

$(\alpha_1, \ldots, \alpha_t), \mu, \sigma \leftarrow \mathcal{A}()$
$\bar{\mu} \leftarrow 3 - \mu$
$X_\epsilon \xleftarrow{\$} \{0,1\}^v \setminus \{(0,0,\ldots,0)\}$
$\tau_\epsilon \xleftarrow{\$} \mathbb{F}_{2^k} \setminus \{0\}$
for $i = 1$ **to** n **do**
  $w_{i,0} \xleftarrow{\$} \mathbb{F}_{2^k} \setminus \{0\}$
  $w_{i,1} \xleftarrow{\$} \mathbb{F}_{2^k} \setminus \{0, w_{i,0}\}$
  With probability $1 - \prod_{i=0}^{t-1}(1 - 2^{i-v})$ **return** $\perp$
  $\vec{d}_{i-1} \xleftarrow{\$} \mathbb{F}_{2^k}^v$
$w_{n,0} \xleftarrow{\$} \mathbb{F}_{2^{\bar{v}}} \setminus \{0\}$
$w_{n,1} \xleftarrow{\$} \mathbb{F}_{2^{\bar{v}}} \setminus \{0, w_{i,0}\}$
With probability $1 - \prod_{i=0}^{t-1}(1 - 2^{i-v})$ **return** $\perp$
$\vec{d}_{n-1} \xleftarrow{\$} \mathbb{F}_{2^{\bar{v}}}^{\bar{v}}$
$\{[z_i]_{\bar{\mu}}\}_{i=1}^t \xleftarrow{\$} \mathbb{F}_{2^k}^t$
$\mathsf{sk} \leftarrow X_\epsilon, \tau_\epsilon, \{w_{i,0}\}_{i=1}^n, \{w_{i,1}\}_{i=1}^n, \{\vec{d}_i\}_{i=0}^{n-1}$
return $\mathcal{A}(\mathsf{sk}, \{[z_i]_{\bar{\mu}}\}_{i=1}^t, \sigma)$

Game 5:
Input : Adversary $\mathcal{B}$

$x \xleftarrow{\$} \mathbb{F}_{2^k}$
$y \leftarrow f(x)$
return $\mathcal{B}(y)$

Game 6:
Input : Adversary $\mathcal{B}$

$y \xleftarrow{\$} \{0,1\}^v \times \mathbb{F}_{2^k}$
return $\mathcal{B}(y)$

Figure 4. Games defining the PRG security.

This is equivalent to

$$[\tau_\epsilon]_{\bar{\mu}} = \left(\left\langle [X_\epsilon]_{\bar{\mu}} + [X_\epsilon]_\mu, \vec{d}_0 \right\rangle + [\tau_\epsilon]_\mu \cdot w_{1,\bar{b}}\right) w_{1,\bar{b}}^{-1} = \left\langle [X_\epsilon]_{\bar{\mu}} + [X_\epsilon]_\mu, \vec{d}_0 \right\rangle w_{1,\bar{b}}^{-1} + [\tau_\epsilon]_\mu.$$

Now

$$\left\langle [X_\epsilon]_{\bar{\mu}}, \vec{d}_0 \right\rangle + [\tau_\epsilon]_{\bar{\mu}} \cdot w_{1,b} = \left\langle [X_\epsilon]_{\bar{\mu}}, \vec{d}_0 \right\rangle + \left(\left\langle [X_\epsilon]_{\bar{\mu}} + [X_\epsilon]_\mu, \vec{d}_0 \right\rangle w_{1,\bar{b}}^{-1} + [\tau_\epsilon]_\mu\right) \cdot w_{1,b} =$$

$$= \left\langle [X_\epsilon]_{\bar{\mu}}, \vec{d}_0 \right\rangle \left(1 + w_{1,b} w_{1,\bar{b}}^{-1}\right) + \left\langle [X_\epsilon]_\mu, \vec{d}_0 \right\rangle w_{1,b} w_{1,\bar{b}}^{-1} + [\tau_\epsilon]_\mu \cdot w_{1,b}.$$

Since $\left(1 + w_{1,b} w_{1,\bar{b}}^{-1}\right) \neq 0$, $\vec{d}_0$ is full-rank, and $[X_\epsilon]_{\bar{\mu}}$ is a random binary vector, then by Lemma 1 the value of $\left\langle [X_\epsilon]_{\bar{\mu}}, \vec{d}_0 \right\rangle \left(1 + w_{1,b} w_{1,\bar{b}}^{-1}\right)$ is uniformly random over $\mathbb{F}_{2^k}$. Since $\left\langle [X_\epsilon]_\mu, \vec{d}_0 \right\rangle w_{1,b} w_{1,\bar{b}}^{-1} + [\tau_\epsilon]_\mu \cdot w_{1,b}$ are known to the adversary, we have that $\left\langle [X_\epsilon]_{\bar{\mu}}, \vec{d}_0 \right\rangle + [\tau_\epsilon]_{\bar{\mu}} \cdot w_{1,b}$ is also uniformly random.

On the other hand, if $\bar{b}$ is in R_0, then we have that

$$\left\langle [X_\epsilon]_{\bar{\mu}}, \vec{d}_0 \right\rangle + [\tau_\epsilon]_{\bar{\mu}} \cdot w_{\epsilon,2} = \left\langle [X_\epsilon]_\mu, \vec{d}_0 \right\rangle + [\tau_\epsilon]_\mu \cdot w_{\epsilon,2}.$$

Hence, $[\tau_\epsilon]_{\bar{\mu}} = \left\langle [X_\epsilon]_{\bar{\mu}} + [X_\epsilon]_\mu, \vec{d}_0 \right\rangle w_{\epsilon,2}^{-1} + [\tau_\epsilon]_\mu$ and

$$\left\langle [X_\epsilon]_{\bar{\mu}}, \vec{d}_0 \right\rangle + [\tau_\epsilon]_{\bar{\mu}} \cdot w_{1,b} = \left\langle [X_\epsilon]_{\bar{\mu}}, \vec{d}_0 \right\rangle \left(1 + w_{1,b} w_{\epsilon,2}^{-1}\right) + \left\langle [X_\epsilon]_\mu, \vec{d}_0 \right\rangle w_{1,b} w_{\epsilon,2}^{-1} + [\tau_\epsilon]_\mu \cdot w_{1,b}$$

We can proceed analogously to the case of $\bar{b} \notin R_0$, and obtain that, $\left\langle [X_\epsilon]_{\bar{\mu}}, \vec{d}_0 \right\rangle + [\tau_\epsilon]_{\bar{\mu}} \cdot w_{1,b}$ is distributed uniformly randomly.

Hence, in all cases, instead of line 1 computation, we can instead choose the input to f uniformly. Therefore, we can replace that line 1 by the same code as used in Game 5

$$x \xleftarrow{\$} \mathbb{F}_{2^k}$$
$$y \leftarrow f(x)$$

which, by PRG security definition, can be replaced by Game 6 oneliner $y \xleftarrow{\$} \{0,1\}^v \times \mathbb{F}_{2^k}$, because f is a PRG.

Analogously one can replace

$$x \xleftarrow{\$} \mathbb{F}_{2^k}$$
$$y \leftarrow f'(x)$$

by

$$y \xleftarrow{\$} \{0,1\}^{\bar{v}} \times \mathbb{F}_{2^{\bar{v}}}$$

for the cases using PRG f'.

Knowing this, we can replace the first occurrence of line (1) with

$$([X_{r\|b}]_{\bar{\mu}}, [\tau_{r\|b}]_{\bar{\mu}}) \xleftarrow{\$} \{0,1\}^v \times \mathbb{F}_{2^k},$$

and get a new game called Game 3.1, which differs from Game 3 by only this line and is therefore $\varepsilon_{\mathsf{PRG}}$-close to Game 3. (The first occurrence of line (1) happened in the first call to ProcessLayer in the main algorithm GenRand.)

Repeating this procedure at every occurrence of line (1) or, in other words, at every active node, we can replace all such lines and reach Game 3.t', which is $t \cdot n \cdot \varepsilon_{\mathsf{PRG}}$-close to Game 3. Here t' refers to the number of active nodes in the tree, and it is easy to see that $t' \leq tn$. Let us write out Game 3.t' and rename it to Game 7.

Game 7:
Input : Adversary $\mathcal{A}$
1 $(\alpha_1, \beta_1), \ldots, (\alpha_t, \beta_t) \leftarrow \mathcal{A}()$
2 $\cdots$ `// These lines are the same as in the previous game in the sequence`
3 **for** $r\|b \in R_i$ **do**
4 $\quad ([X_{r\|b}]_{\bar{\mu}}, [\tau_{r\|b}]_{\bar{\mu}}) \xleftarrow{\$} \{0,1\}^v \times \mathbb{F}_{2^k}$
5 $\quad ([X_{r\|b}]_{\mu}, [\tau_{r\|b}]_{\mu}) \leftarrow f\left(\left\langle [X_r]_\mu, \vec{d}_{i-1} \right\rangle + [\tau_r]_\mu \cdot w_{i,b}\right)$
6 $\quad X_{r\|b} \leftarrow [X_{r\|b}]_{\bar{\mu}} + [X_{r\|b}]_\mu$
7 $\quad \tau_{r\|b} \leftarrow [\tau_{r\|b}]_{\bar{\mu}} + [\tau_{r\|b}]_\mu$
8 $\cdots$
9 **return** $\mathcal{A}(\mathsf{sk}_\mu, \{[z_i]_{\bar{\mu}}\}_{i=1}^t)$

We have that

$$|\text{Game } 3 - \text{Game } 7| \leq t \cdot n \cdot \varepsilon_{\mathsf{PRG}}.$$

Note that the symbol "..." denotes the lines that have not been changed and are the same as in the previous game.

This can be further simplified, because $X_{r\|b}$ and $\tau_{r\|b}$ on lines 30 and 31 of ProcessLayer (that is, on lines 6 and 7 in Game 7) are now also uniformly random. We note that because for levels $i < n-1$, $[\tau_r]_{\bar{\mu}}$ and $[X_r]_{\bar{\mu}}$ are only used for computing the $X_{r\|b}$ and $\tau_{r\|b}$ which

Game 8:

$\cdots$
if $i < n-1$ **then**
  for $r\|b \in R_i$ **do**
    $([X_{r\|b}]_\mu, [\tau_{r\|b}]_\mu) \leftarrow f\left(\left\langle [X_r]_\mu, \vec{d}_{i-1}\right\rangle + [\tau_r]_\mu \cdot w_{i,b}\right)$
    $(X_{r\|b}, \tau_{r\|b}) \xleftarrow{\$} \{0,1\}^v \times \mathbb{F}_{2^k}$
else
  for $r\|b \in R_i$ **do**
    $([X_{r\|b}]_\mu, [\tau_{r\|b}]_\mu) \leftarrow f'\left(\left\langle [X_r]_\mu, \vec{d}_{i-1}\right\rangle + [\tau_r]_\mu \cdot w_{i,b}\right)$
    $(X_{r\|b}, \tau_{r\|b}) \xleftarrow{\$} \{0,1\}^{\bar{v}} \times \mathbb{F}_{2^{\bar{v}}}$
    $[X_{r\|b}]_{\bar{\mu}} \leftarrow X_{r\|b} - [X_{r\|b}]_\mu$
    $[\tau_{r\|b}]_{\bar{\mu}} \leftarrow \tau_{r\|b} - [\tau_{r\|b}]_\mu$
  $\cdots$
$\cdots$
return $\mathcal{A}(\mathsf{sk}_\mu, \{[z_i]_{\bar{\mu}}\}_{i=1}^t)$

are uniformly at randomly distributed, on those levels, we do not have to sample $[\tau_r]_{\bar{\mu}}$ and $[X_r]_{\bar{\mu}}$ and can instead sample $X_{r\|b}$ and $\tau_{r\|b}$ independently from $[X_{r\|b}]_\mu, [\tau_{r\|b}]_\mu$.

For $i = n-1$, we note that $[X_{r\|b}]_{\bar{\mu}}, [\tau_{r\|b}]_{\bar{\mu}}$ are also used for computing the $[z_{r\|bb'}]_{\bar{\mu}}$ and hence, $z_{r\|bb'}$ where $r\|bb' \in R_n$. We thus want to still have a $[X_{r\|b}]_{\bar{\mu}}, [\tau_{r\|b}]_{\bar{\mu}}$ that satisfies $[X_{r\|b}]_{\bar{\mu}} + [X_{r\|b}]_\mu = X_{r\|b}$ and $[\tau_{r\|b}]_{\bar{\mu}} + [\tau_{r\|b}]_\mu = \tau_{r\|b}$. We thus first sample X_r and τ_r independently and randomly in step $i = n-1$, and then compute $[X_{r\|b}]_{\bar{\mu}} \leftarrow X_{r\|b} - [X_{r\|b}]_\mu$ and $[\tau_{r\|b}]_{\bar{\mu}} \leftarrow \tau_{r\|b} - [\tau_{r\|b}]_\mu$.

For $i = n$, we note that for any $r\|b \in R_n$, $[z_{r\|b}]_{\bar{\mu}}$ is computed as $\left\langle [X_r]_{\bar{\mu}}, \vec{d}_{i-1}\right\rangle + [\tau_r]_{\bar{\mu}} \cdot w_{i,b}$ and $z_{r\|b} \leftarrow [z_{r\|b}]_{\bar{\mu}} + [z_{r\|b}]_\mu$. We note that there are no PRG calls here and that here the $[X_r]_{\bar{\mu}}$ and $[\tau_r]_{\bar{\mu}}$ have been computed already in the previous step as described above, thus, no new changes are needed here.

Clearly, this new Game 8 is equivalent to the previous one. Again,

$$|\text{Game } 7 - \text{Game } 8| = 0.$$

We can do the sampling of $X_{r\|b}$ and $\tau_{r\|b}$ right before we need the sampled values and shift the lines to get Game 9. Naturally,

$$|\text{Game } 8 - \text{Game } 9| = 0.$$

At every step, we sample at random the values X_r and τ_r and place them in matrices and vectors. Now, with syntactic sugar, we simply sample the matrices and vectors directly. For $i < n-1$, we do not sample the X_r and τ_r anymore for $r \in R_i$, although the $[X_r]_\mu$ and $[\tau_r]_\mu$ are still normally computed. For $i = n-1$, it is almost the same case, except that we still randomly sample $(X_{r\|b}, \tau_{r\|b})$ and use them to compute $[X_{r\|b}]_{\bar{\mu}}$ and $[\tau_{r\|b}]_{\bar{\mu}}$

We thus get Game 10, with

$$|\text{Game } 9 - \text{Game } 10| = 0.$$

Since for $i < n$, $\vec{d}_{i-1}$ is not used for calculating $X_{r\|b}, \tau_{r\|b}$, then we can try to replace it with a random vector of length v. Let us fix $\vec{d} \in \mathbb{F}_{2^k}^v$. We shall do this by showing that the statistical distance between Game 11 and Game 12 is small. However, let us first calculate the statistical distance $SD(\text{Game } 11, \text{Game } 12)$.

Game 9:
Input : Adversary $\mathcal{A}$

1 $(\alpha_1, \beta_1), \ldots, (\alpha_t, \beta_t) \leftarrow \mathcal{A}()$
2 $[X_\epsilon]_\mu \xleftarrow{\$} \{0,1\}^v$
3 $[\tau_\epsilon]_\mu \xleftarrow{\$} \mathbb{F}_{2^k}$
4 $R_0 \leftarrow \{\epsilon\}$, where ϵ is an empty string
5 **for** $i = 1$ **to** n **do**
6 $\quad \cdots$
7 $\quad$ **if** $i < n$ **then**
8 $\quad\quad A \leftarrow \mathsf{Mat}_{R_{i-1},v}(0)(0)$
9 $\quad\quad \vec{b} \leftarrow \mathsf{Vec}_{|R_{i-1}|}(0)$
10 $\quad\quad j_i \leftarrow 1$
11 $\quad\quad$ **for** $r \| b \in R'_i$ **do**
12 $\quad\quad\quad (X_r, \tau_r) \xleftarrow{\$} \{0,1\}^v \times \mathbb{F}_{2^k}$
13 $\quad\quad\quad A^{(j_i)} \leftarrow X_r$
14 $\quad\quad\quad \vec{b}_j \leftarrow \tau_r \cdot w_{i,b}$
15 $\quad\quad\quad j_i \leftarrow j_i + 1$
16 $\quad\quad$ **for** $r \in \widehat{R}_{i-1}$ **do**
17 $\quad\quad\quad w_{r,2} \xleftarrow{\$} \mathbb{F}_{2^k} \setminus \{w_{i,0}, w_{i,1}\}$
18 $\quad\quad\quad (X_r, \tau_r) \xleftarrow{\$} \{0,1\}^v \times \mathbb{F}_{2^k}$
19 $\quad\quad\quad A^{(j_i)} \leftarrow X_r$
20 $\quad\quad\quad \vec{b}_j \leftarrow \tau_r \cdot w_{r,2}$
21 $\quad\quad\quad j_i \leftarrow j_i + 1$
22 $\quad\quad$ **if** $\mathsf{rank}(A) < |R_{i-1}|$ **then**
23 $\quad\quad\quad$ **return** $\perp$
24 $\quad\quad$ Solve $A\vec{d}_{i-1} = \vec{b}$ and sample $\vec{d}_{i-1}$ from the solution space
25 $\quad\quad \cdots$
26 $\quad$ **else**
27 $\quad\quad \cdots$
28 $\quad \cdots$
29 $\quad$ **return** $\mathcal{A}(\mathsf{sk}_\mu, \{[z_i]_{\bar{\mu}}\}_{i=1}^t)$

By Lemma 10, we know that any full-rank matrix is equally likely in Game 11. In addition, the probability of the matrix not being full rank is $1 - \prod_{i=0}^{|R_{i-1}|-1}(1 - 2^{i-v})$. Let us now consider the probability of abort on line 5. We reach line 5 in Game 11 with probability $\prod_{i=0}^{|R_{i-1}|-1}(1 - 2^{i-v})$ as this is the probability that the matrix obtained is full-rank and thus, we didn't abort on the previous line.

Provided that we have reached that line, the probability of abort is $\xi(|R_{i-1}|, t, v) = 1 - \frac{\prod_{i=0}^{t-1}(1-2^{i-v})}{\prod_{i=0}^{|R_{i-1}|-1}(1-2^{i-v})}$.

Thus, the total abort probability is

$$1 - \prod_{i=0}^{|R_{i-1}|-1}(1 - 2^{i-v}) + \left(\prod_{i=0}^{|R_{i-1}|-1}(1 - 2^{i-v})\right)\left(1 - \frac{\prod_{i=0}^{t-1}(1 - 2^{i-v})}{\prod_{i=0}^{|R_{i-1}|-1}(1 - 2^{i-v})}\right) =$$

$$= 1 - \prod_{i=0}^{|R_{i-1}|-1}(1 - 2^{i-v}) + \prod_{i=0}^{|R_{i-1}|-1}(1 - 2^{i-v}) - \prod_{i=0}^{t-1}(1 - 2^{i-v}) = 1 - \prod_{i=0}^{t-1}(1 - 2^{i-v}).$$

Game 10:
Input : Adversary $\mathcal{A}$

1 $(\alpha_1, \beta_1), \ldots, (\alpha_t, \beta_t) \leftarrow \mathcal{A}()$
2 $\cdots$
3 **for** $i = 1$ **to** n **do**
4 $\quad \cdots$
5 $\quad$ **if** $i < n - 1$ **then**
6 $\quad\quad A \xleftarrow{\$} \{0,1\}^{|R_{i-1}| \times v}$
7 $\quad\quad \vec{b} \xleftarrow{\$} \mathbb{F}_{2^k}^{|R_{i-1}|}$
8 $\quad\quad$ **if** $\mathsf{rank}(A) < |R_{i-1}|$ **then**
9 $\quad\quad\quad$ **return** $\perp$
10 $\quad\quad$ With probability $\xi(|R_{i-1}|, t, v)$, **return** $\perp$
11 $\quad\quad$ Solve $A\vec{d}_{i-1} = \vec{b}$ and sample $\vec{d}_{i-1}$ from the solution space
12 $\quad\quad$ **for** $r \| b \in R_i$ **do**
13 $\quad\quad\quad ([X_{r\|b}]_\mu, [\tau_{r\|b}]_\mu) \leftarrow f\left(\left\langle [X_r]_\mu, \vec{d}_{i-1} \right\rangle + [\tau_r]_\mu \cdot w_{i,b}\right)$
14 $\quad\quad \cdots$
15 $\quad$ **else**
16 $\quad\quad$ **if** $i = n - 1$ **then**
17 $\quad\quad\quad A \xleftarrow{\$} \{0,1\}^{|R_{i-1}| \times v}$
18 $\quad\quad\quad \vec{b} \xleftarrow{\$} \mathbb{F}_{2^k}^{|R_{i-1}|}$
19 $\quad\quad\quad$ **if** $\mathsf{rank}(A) < |R_{i-1}|$ **then**
20 $\quad\quad\quad\quad$ **return** $\perp$
21 $\quad\quad\quad$ Solve $A\vec{d}_{i-1} = \vec{b}$ and sample $\vec{d}_{i-1}$ from the solution space
22 $\quad\quad\quad$ **for** $r \| b \in R_i$ **do**
23 $\quad\quad\quad\quad ([X_{r\|b}]_\mu, [\tau_{r\|b}]_\mu) \leftarrow f'\left(\left\langle [X_r]_\mu, \vec{d}_{i-1} \right\rangle + [\tau_r]_\mu \cdot w_{i,b}\right)$
24 $\quad\quad\quad\quad (X_{r\|b}, \tau_{r\|b}) \xleftarrow{\$} \{0,1\}^{\bar{v}} \times \mathbb{F}_{2^{\bar{v}}}$
25 $\quad\quad\quad\quad [X_{r\|b}]_{\bar{\mu}} \leftarrow X_{r\|b} - [X_{r\|b}]_\mu$
26 $\quad\quad\quad\quad [\tau_{r\|b}]_{\bar{\mu}} \leftarrow \tau_{r\|b} - [\tau_{r\|b}]_\mu$
27 $\quad\quad\quad\quad \cdots$
28 $\quad\quad$ **else**
29 $\quad\quad\quad \cdots$
30 $\quad \cdots$
31 $\quad$ **return** $\mathcal{A}(\mathsf{sk}_\mu, \{[z_i]_{\bar{\mu}}\}_{i=1}^t)$

Thus, every output $\vec{d}$ has the same probability in Game 11 and Game 12 and the probability of $\perp$ is the same in those games, thus $SD(\text{Game } 11, \text{Game } 12) = 0$.

Now we can use these subgames and replace the linear equations in Game 10 with random $\vec{d}$ and get Game 13. We do this with the first $n - 1$ levels. We move the random sampling of $(X_{r\|b}, \tau_{r\|b})$ in Game 10 line 24 in the step where $i = n - 1$ to $i = n$ in Game 13, that is, to lines 24 to 26 in Game 13. This requires some syntactic sugar for formal correctness, which is also highlighted. In total,

$$|\text{Game } 10 - \text{Game } 13| = 0.$$

We do an analogous change in the case of $i = n$, however, it is slightly different, as there we still use the $A, \vec{b}$. Thus, in Game 14, we first sample $\vec{d}_{n-1}$ at random from full-rank $\bar{v} \times \bar{v}$ binary matrices. We then sample $A, \vec{b}$ on the condition that $A\vec{d}_{n-1} = \vec{b}$.

Game 13:
Input : Adversary $\mathcal{A}$

$(\alpha_1, \beta_1), \ldots, (\alpha_t, \beta_t) \leftarrow \mathcal{A}()$
$[X_\epsilon]_\mu \xleftarrow{\$} \{0,1\}^v$
$[\tau_\epsilon]_\mu \xleftarrow{\$} \mathbb{F}_{2^k}$
for $i = 1$ **to** n **do**
  $w_{i,0} \xleftarrow{\$} \mathbb{F}_{2^k} \setminus \{0\}$
  $w_{i,1} \xleftarrow{\$} \mathbb{F}_{2^k} \setminus \{0, w_{i,0}\}$
  $R_i \leftarrow \emptyset$
  $R'_i \leftarrow \emptyset$
  $\widehat{R}_{i-1} \leftarrow \emptyset$
  for $r \in R_{i-1}$ **do**
    for $b \in \{0,1\}$ **do**
      if $\exists \alpha_j$ *such that* α_j *starts with* $r\|b$ **then**
        $R_i \leftarrow R_i \cup \{r\|b\}$
      else
        $R'_i \leftarrow R'_i \cup \{r\|b\}$
    if $r\|0 \in R_i$ *and* $r\|1 \in R_i$ **then**
      $\widehat{R}_{i-1} \leftarrow \widehat{R}_{i-1} \cup \{r\}$
  if $i < n$ **then**
    With probability $1 - \prod_{i=0}^{t-1}(1 - 2^{i-v})$ **return** $\perp$
    $\vec{d}_{i-1} \xleftarrow{\$} \mathbb{F}_{2^k}^v$ such that $\mathsf{rank}_2(\vec{d}_{i-1}) = k$
  else
    $A \leftarrow \mathsf{Mat}_{|R_{i-1}|,k}(0)$
    $\vec{b} \leftarrow \mathsf{Vec}_{|R_{i-1}|}(0)$
    for $j = 1$ **to** $|R_{i-1}|$ **do**
      $X_j \xleftarrow{\$} \mathbb{F}_2^{\bar{v}}$
      $\tau_j \xleftarrow{\$} \mathbb{F}_{2^{\bar{v}}}$
      $A^{(j)} \leftarrow X_j$
      Parse j-th element of R_{i-1} as r
      if $r \in \widehat{R}_{i-1}$ **then**
        $w_{r,2} \xleftarrow{\$} \mathbb{F} \setminus \{0, w_{i,0}, w_{i,1}\}$
        $\vec{b}_j \leftarrow \tau_j \cdot w_{r,2}$
      else
        Let $\bar{b_j}$ be such that $r\|\bar{b_j} \in R'_i$
        $\vec{b}_j \leftarrow \tau_j \cdot w_{i,\bar{b_j}}$
    if $\mathsf{rank}(A) < |R_{i-1}|$ *or* $\mathsf{rank}(\vec{b}) < |R_{i-1}|$ **then**
      return $\perp$
    Solve $A\vec{d}_{i-1} = \vec{b}$ and sample $\vec{d}_{i-1}$ from the solution space such that $\mathsf{rank}_2(\vec{d}_{i-1}) = \bar{v}$
    for $r\|b \in R_i$ **do**
      $[X_r]_{\bar{\mu}} \leftarrow X_r - [X_r]_\mu$
      $[\tau_r]_{\bar{\mu}} \leftarrow \tau_r - [\tau_r]_\mu$
      $[z_{r\|b}]_\mu \leftarrow \left\langle [X_r]_\mu, \vec{d}_{i-1} \right\rangle + [\tau_r]_\mu \cdot w_{i,b}$
      $[z_{r\|b}]_{\bar{\mu}} \leftarrow \left\langle [X_r]_{\bar{\mu}}, \vec{d}_{i-1} \right\rangle + [\tau_r]_{\bar{\mu}} \cdot w_{i,b}$
      $z_{r\|b} \leftarrow [z_{r\|b}]_\mu + [z_{r\|b}]_{\bar{\mu}}$
return $\mathcal{A}(([X_\epsilon]_\mu, [\tau_\epsilon]_\mu, \{w_{i,0}\}_{i=1}^n, \{w_{i,1}\}_{i=1}^n, \{\vec{d}_i\}_{i=0}^{n-1}), \{[z_r]_{\bar{\mu}}\}_{r \in R_n})$

Game 11:

$A \xleftarrow{\$} \{0,1\}^{|R_{i-1}|\times v}$
$\vec{b} \xleftarrow{\$} \mathbb{F}_{2^k}^{|R_{i-1}|}$
if $\mathsf{rank}(A) < |R_{i-1}|$ **then**
  return $\perp$
With probability $\xi(|R_{i-1}|, t, v)$, **return** $\perp$
Solve $A\vec{d} = \vec{b}$ and choose a full-rank $\vec{d}$
return $\vec{d}$

Game 12:

$\vec{d} \xleftarrow{\$} \mathbb{F}_{2^k}^{v}$ such that $\mathsf{rank}_2(\vec{d}) = k$
With probability $1 - \prod_{i=0}^{t-1}(1-2^{i-v})$ **return** $\perp$
else **return** $\vec{d}$

Figure 5. Subgames substituting solving a linear equation with random sampling

Indeed, it is not hard to see that this is what happens in Game 14. Sampling randomly $[X_r]_{\bar{\mu}}, [\tau_r]_{\bar{\mu}}$ on the condition $\left\langle [X_r]_{\mu}, \vec{d}_{n-1} \right\rangle + [\tau_r]_{\mu} \cdot w_{n,0} = \left\langle [X_r]_{\bar{\mu}}, \vec{d}_{n-1} \right\rangle + [\tau_r]_{\bar{\mu}} \cdot w_{n,0}$ is equivalent to sampling them randomly under the constraint that $\left\langle [X_r]_{\mu} + [X_r]_{\bar{\mu}}, \vec{d}_{n-1} \right\rangle + ([\tau_r]_{\mu} + [\tau_r]_{\bar{\mu}}) \cdot w_{n,0} = 0$. Changing notation as it is done in the algorithms, this is equivalent to $\left\langle A^{(j)}, \vec{d}_{n-1} \right\rangle = w_{n,0} \cdot \vec{b}_j$ for some $j \in [1, |R_{n-1}|]$. This is the case for all of the elements of $r \in R_{n-1}$.

We show that for any pair $(\tilde{A}, \tilde{d})$ with $\tilde{A} \in \mathbb{F}_2^{t\times\bar{v}}$ and $\tilde{d} \in \mathbb{F}_2^{\bar{v}\times\bar{v}}$ with both $\tilde{A}$ and $\tilde{d}$ having full rank, we have the following probability

$$\Pr[A = \tilde{A} \wedge \vec{d}_{n-1} = \tilde{d}] = \frac{1}{2^{\bar{v}t}\prod_{i=0}^{t-1}(1-2^{i-\bar{v}}) \cdot 2^{\bar{v}^2}\prod_{i=0}^{\bar{v}-1}(1-2^{i-\bar{v}})}$$

holding in both games Game 13 and Game 14.

Note that if $\tilde{A}$ and $\tilde{d}$ are full-rank, then also $\tilde{A}\tilde{d}$ is full-rank. In Game 13, we first sample $A, \vec{b}$ and if they are full-rank, we sample $\vec{d}_{n-1}$ among the solutions to $A\vec{d}_{n-1} = \vec{b}$. Thus, for any full-rank $\tilde{A}, \tilde{d}$,

$$\Pr[A = \tilde{A} \wedge \vec{d}_{n-1} = \tilde{d}] = \Pr[A = \tilde{A}] \Pr[\vec{d}_{n-1} = \tilde{d} | A = \tilde{A}] =$$

$$= \sum_{\tilde{\vec{b}} \in \mathbb{F}_2^{t\times\bar{v}}} \Pr[A = \tilde{A}] \Pr[\vec{b} = \tilde{\vec{b}} | A = \tilde{A}] \Pr[\vec{d}_{n-1} = \tilde{d} | A = \tilde{A} \wedge \vec{b} = \tilde{\vec{b}}] =$$

$$= \sum_{\tilde{\vec{b}} \in \mathbb{F}_2^{t\times\bar{v}}} \Pr[A = \tilde{A}] \Pr[\vec{b} = \tilde{\vec{b}}] \Pr[\vec{d}_{n-1} = \tilde{d} | A = \tilde{A} \wedge \vec{b} = \tilde{\vec{b}}] =$$

$$= \Pr[A = \tilde{A}] \Pr[\vec{b} = \tilde{A}\tilde{d}] \Pr[\vec{d}_{n-1} = \tilde{d} | A = \tilde{A} \wedge \vec{b} = \tilde{A}\tilde{d}] =$$

$$= \frac{1}{|\{\tilde{A} \in \mathbb{F}_2^{t\times\bar{v}} | \mathsf{rank}_2(\tilde{A}) = t\}|} \frac{1}{|\{\tilde{\vec{b}} \in \mathbb{F}_2^{t\times\bar{v}} | \mathsf{rank}_2(\tilde{\vec{b}}) = t\}|} \frac{1}{|\{\tilde{d} \in \mathbb{F}_2^{\bar{v}\times\bar{v}} | \mathsf{rank}_2(\tilde{d}) = \bar{v} \wedge \tilde{A}\tilde{d} = \tilde{\vec{b}}\}|} =$$

$$= \frac{1}{2^{\bar{v}t}\prod_{i=0}^{t-1}(1-2^{i-\bar{v}})} \cdot \frac{1}{2^{\bar{v}t}\prod_{i=0}^{t-1}(1-2^{i-\bar{v}})} \cdot \frac{1}{2^{\bar{v}(\bar{v}-t)}\prod_{i=0}^{\bar{v}-t-1}(1-2^{i-(\bar{v}-t)})} =$$

$$= \frac{1}{2^{\bar{v}t}\prod_{i=0}^{t-1}(1-2^{i-\bar{v}}) \cdot 2^{\bar{v}^2}\prod_{i=0}^{\bar{v}-1}(1-2^{i-\bar{v}})}.$$

Here we evaluated $|\{\tilde{d} \in \mathbb{F}_2^{\bar{v}\times\bar{v}} | \mathsf{rank}_2(\tilde{d}) = \bar{v} \wedge \tilde{A}\tilde{d} = \tilde{\vec{b}}\}| = 2^{\bar{v}(\bar{v}-t)}\prod_{i=0}^{\bar{v}-t-1}(1-2^{i-(\bar{v}-t)})$. This comes from first counting all the solutions to $\tilde{A}\tilde{d} = \tilde{\vec{b}}$, which is $2^{\bar{v}(\bar{v}-t)}$ by Lemma 5,

and then multiplying with the probability that a solution is full-rank, which is $\prod_{i=0}^{\bar{v}-t-1}(1-2^{i-(\bar{v}-t)})$ by Lemma 4.

On the other hand, in Game 14, we have that

$$\begin{aligned}
&\Pr[A=\tilde{A}\wedge\vec{d}_{n-1}=\tilde{d}]=\Pr[\vec{d}_{n-1}=\tilde{d}]\Pr[A=\tilde{A}|\vec{d}_{n-1}=\tilde{d}]=\\
&=\sum_{\tilde{\vec{b}}\in\mathbb{F}_2^{t\times\bar{v}}}\Pr[\vec{d}_{n-1}=\tilde{d}]\Pr[A=\tilde{A}\wedge\vec{b}=\tilde{\vec{b}}|\vec{d}_{n-1}=\tilde{d}]=\\
&=\Pr[\vec{d}_{n-1}=\tilde{d}]\Pr[A=\tilde{A}\wedge\vec{b}=\tilde{A}\tilde{d}|\vec{d}_{n-1}=\tilde{d}]=\\
&=\Pr[\vec{d}_{n-1}=\tilde{d}]\frac{|\{(\tilde{A},\tilde{\vec{b}})\in\mathbb{F}_2^{t\times\bar{v}}\times\mathbb{F}_2^{t\times\bar{v}}|\tilde{A}\tilde{d}=\tilde{\vec{b}}\wedge\mathsf{rank}_2(\tilde{A})=t\wedge\mathsf{rank}_2(\tilde{\vec{b}})=t\}|}{|\{(\tilde{A},\tilde{\vec{b}})\in\mathbb{F}_2^{t\times\bar{v}}\times\mathbb{F}_2^{t\times\bar{v}}|\mathsf{rank}_2(\tilde{A})=t\wedge\mathsf{rank}_2(\tilde{\vec{b}})=t\}|}=\\
&=\Pr[\vec{d}_{n-1}=\tilde{d}]\frac{1}{|\{\tilde{\vec{b}}\in\mathbb{F}_2^{t\times\bar{v}}|\mathsf{rank}_2(\tilde{\vec{b}})=t\}|}=\frac{1}{2^{\bar{v}^2}\prod_{i=0}^{\bar{v}-1}(1-2^{i-\bar{v}})}\frac{1}{2^{\bar{v}t}\prod_{i=0}^{t-1}(1-2^{i-\bar{v}})}=\\
&=\frac{1}{2^{\bar{v}t}\prod_{i=0}^{t-1}(1-2^{i-\bar{v}})\cdot 2^{\bar{v}^2}\prod_{i=0}^{\bar{v}-1}(1-2^{i-\bar{v}})}.
\end{aligned}$$

The second-to-last-equality comes from first noting that $\vec{d}_{n-1}$ is chosen from the set of full-rank $\bar{v}\times\bar{v}$ binary matrices, of which there are $2^{\bar{v}^2}\prod_{i=0}^{\bar{v}-1}(1-2^{i-\bar{v}})$ by Lemma 2, and secondly from noting that the number of full-rank $t\times\bar{v}$ binary matrices is $2^{\bar{v}t}\prod_{i=0}^{t-1}(1-2^{i-\bar{v}})$ by the same lemma. Thus,

$$|\text{Game } 13-\text{Game } 14|=0.$$

Let us argue that $[z_{r\|b}]_{\bar{\mu}}$ are randomly distributed. If $r\in R_{n-1}$ and $r\in\widehat{R_{n-1}}$, then we have that $z_{r\|0}=\left\langle X_r,\vec{d}_{n-1}\right\rangle+\tau_r\cdot w_{n,0}$ and $z_{r\|1}=\left\langle X_r,\vec{d}_{n-1}\right\rangle+\tau_r\cdot w_{n,1}$, then deducing those equations from each other gives us

$$z_{r\|0}-z_{r\|1}=\tau_r\cdot w_{n,0}-\tau_r\cdot w_{n,1}$$

and hence, must first hold that $\tau_r=\frac{z_{r\|0}-z_{r\|1}}{w_{n,0}-w_{n,1}}$. Then the value of $\left\langle X_r,\vec{d}_{n-1}\right\rangle$ must be $z_{r\|0}-\tau_r\cdot w_{n,0}=z_{r\|0}-\frac{z_{r\|0}-z_{r\|1}}{w_{n,0}-w_{n,1}}\cdot w_{n,0}$. By Lemma 1, we know that for a fixed full-rank $\vec{d}_{n-1}$ every specific value $\tilde{a}\in\mathbb{F}_{2^k}$, there are exactly 2^{v-k} values for X_r when $\left\langle X_r,\vec{d}_{n-1}\right\rangle=\tilde{a}$.

Thus, for a particular pair of values for (z_0,z_1) to happen, it must happen that $\tau_r=\frac{z_{r\|0}-z_{r\|1}}{w_{n,0}-w_{n,1}}$ and $\left\langle X_r,\vec{d}_{n-1}\right\rangle=z_{r\|0}-\frac{z_{r\|0}-z_{r\|1}}{w_{n,0}-w_{n,1}}\cdot w_{n,0}$, the probability of which is $2^{-v}2^{v-k}2^{-k}=2^{-2k}$.

If $r\in R_{n-1}$ and $r\notin\widehat{R_{n-1}}$, then exactly one of $r\|0$ and $r\|1$ is active and one is inactive. Let us denote the inactive child by $r\|b$ and the active one by $r\|\bar{b}$. In that case,

$$z_i=[z_i]_{\bar{\mu}}+[z_i]_{\mu}=\left\langle[X_r]_{\bar{\mu}},\vec{d}_{n-1}\right\rangle+[\tau_r]_{\bar{\mu}}\cdot w_{n,\bar{b}}+\left\langle[X_r]_{\mu},\vec{d}_{n-1}\right\rangle+[\tau_r]_{\mu}\cdot w_{n,\bar{b}}=$$

$$=\left\langle[X_r]_{\bar{\mu}},\vec{d}_{n-1}\right\rangle+[\tau_r]_{\bar{\mu}}\cdot w_{n,\bar{b}}+\left\langle[X_r]_{\mu},\vec{d}_{n-1}\right\rangle+[\tau_r]_{\mu}\cdot w_{n,\bar{b}}+\left\langle[X_r]_{\bar{\mu}},\vec{d}_{n-1}\right\rangle+[\tau_r]_{\bar{\mu}}\cdot w_{n,b}+$$

$$+\left\langle[X_r]_{\mu},\vec{d}_{n-1}\right\rangle+[\tau_r]_{\mu}\cdot w_{n,b}=[\tau_r]_{\bar{\mu}}\cdot(w_{n,\bar{b}}+w_{n,b})+[\tau_r]_{\mu}\cdot(w_{n,\bar{b}}+w_{n,b}).$$

Note that from the viewpoint of an adversary who knows sk_{μ}, $[\tau_r]_{\mu}$ and $(w_{n,\bar{b}}+w_{n,b})$ are known and $[\tau_r]_{\bar{\mu}}$ is distributed uniformly at random. From this, it follows that from the viewpoint of the adversary, z_i is also uniformly randomly distributed. Because of this, we

Game 14:
Input : Adversary $\mathcal{A}$

$(\alpha_1, \beta_1), \ldots, (\alpha_t, \beta_t) \leftarrow \mathcal{A}()$
$[X_\epsilon]_\mu \xleftarrow{\$} \{0,1\}^v$
$[\tau_\epsilon]_\mu \xleftarrow{\$} \mathbb{F}_{2^k}$
for $i = 1$ **to** n **do**
  $\ldots$
  if $i < n$ **then**
    With probability $1 - \prod_{i=0}^{t-1}(1 - 2^{i-v})$ **return** $\perp$
    $\vec{d}_{i-1} \xleftarrow{\$} \mathbb{F}_{2^k}^v$
    for $r \| b \in R_i$ **do**
      $([X_{r\|b}]_\mu, [\tau_{r\|b}]_\mu) \leftarrow f\left(\left\langle [X_r]_\mu, \vec{d}_{i-1} \right\rangle + [\tau_r]_\mu \cdot w_{i,b}\right)$
  else
    $\vec{d}_{n-1} \xleftarrow{\$} \mathbb{F}_{2^{\bar{v}}}^{\bar{v}}$ on the condition that $\mathsf{rank}_2(\vec{d}_{i-1}) = \bar{v}$
    For all $r \in R_{n-1}$:
    if $r \notin \widehat{R_{n-1}}$ **then**
      if $r\|0 \in R'_n$ **then**
        Sample randomly $[X_r]_{\bar{\mu}}, [\tau_r]_{\bar{\mu}}$ on condition
        $\left\langle [X_r]_\mu, \vec{d}_{n-1} \right\rangle + [\tau_r]_\mu \cdot w_{n,0} = \left\langle [X_r]_{\bar{\mu}}, \vec{d}_{n-1} \right\rangle + [\tau_r]_{\bar{\mu}} \cdot w_{n,0}$
      else
        Sample randomly $[X_r]_{\bar{\mu}}, [\tau_r]_{\bar{\mu}}$ on condition
        $\left\langle [X_r]_\mu, \vec{d}_{n-1} \right\rangle + [\tau_r]_\mu \cdot w_{n,1} = \left\langle [X_r]_{\bar{\mu}}, \vec{d}_{n-1} \right\rangle + [\tau_r]_{\bar{\mu}} \cdot w_{n,1}$
    else
      Sample randomly $[X_r]_{\bar{\mu}}, [\tau_r]_{\bar{\mu}}$ on condition
      $\left\langle [X_r]_\mu, \vec{d}_{n-1} \right\rangle + [\tau_r]_\mu \cdot w_{r,2} = \left\langle [X_r]_{\bar{\mu}}, \vec{d}_{n-1} \right\rangle + [\tau_r]_{\bar{\mu}} \cdot w_{r,2}$
    $A \leftarrow \begin{pmatrix} [X_{r_1}]_{\bar{\mu}} + [X_{r_1}]_\mu \\ \vdots \\ [X_{r_1}]_{\bar{\mu}} + [X_{r_1}]_\mu \end{pmatrix}, \vec{b} \leftarrow \begin{pmatrix} [\tau_{r_1}]_{\bar{\mu}} + [\tau_{r_1}]_\mu \\ \vdots \\ [\tau_{r_1}]_{\bar{\mu}} + [\tau_{r_1}]_\mu \end{pmatrix}$
    if $\mathsf{rank}(A) < |R_{i-1}|$ *or* $\mathsf{rank}(\vec{b}) < |R_{i-1}|$ **then**
      return $\perp$
    for $r \| b \in R_i$ **do**
      $[z_{r\|b}]_{\bar{\mu}} \leftarrow \left\langle [X_r]_{\bar{\mu}}, \vec{d}_{i-1} \right\rangle + [\tau_r]_{\bar{\mu}} \cdot w_{i,b}$
      $[z_{r\|b}]_\mu \leftarrow \left\langle [X_r]_\mu, \vec{d}_{i-1} \right\rangle + [\tau_r]_\mu \cdot w_{i,b}$
      $z_{r\|b} \leftarrow [z_{r\|b}]_{\bar{\mu}} + [z_{r\|b}]_\mu$
return $\mathcal{A}(([X_\epsilon]_\mu, [\tau_\epsilon]_\mu, \{w_{i,0}\}_{i=1}^n, \{w_{i,1}\}_{i=1}^n, \{\vec{d}_i\}_{i=0}^{n-1}), \{[z_r]_{\bar{\mu}}\}_{r \in R_n})$

can sample the $\{[z_r]_{\bar{\mu}}\}_{r \in R_i}$ uniformly at random, which we do in Game 15. We also do not have to sample the A and $\vec{b}$ anymore. Since the only way how sampling A and $\vec{b}$ can affect the output is when they are not full-rank, we will now estimate the probability of this event.

Clearly,

$$0 \leq \Pr[\mathsf{rank}(A) < t \text{ or } \mathsf{rank}_2(\vec{b}) < t \text{ in Game 14}] \leq$$

$$\leq \Pr[\mathsf{rank}(A) < t \text{ in Game 14}] + \Pr[\mathsf{rank}_2(\vec{b}) < t \text{ in Game 14}]$$

As A is formed from these randomly sampled X_r, clearly it is a randomly chosen matrix, and is thus invertible with probability $\prod_{i=0}^{t-1}(1-2^{i-\bar{v}})$ by Lemma 2, hence, $\Pr[\mathsf{rank}(A) < t \text{ in Game } 14] = 1 - \prod_{i=0}^{t-1}(1-2^{i-\bar{v}})$.

As for $\vec{b}$, we note that we can first choose $\tau_r \cdot w$ at random and then compute A from that. Thus, also the probability that $\vec{b}$ is not full-rank is $1 - \prod_{i=0}^{t-1}(1-2^{i-v})$. Hence,

$$0 \leq \Pr[\mathsf{rank}(A) < t \text{ or } \mathsf{rank}_2(\vec{b}) < t \text{ in Game } 14] \leq 2 - 2\prod_{i=0}^{t-1}(1-2^{i-\bar{v}}).$$

Hence,

$$-(1-\prod_{i=0}^{t-1}(1-2^{i-\bar{v}})) \leq \Pr[\text{Game } 14 \text{ outputs } \perp \text{ at line } 23] -$$

$$-\Pr[\text{Game } 15 \text{ outputs } \perp \text{ at line } 12] \leq (1-\prod_{i=0}^{t-1}(1-2^{i-\bar{v}})).$$

Since, by the argument above, there are no more statistical differences between the games, we can estimate that

$$|\text{Game } 14 - \text{Game } 15| \leq (1-\prod_{i=0}^{t-1}(1-2^{i-\bar{v}})) \leq 2^{t-\bar{v}} \leq 2^{-k-\kappa-2} < 2^{-k-\kappa}.$$

Game 15:
Input : Adversary $\mathcal{A}$

$(\alpha_1, \beta_1), \ldots, (\alpha_t, \beta_t) \leftarrow \mathcal{A}()$
$[X_\epsilon]_\mu \xleftarrow{\$} \{0,1\}^v$
$[\tau_\epsilon]_\mu \xleftarrow{\$} \mathbb{F}_{2^k}$
for $i = 1$ **to** n **do**
  $\ldots$
  if $i < n$ **then**
    With probability $1 - \prod_{i=0}^{t-1}(1-2^{i-v})$ **return** $\perp$
    $\vec{d}_{i-1} \xleftarrow{\$} \mathbb{F}_{2^k}^v$
    for $r \| b \in R_i$ **do**
      $([X_{r\|b}]_\mu, [\tau_{r\|b}]_\mu) \leftarrow f\left(\left\langle [X_r]_\mu, \vec{d}_{i-1} \right\rangle + [\tau_r]_\mu \cdot w_{i,b}\right)$
  else
    With probability $1 - \prod_{i=0}^{t-1}(1-2^{i-\bar{v}})$ **return** $\perp$
    $\vec{d}_{n-1} \xleftarrow{\$} \mathbb{F}_{2^{\bar{v}}}^{\bar{v}}$ on the condition that $\mathsf{rank}_2(\vec{d}_{i-1}) = \bar{v}$
    for $r \| b \in R_i$ **do**
      $[z_{r\|b}]_{\bar{\mu}} \xleftarrow{\$} \mathbb{F}_{2^{\bar{v}}}$
      $[z_{r\|b}]_\mu \leftarrow \left\langle [X_r]_\mu, \vec{d}_{i-1} \right\rangle + [\tau_r]_\mu \cdot w_{i,b}$
      $z_{r\|b} \leftarrow [z_{r\|b}]_{\bar{\mu}} + [z_{r\|b}]_\mu$
    $\mathsf{state} \leftarrow R_i, \{[z_r]_{\bar{\mu}}\}_{r\in R_i}, \{[z_r]_\mu\}_{r\in R_i}, \{z_r\}_{r\in R_i}$
return $\mathcal{A}(([X_\epsilon]_\mu, [\tau_\epsilon]_\mu, \{w_{i,0}\}_{i=1}^n, \{w_{i,1}\}_{i=1}^n, \{\vec{d}_i\}_{i=0}^{n-1}), \{[z_r]_{\bar{\mu}}\}_{r\in R_n})$

We note now that the adversary does not obtain anything that depends on the $[X_r]_\mu, [\tau_r]_\mu$ or $[z_r]_\mu$, and thus, we can omit all of those lines. Thus, we obtain Game 16 and we have that Game 15 = Game 16.

Game 16:
Input : Adversary $\mathcal{A}$

$(\alpha_1, \beta_1), \ldots, (\alpha_t, \beta_t) \leftarrow \mathcal{A}()$
$[X_\epsilon]_\mu \xleftarrow{\$} \{0,1\}^v$
$[\tau_\epsilon]_\mu \xleftarrow{\$} \mathbb{F}_{2^k}$
for $i = 1$ **to** $n-2$ **do**
  $w_{i,0} \xleftarrow{\$} \mathbb{F}_{2^k} \setminus \{0\}$
  $w_{i,1} \xleftarrow{\$} \mathbb{F}_{2^k} \setminus \{0, w_{i,0}\}$
  $R_i \leftarrow \emptyset$
  $R'_i \leftarrow \emptyset$
  $\widehat{R}_{i-1} \leftarrow \emptyset$
  for $r \in R_{i-1}$ **do**
    for $b \in \{0,1\}$ **do**
      if $\exists \alpha_j$ *such that* α_j *starts with* $r \| b$ **then**
        $R_i \leftarrow R_i \cup \{r\|b\}$
      else
        $R'_i \leftarrow R'_i \cup \{r\|b\}$
    if $r\|0 \in R_i$ *and* $r\|1 \in R_i$ **then**
      $\widehat{R}_{i-1} \leftarrow \widehat{R}_{i-1} \cup \{r\}$
  $\vec{d}_{i-1} \xleftarrow{\$} \mathbb{F}_{2^k}^v$
  if $i = n$ **then**
    With probability $1 - \prod_{i=0}^{t-1}(1 - 2^{i-\bar{v}})$ **return** $\perp$
    $\vec{d}_{n-1} \xleftarrow{\$} \mathbb{F}_{2^{\bar{v}}}^{\bar{v}}$ on the condition that $\mathsf{rank}_2(\vec{d}_{i-1}) = \bar{v}$
    for $r\|b \in R_i$ **do**
      $[z_{r\|b}]_{\bar{\mu}} \xleftarrow{\$} \mathbb{F}_{2^k}$
return $\mathcal{A}(([X_\epsilon]_\mu, [\tau_\epsilon]_\mu, \{w_{i,0}\}_{i=1}^n, \{w_{i,1}\}_{i=1}^n, \{\vec{d}_i\}_{i=0}^{n-1}), \{[z_r]_{\bar{\mu}}\}_{r \in R_n})$

We now note that the output does not depend on the R_i and they are only used to index the $[z_{r\|b}]_{\bar{\mu}}$ in the last level. We will now omit these and set the indices of the $[z_{r\|b}]_{\bar{\mu}}$ to be instead the set $\{0, \ldots, t-1\}$. We thus obtain Game 17.

Note that Game 17 = Game 4. This means we have found the computational distance that we required as all of the above summarises to comparison of Game 4 and Game 3:

$$|\Pr[\text{Game } 3(\mathcal{A}) = 1] - \Pr[\text{Game } 4(\mathcal{A}) = 1]| \leq t \cdot n \cdot \varepsilon_{\mathsf{PRG}} + 2^{-\kappa-k}.$$

□

Game 17:
Input : Adversary $\mathcal{A}$

$(\alpha_1, \ldots, \alpha_t) \leftarrow \mathcal{A}()$
$[X_\epsilon]_\mu \xleftarrow{\$} \{0,1\}^v$
$[\tau_\epsilon]_\mu \xleftarrow{\$} \mathbb{F}_{2^k}$
for $i = 1$ **to** $n-2$ **do**
  $w_{i,0} \xleftarrow{\$} \mathbb{F}_{2^k} \setminus \{0\}$
  $w_{i,1} \xleftarrow{\$} \mathbb{F}_{2^k} \setminus \{0, w_{i,0}\}$
  With probability $1 - \prod_{i=0}^{t-1}(1 - 2^{i-v})$ **return** $\perp$
  $\vec{d}_{i-1} \xleftarrow{\$} \mathbb{F}_{2^k}^v$
  if $i = n$ **then**
    for $j \in \{0, \ldots, t-1\}$ **do**
      $[z_j]_{\bar{\mu}} \xleftarrow{\$} \mathbb{F}_{2^k}$
return $\mathcal{A}(([X_\epsilon]_\mu, [\tau_\epsilon]_\mu, \{w_{i,0}\}_{i=1}^n, \{w_{i,1}\}_{i=1}^n, \{\vec{d}_i\}_{i=0}^{n-1}), \{[z_r]_{\bar{\mu}}\}_{r \in R_n})$

In Theorem 5, we show that if we have a random-DMPF that is ε_r-close to random, then using it as a black box inside the Algorithm 8 and Algorithm 9, where it is called as GenRand, we obtain a secure FSS scheme. As before, this proof is a sequence of games (seen in Figure 6) relating SLAMP-FSS with a general random DPMF scheme to the security definition for FSS schemes in Definition 8. Note that these algorithms generalise the SLAMP-FSS constructions in Algorithm 3 and Algorithm 5.

$$\mathcal{G}_{18} \equiv \mathcal{G}_{21} \overset{\varepsilon_r}{\approx} \mathcal{G}_{22} \equiv \mathcal{G}_{23} \equiv \mathcal{G}_{24} \overset{2^{-\kappa-2}}{\approx} \mathcal{G}_{25} \overset{2^{-\kappa-1}}{\approx} \mathcal{G}_{26} \equiv \mathcal{G}_{20} \overset{\varepsilon_r + 3 \cdot 2^{-\kappa-2}}{\approx} \mathcal{G}_{19}$$

Figure 6. Relations between the games of Theorem 5. $\mathcal{G}_i$ corresponds to Game i, $\mathcal{G}_i \equiv \mathcal{G}_j$ means that games i and j are equivalent, and $\mathcal{G}_i \overset{\varepsilon}{\approx} \mathcal{G}_j$ means that they are ε-close.

The proof starts from the FSS security definition (Definition 8) restated as a pair of games Game 18 and Game 19. The goal then becomes to transform these games into a game where the output of Gen is independent from its input (Game 20). For this, we inline key generation and use the security property of the random-DMPF scheme (Definition 14) to randomise the keys to an extent (Game 21 and Game 22). We then replace Z with a uniformly random matrix and remove dependencies between $\vec{g}$ and $\vec{\beta}$ (Game 23, Game 24, Game 25 and Game 26), and thoroughly examine the security loss that these steps introduce. The transformation is given from Game 18 to Game 20, however, it is symmetric to the transformation between Game 19 and Game 20.

Algorithm 8: General FSS key generation algorithm
Input : $(\alpha_1, \ldots, \alpha_t), (\beta_1, \ldots, \beta_t)$

$\mathsf{sk}'_1, \mathsf{sk}'_2, \mathsf{state} \leftarrow \mathsf{GeneralGenRand}((\alpha_1, \ldots, \alpha_t))$
$\mathsf{sk}''_1, \mathsf{sk}''_2 \leftarrow \mathsf{DeRand}((\beta_1, \ldots, \beta_t), \mathsf{state})$
$\mathsf{sk}_1 \leftarrow \mathsf{sk}'_1, \mathsf{sk}''_1$
$\mathsf{sk}_2 \leftarrow \mathsf{sk}'_2, \mathsf{sk}''_2$
Give to $\mathcal{P}_1$ the key sk_1
Give to $\mathcal{P}_2$ the key sk_2

Algorithm 9: General Evaluation Algorithm $\mathsf{GeneralEval}^{\mathsf{p}}$

Input : $\mathsf{sk}_{\mathsf{p}} = (\mathsf{sk}'_{\mathsf{p}}, \mathsf{sk}''_{\mathsf{p}}), x$

$z_{\mathsf{p}} \leftarrow \mathsf{GeneralRandEval}^{\mathsf{p}}(\mathsf{sk}'_{\mathsf{p}}, x)$

$y_{\mathsf{p}} \leftarrow \mathsf{DeRandEval}^{\mathsf{p}}(z_{\mathsf{p}}, \mathsf{sk}''_{\mathsf{p}} = \vec{g})$

return y_{p}

Game 18:

$(\alpha_1^0, \beta_1^0), \ldots, (\alpha_t^0, \beta_t^0), (\alpha_1^1, \beta_1^1), \ldots, (\alpha_t^1, \beta_t^1), \sigma, \mu \leftarrow \mathcal{A}()$

$c \leftarrow 0$

$(\alpha_1, \beta_1), \ldots, (\alpha_t, \beta_t) \leftarrow (\alpha_1^c, \beta_1^c), \ldots, (\alpha_t^c, \beta_t^c)$

$\mathsf{sk}_1, \mathsf{sk}_2, \leftarrow \mathsf{Gen}((\alpha_1^c, \beta_1^c), \ldots, (\alpha_t^c, \beta_t^c))$

return $\mathcal{A}(\mathsf{sk}_\mu, \sigma)$

Theorem 5. *Let $k+t+\kappa+2 \leq \bar{v}$. Let us have Algorithms* GeneralGenRand, GeneralRandEval *with the following properties. For a random-DMPF scheme, key generation* GeneralGenRand *takes in a set of t points $(\alpha_1, \ldots, \alpha_t) \in (\{0,1\}^n)^t$ and outputs $\mathsf{sk}'_1, \mathsf{sk}'_2, \mathsf{state}$ where $\mathsf{state} = (\{\alpha_i\}_{i=1}^t, \{[z_i]_1\}_{i=1}^t, \{[z_i]_2\}_{i=1}^t, \{z_i\}_{i=1}^t)$, where for all i, $[z_i]_1, [z_i]_2 \in \mathbb{F}_{2^{\bar{v}}}$. For any $x \in \{0,1\}^n$, it holds that*

$$\mathsf{GeneralRandEval}_1(\mathsf{sk}'_1, x) + \mathsf{GeneralRandEval}_2(\mathsf{sk}'_2, x) = \begin{cases} z_j, & \text{if } \exists j \text{ such that } x = \alpha_j, \\ 0, & \text{otherwise.} \end{cases}$$

Also, for all $j \in \{1, \ldots, t\}$, $\mathsf{GeneralRandEval}_1(\mathsf{sk}'_1, \alpha_j) = [z_j]_1$ and $\mathsf{GeneralRandEval}_2(\mathsf{sk}'_2, \alpha_j) = [z_j]_2$. Let $(\mathsf{GeneralGenRand}, \mathsf{GeneralRandEval})$ be ε_r-close to random. The output of GeneralEval *in Algorithm 9 is in $\mathbb{F}_{2^k}$.*

Then Algorithm 8 and Algorithm 9 form a $2\varepsilon_{ROR}$-secure FSS scheme where $\varepsilon_{ROR} = \varepsilon_r + 3 \cdot 2^{-\kappa-2}$

Proof. The security of an FSS scheme is defined to be $\bar{\varepsilon}$-secure in Definition 8, where the adversary provides two functions φ_0, φ_1 and chooses a party $\mathcal{P}_\mu \in \{\mathcal{P}_1, \mathcal{P}_2\}$, to corrupt, the generating process creates two keys $\mathsf{sk}_1, \mathsf{sk}_2$ that correspond to φ_c where $c \in \{0,1\}$. The adversary is given the key sk_μ to and it cannot guess c better than $\frac{1}{2} + \bar{\varepsilon}$. In our case, we can state it as $|\text{Game } 18 - \text{Game } 19| \leq \bar{\varepsilon}$.

To show that this is the case, it suffices to argue that there is a function GenSecond that does not take the φ_1 nor φ_2 as an input, that produces $\mathsf{sk}_1, \mathsf{sk}_2$, where the adversary is given sk_μ, and where the adversary cannot distinguish this from games 18, or 19, that is, in our case, $|\text{Game } 18 - \text{Game } 20| \leq \frac{\bar{\varepsilon}}{2}$ and $|\text{Game } 20 - \text{Game } 19| \leq \frac{\bar{\varepsilon}}{2}$. Since these two cases are symmetric, it is sufficient to just show one of these inequalities.

Game 19:

$(\alpha_1^0, \beta_1^0), \ldots, (\alpha_t^0, \beta_t^0), (\alpha_1^1, \beta_1^1), \ldots, (\alpha_t^1, \beta_t^1), \sigma, \mu \leftarrow \mathcal{A}()$

$c \leftarrow 1$

$(\alpha_1, \beta_1), \ldots, (\alpha_t, \beta_t) \leftarrow (\alpha_1^c, \beta_1^c), \ldots, (\alpha_t^c, \beta_t^c)$

$\mathsf{sk}_1, \mathsf{sk}_2, \leftarrow \mathsf{Gen}((\alpha_1^c, \beta_1^c), \ldots, (\alpha_t^c, \beta_t^c))$

return $\mathcal{A}(\mathsf{sk}_\mu, \sigma)$

We thus expand Game 18 into Game 21 and Game 20 into Game 26. For our claim it thus suffices to show that $|\text{Game } 21 - \text{Game } 26| \leq \varepsilon_{ROR}$.

By definition of ε_r-close to random, we can replace a call to GenRand with a call to a function GenPrim and the adversary can only notice this change with probability less

Game 20:

$(\alpha_1^0, \beta_1^0), \ldots, (\alpha_t^0, \beta_t^0), (\alpha_1^1, \beta_1^1), \ldots, (\alpha_t^1, \beta_t^1), \sigma, \mu \leftarrow \mathcal{A}()$
$\mathsf{sk}_1, \mathsf{sk}_2, \leftarrow \mathsf{GenSecond}()$
return $\mathcal{A}(\mathsf{sk}_\mu, \sigma)$

Game 21:

$(\alpha_1^0, \beta_1^0), \ldots, (\alpha_t^0, \beta_t^0), (\alpha_1^1, \beta_1^1), \ldots, (\alpha_t^1, \beta_t^1), \sigma \leftarrow \mathcal{A}()$
$c \leftarrow 0$
$(\alpha_1, \beta_1), \ldots, (\alpha_t, \beta_t) \leftarrow (\alpha_1^c, \beta_1^c), \ldots, (\alpha_t^c, \beta_t^c)$
$\mathsf{sk}_1, \mathsf{sk}_2, \{([z_1]_1, \ldots, [z_t]_1)\}, \{([z_1]_2, \ldots, [z_t]_2)\} \leftarrow \mathsf{GenRand}((\alpha_1^c, \beta_1^c), \ldots, (\alpha_t^c, \beta_t^c), f)$
$\mu \in \{1, 2\} \leftarrow \mathcal{A}()$
$\bar{\mu} \leftarrow 3 - \mu$
$\vec{\beta} \leftarrow \{(\beta_1^c, \ldots, \beta_t^c)\}$
for $j = 1$ **to** t **do**
  $\{[z_{i,j}]_{\bar{\mu}}\}_{i=1}^{\bar{v}} \leftarrow \mathsf{Bits}([z_j]_{\bar{\mu}})$
  $\{[z_{i,j}]_{\mu}\}_{i=1}^{\bar{v}} \leftarrow \mathsf{Bits}([z_j]_{\mu})$
for $i = 1$ **to** t **do**
  for $j = 1$ **to** $\bar{v}$ **do**
    $z_{i,j} \leftarrow [z_{i,j}]_{\bar{\mu}} + [z_{i,j}]_{\mu}$
$Z \leftarrow \{z_{i,j}\}_{i=1,j=1}^{t,\bar{v}}$
if $\mathsf{rank}(Z) < t$ **then**
  return $\perp$
Solve $Z\vec{g} = \vec{\beta}$ and sample $\vec{g}$ from the solution space
return $\mathcal{A}(\mathsf{sk}_\mu, \vec{g})$

than ε_r. Thus, our first game-hop is replacing the call to GenRand in Game 21 to a call to GenPrim in Game 22.

We now show that |Game 21−Game 22| $\leq \varepsilon_r$. By our assumption, that (GeneralGenRand, GeneralRandEval) is ε_r-close to random,

$$|\Pr[\text{Game } 1(\mathcal{A}) = 1] - \Pr[\text{Game } 2(\mathcal{A}) = 1]| \leq \varepsilon_r.$$

Recall that as shorthand we also write |Game 1 − Game 2| $\leq \varepsilon_r$. Suppose, by contradiction, that there is an adversary $\mathcal{B}$ who can distinguish games 21 and 22 better than ε_r. We show how to construct an adversary $\mathcal{A}$ who can distinguish the games Game 1 and Game 2 better than ε_r. This adversary is given in Algorithm 10. This $\mathcal{A}$ first gets $(\alpha_1^0, \beta_1^0), \ldots, (\alpha_t^0, \beta_t^0), (\alpha_1^1, \beta_1^1), \ldots, (\alpha_t^1, \beta_t^1), \mu$ from $\mathcal{B}$. They set the bit c as 0. Then, in the distinguishing game between Game 1 and Game 2 they submit $(\alpha_1^c, \ldots, \alpha_t^c)$.

Either Gen or GenPrim outputs $\mathsf{sk}_1, \mathsf{sk}_2$, depending on whether we are playing Game 1 or Game 2, respectively. They submit μ and get back $\mathsf{sk}_\mu, \{[z_i]_{\bar{\mu}}\}_{j=1}^t$. Then they compute $[z_j]_\mu$ by calling the evaluation algorithm on $\{\alpha_j^c\}_{j=1}^t$ using sk_μ. They then obtain $\{[z_{i,j}]_\mu\}_{i=1}^{\bar{v}}$ from the bits of the $\{[z_j]_\mu\}_{j=1}^t$ and $\{[z_{i,j}]_{\bar{\mu}}\}_{i=1}^{\bar{v}}$ from the bits of the $\{[z_j]_{\bar{\mu}}\}_{j=1}^t$. They then compute $\vec{g}$ exactly as it is computed in Games 21 and 26. Now clearly, if the game being played by $\mathcal{A}$ is 1, the distribution of $\mathsf{sk}_\mu, \vec{g}$ is the same as in Game 21. If the game being played by $\mathcal{A}$ is 2, the distribution of $\mathsf{sk}_\mu, \vec{g}$ is the same as in Game 26. Thus, if $\mathcal{B}$ can distinguish between these two games better than ε_r, then also $\mathcal{A}$ can distinguish between Game 21 and Game 26 better than ε_r which contradicts our assumption. Thus,

$$|\text{Game } 21 - \text{Game } 22| \leq \varepsilon_r.$$

Game 22:

$(\alpha_1^0, \beta_1^0), \ldots, (\alpha_t^0, \beta_t^0), (\alpha_1^1, \beta_1^1), \ldots, (\alpha_t^1, \beta_t^1), \sigma \leftarrow \mathcal{A}()$
$c \leftarrow 0$
$(\alpha_1, \beta_1), \ldots, (\alpha_t, \beta_t) \leftarrow (\alpha_1^c, \beta_1^c), \ldots, (\alpha_t^c, \beta_t^c)$
$\mathsf{sk}_1, \mathsf{sk}_2, \{([z_1]_1, \ldots, [z_t]_1)\}, \{([z_1]_2, \ldots, [z_t]_2)\} \leftarrow \mathsf{GenPrim}()$
$\mu \in \{1, 2\} \leftarrow \mathcal{A}()$
$\bar{\mu} \leftarrow 3 - \mu$
$\vec{\beta} \leftarrow \{(\beta_1^c, \ldots, \beta_t^c)\}$
for $j = 1$ **to** t **do**
  $\{[z_{i,j}]_{\bar{\mu}}\}_{i=1}^{\bar{v}} \leftarrow \mathsf{Bits}([z_j]_{\bar{\mu}})$
  $\{[z_{i,j}]_{\mu}\}_{i=1}^{\bar{v}} \leftarrow \mathsf{Bits}([z_j]_{\mu})$
for $i = 1$ **to** t **do**
  for $j = 1$ **to** $\bar{v}$ **do**
    $z_{i,j} \leftarrow [z_{i,j}]_{\bar{\mu}} + [z_{i,j}]_{\mu}$
$Z \leftarrow \{z_{i,j}\}_{i=1,j=1}^{t,\bar{v}}$
if $\mathsf{rank}_2(Z) < t$ **then**
  return $\perp$
Solve $Z\vec{g} = \vec{\beta}$ and sample $\vec{g}$ from the solution space
return $\mathcal{A}(\mathsf{sk}_{\mu}, \vec{g})$

We note that since in GenPrim, the values $[z_j]_{\bar{\mu}}$ are chosen randomly and independently from anything else, we can essentially move this sampling outside of the GenPrim. We thus, in Game 23, do not use the $[z_j]_{\bar{\mu}}$ that came out of GenPrim and instead resample them randomly in the body of our algorithm. Since the $[z_j]_{\bar{\mu}}$ are sampled randomly and independently in both cases, we have

$$\text{Game } 22 = \text{Game } 23$$

where we use to shorthand Game 22 = Game 23 for |Game 22 − Game 23| = 0.

We note that the computation of $z_{i,j}$ in Game 23 is equivalent to just choosing the bits $z_{i,j}$ randomly. Thus, in Game 24, we simply sample the bits $z_{i,j}$ randomly.

We have that

$$\text{Game } 23 = \text{Game } 24.$$

Now let us add the constraint that $\vec{g}$ must be full-rank.

As the next step, we add the step where we return $\perp$ if $\vec{g}$ is not full rank. By Lemma 9 we have that the probability that $\vec{g}$ is full-rank is at least $\prod_{i=0}^{t-1}(1 - 2^{i-\bar{v}+k-u}) \geq 1 - 2^{t+k-\bar{v}-u}$ (where the inequality is due to Lemma 6), where $u = \mathsf{rank}_2(\vec{\beta})$. Thus, the probability that $\vec{g}$ is not full-rank is at most

$$2^{t+k-\bar{v}-u} \leq 2^{t+k-\bar{v}} \leq 2^{-\kappa-2}. \tag{2}$$

Thus,

$$|\text{Game } 24 - \text{Game } 25| \leq 2^{-\kappa-2}.$$

We will now find the statistical distance between Games 25 and Game 26.

We note that the only thing that differs between these two games is that the value $\vec{g}$ is sampled differently, and the way one can obtain $\perp$ is different. First notice that in Game 25, we return $\perp$ if the randomly sampled Z is not full-rank, which happens with probability $1 - \prod_{i=0}^{t-1}(1 - 2^{i-\bar{v}})$, which corresponds to the line 10 in Game 26. Concerning the probability of returning $\perp$ due to $\vec{g}$ not being full-rank, by Equation 2, it is at most

Algorithm 10: Adversary for distinguishing Games 1 and 2

Get $(\alpha_1^0, \beta_1^0), \ldots, (\alpha_t^0, \beta_t^0), (\alpha_1^1, \beta_1^1), \ldots, (\alpha_t^1, \beta_t^1)$ from $\mathcal{B}()$
$c \leftarrow 0$
Submit $(\alpha_1^c, \ldots, \alpha_t^c)$ to the distinguishing game
Get μ from $\mathcal{B}()$
Submit μ to the distinguishing game
Get $\mathsf{sk}_\mu, \{[z_j]_{\bar{\mu}}\}_{j=1}^t$ from the distinguishing game [where $\mathsf{sk}_1, \mathsf{sk}_2 \leftarrow (\mathsf{Gen}((\alpha_1^c, \ldots, \alpha_t^c), f)$ or $\mathsf{GenPrim}(f))$]
$\vec{\beta} \leftarrow \{(\beta_1^c, \ldots, \beta_t^c)\}$
for $j = 1$ **to** t **do**
    $[z_j]_\mu \leftarrow \mathsf{EvalRand}^\mu(\mathsf{sk}_\mu, f, \alpha_j^c)$
    $\{[z_{i,j}]_\mu\}_{i=1}^{\bar{v}} \leftarrow \mathsf{Bits}([z_j]_\mu)$
    $\{[z_{i,j}]_{\bar{\mu}}\}_{i=1}^{\bar{v}} \leftarrow \mathsf{Bits}([z_j]_{\bar{\mu}})$
for $i = 1$ **to** t **do**
    for $j = 1$ **to** $\bar{v}$ **do**
        $z_{i,j} \leftarrow [z_{i,j}]_1 + [z_{i,j}]_2$
$Z \leftarrow \{z_{i,j}\}_{i=1,j=1}^{t,\bar{v}}$
if $\mathsf{rank}(Z) < t$ **then**
    return $\perp$
Solve $Z\vec{g} = \vec{\beta}$ and sample $\vec{g}$ from the solution space
Give $\mathsf{sk}_\mu, \vec{g}$ to $\mathcal{B}()$
Get $\tilde{c}$ from $\mathcal{B}(\mathsf{sk}_\mu, \vec{g})$
return $\tilde{c}$

$2^{-\kappa-2}$ in Game 25 and is equal to $1 - \prod_{i=0}^{k-1}(1 - 2^{i-\bar{v}}) \leq 2^{k-\bar{v}} \leq 2^{-\kappa-2-t}$ in Game 26. Thus, both error probabilities can be lower-bounded by 0 and upper-bounded by $2^{-\kappa-2}$. Thus, the difference of the probabilities of outputting $\perp$ in these two games is at most $2^{-\kappa-2}$.

We now argue that every full-rank $\vec{g}$ has an equal probability of happening for a fixed $\vec{\beta}$. Let $\bar{v} \geq k$. Denote by $R(B, k, \bar{v}, s)$ the number of $\bar{v} \times k$ binary full-rank matrices where the upper s rows consist of the $s \times k$ matrix B. Note that since rank is preserved after transposing, we denote by $R(B, k, \bar{v}, s)$ also the number of $k \times \bar{v}$ binary full-rank matrices where the first s columns consist of the $k \times s$ matrix B.

Let $B = \mathsf{Bits}(\vec{\beta})$. Fix a full-rank $\vec{g}$. Let $G = \mathsf{Bits}(\vec{g})$. We can express $G = Y_G \begin{pmatrix} I_k \\ 0 \end{pmatrix}$ for some invertible $Y_G \in \mathbb{F}_2^{\bar{v} \times \bar{v}}$.

Hence, the equation becomes $ZY_G \begin{pmatrix} I_k \\ 0 \end{pmatrix} = B$. Now, we note that for any Z, denoting $ZY_G = \begin{pmatrix} Z_{1,Y_G} & Z_{2,Y_G} \end{pmatrix}$, we get that $ZY_G \begin{pmatrix} I_k \\ 0 \end{pmatrix} = B$ iff $Z_{1,Y_G} = B$, and that Z is full-rank iff $\begin{pmatrix} B & Z_{2,Y_G} \end{pmatrix}$ is full-rank. Here $Z_{1,Y_G} \in \mathbb{F}_2^{t \times k}$ and $Z_{2,Y_G} \in \mathbb{F}_2^{t \times \bar{v}-k}$.

Note that if $\begin{pmatrix} B & Z_{2,Y_G} \end{pmatrix} Y_G^{-1} = \begin{pmatrix} B & Z'_{2,Y_G} \end{pmatrix} Y_G^{-1}$, then $Z_{2,Y_G} = Z'_{2,Y_G}$. (Indeed, we note that the equation is equivalent to $\begin{pmatrix} B - B & Z_{2,Y_G} - Z'_{2,Y_G} \end{pmatrix} Y_G^{-1} = 0$, that is $\begin{pmatrix} 0 & Z_{2,Y_G} - Z'_{2,Y_G} \end{pmatrix} = 0Y_G = 0$, hence, $Z_{2,Y_G} - Z'_{2,Y_G} = 0$.) Hence, for every fixed B and Y_G there are as many choices for Z as there are for Z_{2,Y_G}, which is $2^{t(\bar{v}-k)}$, and all the Z are equally likely.

We also note that if we can express $G = Y_G \begin{pmatrix} I_k \\ 0 \end{pmatrix}$ and $G = Y'_G \begin{pmatrix} I_k \\ 0 \end{pmatrix}$ for some different

Game 23:

$(\alpha_1^0, \beta_1^0), \ldots, (\alpha_t^0, \beta_t^0), (\alpha_1^1, \beta_1^1), \ldots, (\alpha_t^1, \beta_t^1), \sigma \leftarrow \mathcal{A}()$
$c \leftarrow 0$
$(\alpha_1, \beta_1), \ldots, (\alpha_t, \beta_t) \leftarrow (\alpha_1^c, \beta_1^c), \ldots, (\alpha_t^c, \beta_t^c)$
$\mathsf{sk}_1, \mathsf{sk}_2, \{([z_1]_1, \ldots, [z_t]_1)\}, \{([z_1]_2, \ldots, [z_t]_2)\} \leftarrow \mathsf{GenPrim}()$
$\mu \in \{1, 2\} \leftarrow \mathcal{A}()$
$\bar{\mu} \leftarrow 3 - \mu$
$\vec{\beta} \leftarrow \{(\beta_1^c, \ldots, \beta_t^c)\}$
for $j = 1$ **to** t **do**
  $[z_j]_{\bar{\mu}} \xleftarrow{\$} \mathbb{F}_{2^{\bar{v}}}$
  $\{[z_{i,j}]_{\bar{\mu}}\}_{i=1}^{\bar{v}} \leftarrow \mathsf{Bits}([z_j]_{\bar{\mu}})$
  $\{[z_{i,j}]_{\mu}\}_{i=1}^{\bar{v}} \leftarrow \mathsf{Bits}([z_j]_{\mu})$
for $i = 1$ **to** t **do**
  for $j = 1$ **to** $\bar{v}$ **do**
    $z_{i,j} \leftarrow [z_{i,j}]_{\bar{\mu}} + [z_{i,j}]_{\mu}$
$Z \leftarrow \{z_{i,j}\}_{i=1,j=1}^{t,\bar{v}}$
if $\mathsf{rank}_2(Z) < t$ **then**
  return $\perp$
Solve $Z\vec{g} = \vec{\beta}$ and sample $\vec{g}$ from the solution space
return $\mathcal{A}(\mathsf{sk}_{\mu}, \vec{g})$

invertible matrices Y_G and Y'_G, then for any Z, $ZY_G \begin{pmatrix} I_k \\ 0 \end{pmatrix} = B$ iff $ZY'_G \begin{pmatrix} I_k \\ 0 \end{pmatrix} = B$, because $ZY_G \begin{pmatrix} I_k \\ 0 \end{pmatrix} = ZG = ZY'_G \begin{pmatrix} I_k \\ 0 \end{pmatrix}$. Thus, the choice of Y_G does not matter here.

Thus, for a given full-rank G there are $R(B, t, \bar{v}, k)$ values of full-rank Z when $ZG = B$. Indeed, the number of full-rank Z is the number of $t \times \bar{v} - k$ matrices Z_{2,Y_G} such that $\begin{pmatrix} B & Z_{2,Y_G} \end{pmatrix}$ is full-rank, which is how we defined $R(B, t, \bar{v}, k)$.

Symmetrically, for a fixed Z and B, stating $Z = \begin{pmatrix} I_t & 0 \end{pmatrix} W_Z$, we have that G is a solution to $ZG = B$ iff $W_Z G = \begin{pmatrix} B \\ G_2 \end{pmatrix}$ for an arbitrary $G_2 \in \mathbb{F}_2^{\bar{v}-t \times k}$, and to every choice of G_2 corresponds a different G. We also note that if we can express $Z = \begin{pmatrix} I_t & 0 \end{pmatrix} W_Z$ and $Z = \begin{pmatrix} I_t & 0 \end{pmatrix} W'_Z$ for some different invertible matrices W_Z and W'_Z, then for any G, $\begin{pmatrix} I_t & 0 \end{pmatrix} W_Z G = B$ iff $\begin{pmatrix} I_t & 0 \end{pmatrix} W'_Z G = B$. Thus, analogously, the choice of W_Z does not matter here. Hence, we have that for given B and full-rank Z there are $R(B, k, \bar{v}, t)$ full-rank solutions G to $ZG = B$.

Hence, for any full-rank $\tilde{g}$, with $\mathsf{Bits}(\tilde{g}) = Y_G \begin{pmatrix} I_k \\ 0 \end{pmatrix}$, we have that in Game 25,

$$\Pr[\vec{g} = \tilde{g}] = \sum_{\substack{Z \in \{0,1\}^{t \times \bar{v}} \\ \mathsf{rank}(Z) = t \\ \exists Z_2 : ZY_G = \begin{pmatrix} B & Z_2 \end{pmatrix}}} \Pr_{\substack{\vec{g} \xleftarrow{\$} \text{full-rank} \\ \text{solutions to} \\ Z\vec{g} = \vec{\beta}}} \left[\vec{g} = \tilde{g} \mid \vec{\beta} \cap Z \right] =$$

$$= \sum_{\substack{Z \in \{0,1\}^{t \times \bar{v}} \\ \mathsf{rank}(Z) = t \\ \exists Z_2 : ZY_G = \begin{pmatrix} B & Z_2 \end{pmatrix}}} \frac{1}{R(B, k, \bar{v}, t)} = \frac{R(B, t, \bar{v}, k)}{R(B, k, \bar{v}, t)}.$$

We see that this does not depend on $\tilde{g}$ and must therefore be the same for all full-rank

Game 24:

$(\alpha_1^0, \beta_1^0), \ldots, (\alpha_t^0, \beta_t^0), (\alpha_1^1, \beta_1^1), \ldots, (\alpha_t^1, \beta_t^1), \sigma \leftarrow \mathcal{A}()$
$c \leftarrow 0$
$(\alpha_1, \beta_1), \ldots, (\alpha_t, \beta_t) \leftarrow (\alpha_1^c, \beta_1^c), \ldots, (\alpha_t^c, \beta_t^c)$
$\mathsf{sk}_1, \mathsf{sk}_2, \{([z_1]_1, \ldots, [z_t]_1)\}, \{([z_1]_2, \ldots, [z_t]_2)\} \leftarrow \mathsf{GenPrim}()$
$\mu \in \{1, 2\} \leftarrow \mathcal{A}()$
$\bar{\mu} \leftarrow 3 - \mu$
$\vec{\beta} \leftarrow \{(\beta_1^c, \ldots, \beta_t^c)\}$
for $i = 1$ **to** t **do**
  for $j = 1$ **to** $\bar{v}$ **do**
    $z_{i,j} \xleftarrow{\$} \{0, 1\}$
$Z \leftarrow \{z_{i,j}\}_{i=1,j=1}^{t,\bar{v}}$
if $\mathsf{rank}_2(Z) < t$ **then**
  return $\perp$
Solve $Z\vec{g} = \vec{\beta}$ and sample $\vec{g}$ from the solution space
return $\mathcal{A}(\mathsf{sk}_\mu, \vec{g})$

Game 25:

...
if $\mathsf{rank}_2(Z) < t$ **then**
  return $\perp$
Solve $Z\vec{g} = \vec{\beta}$ and sample $\vec{g}$ from the solution space
if $\mathsf{rank}_2(\vec{g}) < k$ **then**
  return $\perp$
return $\mathcal{A}(\mathsf{sk}_\mu, \vec{g})$

solutions.

Denote $\rho_1 := \Pr[\text{Game } 26 = \perp]$ and $\rho_2 := \Pr[\text{Game } 25 = \perp]$. Hence,

$$\begin{aligned}
|\text{Game } 26 - \text{Game } 25| &= |\Pr[\text{Game } 26 = \perp] - \Pr[\text{Game } 25 = \perp]| + \\
&+ \sum_{\substack{\vec{g} \in \mathbb{F}_2^{k \times \bar{v}} \\ \mathsf{rank}_2(\vec{g}) = k}} |\Pr[\text{Game } 26 = \vec{g}] - \Pr[\text{Game } 25 = \vec{g}]| = \\
&= |\rho_1 - \rho_2| + \sum_{\substack{\vec{g} \in \mathbb{F}_2^{k \times \bar{v}} \\ \mathsf{rank}_2(\vec{g}) = k}} \left| \frac{1 - \rho_1}{\prod_{i=0}^{k-1}(1 - 2^{i-\bar{v}})} - \frac{1 - \rho_2}{\prod_{i=0}^{k-1}(1 - 2^{i-\bar{v}})} \right| = \\
&= |\rho_1 - \rho_2| + \frac{\prod_{i=0}^{k-1}(1 - 2^{i-\bar{v}})}{\prod_{i=0}^{k-1}(1 - 2^{i-\bar{v}})} |\rho_1 - \rho_2| = 2|\rho_1 - \rho_2| \leq 2 \cdot 2^{-\kappa-2}
\end{aligned}$$

Thus, in conclusion, we obtain that

$$|\text{Game } 21 - \text{Game } 26| \leq \varepsilon_r + 3 \cdot 2^{-\kappa-2}$$

□

As the last step, we want to show that it is not possible to tell any information about α_j or β_j, for any $j \in \{1, \ldots, t\}$, from the viewpoint of one party. This is achieved in Theorem 6.

Game 26:

$(\alpha_1^0, \beta_1^0), \ldots, (\alpha_t^0, \beta_t^0), (\alpha_1^1, \beta_1^1), \ldots, (\alpha_t^1, \beta_t^1), \sigma \leftarrow \mathcal{A}()$
$c \leftarrow 0$
$(\alpha_1, \beta_1), \ldots, (\alpha_t, \beta_t) \leftarrow (\alpha_1^c, \beta_1^c), \ldots, (\alpha_t^c, \beta_t^c)$
$\mathsf{sk}_1, \mathsf{sk}_2, \{([z_1]_1, \ldots, [z_t]_1)\}, \{([z_1]_2, \ldots, [z_t]_2)\} \leftarrow \mathsf{GenPrim}()$
$\mu \in \{1, 2\} \leftarrow \mathcal{A}()$
$\bar{\mu} \leftarrow 3 - \mu$
$\vec{g} \xleftarrow{\$} \mathbb{F}_{2^k}^{\bar{v}}$
if $\mathsf{rank}_2(\vec{g}) < k$ **then**
  return $\perp$
With probability $1 - \prod_{i=0}^{t-1}(1 - 2^{i-\bar{v}})$, **return** $\perp$
return $\mathcal{A}(\mathsf{sk}_\mu, \vec{g})$

We use the reformulation of the indistinguishability-based FSS security definition given in Definition 8 and show that for every efficient adversary $\mathcal{A}$, |Game 27 − Game 28| is negligible. In these games, the adversary inputs two alternatives for $\vec{\alpha}$ and $\vec{\beta}$ that define the point function and has to distinguish which of the pairs was used to generate the keys that it receives.

Game 27:

$(\alpha_1^0, \beta_1^0), \ldots, (\alpha_t^0, \beta_t^0),$
$(\alpha_1^1, \beta_1^1), \ldots, (\alpha_t^1, \beta_t^1), \sigma, \mu \leftarrow \mathcal{A}()$
$c \leftarrow 0$
$\mathsf{sk}_1, \mathsf{sk}_2, \leftarrow \mathsf{Gen}((\alpha_1^c, \beta_1^c), \ldots, (\alpha_t^c, \beta_t^c))$
return $\mathcal{A}(\mathsf{sk}_\mu, \sigma)$

Game 28:

$(\alpha_1^0, \beta_1^0), \ldots, (\alpha_t^0, \beta_t^0),$
$(\alpha_1^1, \beta_1^1), \ldots, (\alpha_t^1, \beta_t^1), \sigma, \mu \leftarrow \mathcal{A}()$
$c \leftarrow 1$
$\mathsf{sk}_1, \mathsf{sk}_2, \leftarrow \mathsf{Gen}((\alpha_1^c, \beta_1^c), \ldots, (\alpha_t^c, \beta_t^c))$
return $\mathcal{A}(\mathsf{sk}_\mu, \sigma)$

Figure 7. Games defining security of FSS key generation with Gen

Theorem 6. *Suppose that f, f' are $\varepsilon_{\mathsf{PRG}}$-PRGs. Let $v > t + \kappa$, $\bar{v} > k + t + \kappa + 2$. Denote*

$$\varepsilon_{ROR} := t \cdot n \cdot \varepsilon_{\mathsf{PRG}} + 2^{-k-\kappa} + 3 \cdot 2^{-\kappa-2}$$

then SLAMP-FSS is $(1, 2\varepsilon_{ROR})$-secure.

That is, for all $\varepsilon_{\mathsf{PRG}}$-pseudo random generators f, f', if $\mathcal{P}_i$ is corrupted, $i \in \{1, 2\}$, and $\mathcal{P}_{3-i}$ is not corrupted, then for all PPT algorithms $\mathcal{A}$

$$\Pr[\textit{Game 27}(\mathcal{A}) = 1] - \Pr[\textit{Game 28}(\mathcal{A}) = 1] \leq 2\varepsilon_{ROR}$$

Proof. We get from Theorem 4 that SLAMPR-FSS is $t \cdot n \cdot \varepsilon_{\mathsf{PRG}} + 2^{-k-\kappa}$-close to random. Therefore, from Theorem 5 we get that SLAMP-FSS is $2(t \cdot n \cdot \varepsilon_{\mathsf{PRG}} + 2^{-k-\kappa} + 3 \cdot 2^{-\kappa-2})$-secure. □

7 Comparison to prior works

We have implemented a prototype[2] of SLAMP-FSS in the Rust programming language. Our prototype instantiates SLAMP-FSS with the concrete parameters of $n = 20$, $t = 22$, $k, v = 64$ and $\bar{v} = 128$. We implement the PRG functions f and f' using fixed-key AES encryption in the same manner as [BGH+25].

[2]The prototype code is published on GitHub https://github.com/jrmngndr/slamp-fss .

Table 1. The average execution time of primitive functions and the number of calls to them in SLAMP-FSS

Primitive	Cost[3]	Number of calls in Gen	Number of calls in FullEval
PRG f	30.1 ns	$\leq 2(n-2)t$	2^{n-1}
PRG f'	32.7 ns	$2t$	2^{n-1}
Solve $Ax = \vec{b}$, $A \in (\mathbb{F}_2)^{tv}$, $\vec{b} \in (\mathbb{F}_{2^k})^t$	39.4 μs	n	0
Solve $Ax = \vec{b}$, $A \in (\mathbb{F}_2)^{t\bar{v}}$, $\vec{b} \in (\mathbb{F}_{2^k})^t$	84.5 μs	1	0
Find $\mathsf{rank}_2(A)$, $A \in (\mathbb{F}_2)^{tv}$, [1]	17.7 μs	$n-1$	0
Find $\mathsf{rank}_2(A)$, $A \in (\mathbb{F}_2)^{t\bar{v}}$, [1]	35.6 μs	3	0
Find $\mathsf{rank}_2(A)$, $A \in (\mathbb{F}_2)^{t(\bar{v}+k)}$, [1]	52.5 μs	1	0
Find $\mathsf{rank}_2(A)$, $A \in (\mathbb{F}_2)^{\bar{v}^2}$, [2]	69.9 μs	3.8 [4]	0
$\mathbb{F}_{2^k}$ multiplication	32.2 ns	$\leq 4(n-1)t$	2^{n-1}
$\mathbb{F}_{2^{\bar{v}}}$ multiplication	27.8 ns	$4t$	2^{n-1}
$\langle \vec{e}, \vec{f} \rangle$, where $\vec{e} \in (\mathbb{F}_2)^v$,$\vec{f} \in (\mathbb{F}_{2^k})^v$	37.3 ps	$2(n-1)t$	2^{n-1}
$\langle \vec{e}, \vec{f} \rangle$, where $\vec{e} \in (\mathbb{F}_2)^{\bar{v}}$,$\vec{f} \in (\mathbb{F}_{2^{\bar{v}}})^{\bar{v}}$	37.2 ps	$2t$	2^n

[1] These can be omitted, with the result being a negligible probability of incorrect output.
[2] This can be omitted if the output of f' is set to be $(\mathbb{F}_2)^{\bar{v}+\kappa} \times \mathbb{F}_{2^{\bar{v}}}$.
[3] The cost is the average execution time. Measured with parameters $t = 22$, $k = 64$, $v = 64$, $\bar{v} = 128$ on an Intel Xeon E5-2640 v3 2,6 GHz CPU.
[4] Expected number of calls, by Corollary 1

In Table 1 we have measured the amortised execution time of the primitive function calls used by SLAMP-FSS. The fixed-key implementation of the PRG functions is considerably faster than implementing them using AES in counter mode and it enables evaluating the PRG functions of a single layer in a single batch. Batch evaluation of AES encryption benefits from specialised CPU instructions such as AES-NI on x86 instruction set CPUs. See [WYG+17] for a discussion on the implementation of PRGs in FSS schemes.

Another important primitive operation in SLAMP-FSS is multiplication in $GF(2^{\bar{v}})$ and $GF(2^k)$. The concrete instantiation of $GF(2^{128})$ is faster than $GF(2^{64})$ since the former is used in AES in Galois/Counter Mode (GCM) and an optimised library implementation exists[3].

The concrete values for t and n (in this paragraph, $N = 2^n$) depend heavily on the application. For example, in PCG-s the DMPF is used to share the error vector for the LPN assumption. For some concrete numbers, the PCG solution Dory [GLH+25] lists some parameters for LPN assumption to achieve 128-bit security of the PCG. For example, $N = 293,601,280$ and $t = 1,120$ or $N = 51,478,528$ and $t = 1,571$, these cases are close to $N = 2^{27}$. DMPF construction in [BGH+25] reports the performance of their scheme for $t < 2^{14}$ for $N = 2^{27}$ and $N = 2^{60}$. Figure 8 shows the expected runningtime of SLAMP-FSS using the numbers from Table 1 for $n \leq 60$. The values for key generation are evaluated using the upper bounds for the number of calls to each operation. Note that the evaluation time is independent of the number of nonzero points t.

The theoretical complexity comparisons to related work is given in Table 2. We denote the number of nonzero points as t, the size of the field or group for the DMPF as k and n is the length of the function input, hence, FullEval evaluates on 2^n inputs. Note that we use κ as the security parameter in this paper, but v as the length of the input to the PRG. Often, the PRG input and the security parameter are the same, hence we use v as the

[3] https://docs.rs/polyval/latest/polyval/

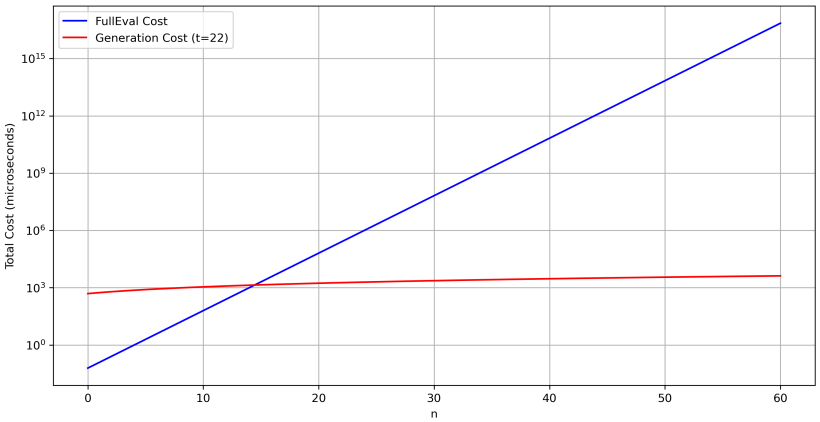


Figure 8. Expected SLAMP-FSS runningtime based on Table 1

Table 2. Multi-point function secret sharing scheme comparison

Scheme	Key size	Number of PRG calls in Gen	Number of PRG calls in Eval	Number of PRG calls in FullEval
Simple [BGI16]	$O(t(vn+k))$	$O(tn)$	tn	$2t2^n$
[BCGI18], batch codes	$O(t^{1+\epsilon}(v\lceil n - \log t\rceil + k))$	$O(n)$	-	$O(2^n)$
[BCGI18], RSD [1]	$O(t(vn+k))$	$O(tn)$	n	$(1+\lceil\frac{k}{v+2}\rceil)2^n$
[SGRR19], PBC [2]	$O(t(vn+k))$	$O(tn)$	-	$3\cdot 2^n$
[BGH+25], OKVS	$n\cdot\zeta+\eta$ [3]	$2t(n+1)$	n	$2\cdot 2^n$
[BGH+25], big-state	$t(\kappa+2t)n+tk$	$2tn$	$n+1$	$2\cdot 2^n$
SLAMP-FSS	$\approx vnk$	$2(n-1)t$	n	2^n

[1] The algorithm requires a stronger assumption.
[2] The algorithm has a small $\approx 2^{-40}$ error probability and is an instantiation of a weaker primitive called a *known-point FSS.*
[3] Here $\zeta = OKVS.CodeSize$ and $\eta = OKVS_{conv}.CodeSize$. OKVS size scales linearly with the number of encoded elements in OKVS. Note that the code size and decoding time are trade-offs in OKVS schemes, longer keys may reduce the OKVS evaluation time while increasing the communication required in the DMPF scheme [BGH+25].

security parameter for the schemes in the table. The partitioning construction in [BCGI18] results in a distribution of the secret points with a much lower entropy, however, using the stronger Regular Syndrome Decoding (RSD) assumption, one can still build the necessary applications securely using this construction. In practice, this means that a larger value of t is required, for example, as in [BBMS22], where the parameter t is larger by the statistical security parameter.

We note that we mainly compare ourselves to [BGH+25], notably our number of PRG calls is comparable to theirs. Indeed, the outputs of their PRGs are slightly more than twice as long as the input. In our PRG calls, half of the calls have an output that is twice the length of the input and half of the calls have an output that is four times as long as the input. However, in experiments in Table 1 we did not notice a significant time difference for the two PRGs we use, thus, we estimate that the average cost of a PRG-call is similar. The current comparison does not include calls to the OKVS.Encode and OKVS.Decode for [BGH+25] or the field operations for SLAMP-FSS. The full evaluation contains $2\cdot 2^n$ calls to OKVS.Decode, and in the construction of [BPSY23], each call contains two calls to

Table 3. The number of PRG calls needed for some common parameter set

Scheme	PRG calls in Gen			PRG calls in FullEval		
	$t = 22$ $n = 20$	$t = 1571$ $n = 27$	$t = 2^{14}$ $n = 60$	$t = 22$ $n = 20$	$t = 1571$ $n = 27$	$t = 2^{14}$ $n = 60$
[BGH+25], OKVS	924	87,976	1,998,848	2^{21}	2^{28}	2^{61}
[BGH+25], big-state	880	84,834	1,966,080	2^{21}	2^{28}	2^{61}
SLAMP-FSS	836	81,692	1,933,312	2^{20}	2^{27}	2^{60}

Table 4. Average execution time comparison of multi-point function secret sharing schemes

Scheme	Gen	EvalRand	FullEval
[BGH+25], big-state	396.0 μs	2.82 μs	102 ms
SLAMP-FSS	1840 μs	9.55 μs	1480 ms

[1] Measurements performed with parameters $n = 20$, $t = 22$, $k = 64$, $v = 64$, $\bar{v} = 128$ on an Intel Xeon E5-2640 v3 2,6 GHz CPU.

a hash function, which are assumed to have non-negligible costs. We see that SLAMP-FSS has a similar number of PRG calls in Gen and EvalRand, to [BGH+25], and that the key-size is comparable. However, the number of PRG calls in SLAMP-FSS for the FullEval is half of [BGH+25]. Making SLAMP-FSS at the very least, an interesting counterpart to [BGH+25]. The concrete number of PRG calls for some concrete set of parameters for SLAMP-FSS and [BGH+25] are given in Table 3.

Comparison of average runtime of key generation, single point evaluation and full domain evaluation is given in Table 4. We compare the concrete instantiation of SLAMP-FSS in our prototype to the implementation of the scheme from [BGH+25] which performs best with the same-sized domain and number of nonzero points. The two schemes were measured on the same hardware, the reported execution times are the averages across a large number of repetitions.

The most significant benchmark for applications of MPFSS is the full domain evaluation time. Despite improving upon the number of PRG calls in FullEval, our implementation is considerably slower than [BGH+25] big-state (1480 ms compared to 102 ms). Two optimisations that were included in the implementation of [BGH+25] but not in our prototype are likely to explain this difference. Including these optimisations in future implementations of SLAMP-FSS would likely ensure that the implementation of SLAMP-FSS outperforms the implementation included in [BGH+25]. Note that, based on Table 1, we would expect our scheme to have full evaluation time of about 65 ms which suggests that there is room for improvement.

Firstly, we note that [BGH+25] mention an improvement where they use a PRG with a length-quadrupling output in the last round, which allows them to reduce the total calls to PRGs by 25%. However, this brings their PRG-count down to $1.5 \cdot 2^n$, which is still significantly higher than our 2^n. On the other hand, if we similarly used a PRG with a larger output in the last round, this would allow us to choose the output to be $(\mathbb{F}_2)^{\bar{v}+\kappa} \times \mathbb{F}_{2^{\bar{v}}}$. This change would mean that a solution to $A\vec{d}_{n-1} = \vec{b}$ would be full-rank with an overwhelming probability, which would allow us to omit that particular check, which is a costly part of our Gen algorithm.

Secondly, the implementation of [BGH+25] uses the early termination optimisation from [BGI16] to reduce the last layers of the DPF tree. In their implementation, the tree is reduced by 7 levels, which improves the performance of FullEval by a factor of 128. The same optimisation is compatible with SLAMP-FSS as well, but has not been implemented in the prototype. Because our prototype implementation does not include these optimisations, the execution times between SLAMP-FSS and [BGH+25] are not directly comparable, with equivalent implementations, SLAMP-FSS would compare more

favourably.

The implementation RevCuckoo DMPF full evaluation (expand) in [ARR25] reports execution times from 1 to 30 ms. Their generation times are not comparable, as they focus on distributed generation. Overall, comparisons with [ARR25] and [BGH+25] indicate that our scheme has competitive performance, however, the prototype implementation could benefit from several known optimisations.

8 Conclusions

We presented new multi-point function secret sharing schemes SLAMP-FSS and SLAMPR-FSS and proved that they are complete and secure with a generation function that succeeds with overwhelming probability. We expect that SLAMP-FSS would be reasonable for cases where t is not too large, say, $t \leq 32$ or $t \leq 64$. We have thus presented a new distributed multi-point function with performance results that are on par with other current schemes.

We separate the scheme into a random-DMPF (SLAMPR-FSS) and a conversion to get the DMPF with fixed outputs, where, in some applications, the former may be sufficient. The biggest advantage of SLAMP-FSS over its competitors is that the number of PRG evaluations needed for full evaluation is much smaller while keeping the other efficiency metrics competitive with the other schemes. Moreover, SLAMP-FSS is built from simple mathematical constructions and is conceptually simple. It seems likely that it can be combined with other techniques, like packing the key to reduce its size [BGI16] or a distributed setup [DS17], in the future. In addition, our implementation can be enhanced with the optimisations considered in [BGH+25].

Acknowledgments

The research was supported by the European Union and Estonian Research Council grants PSG1228, PRG2531 and TEM-TA119 and Office of Naval Research (ONR) under grant number N62909-23-1-2046.

References

[ACLS18] Sebastian Angel, Hao Chen, Kim Laine, and Srinath T. V. Setty. PIR with compressed queries and amortized query processing. In *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*, pages 962–979. IEEE Computer Society, 2018. doi:10.1109/SP.2018.00062.

[AR91] Howard Anton and Chris Rorres. *Elementary linear algebra: applications version.* John Wiley & Sons, sixth edition, 1991.

[ARR25] Amit Agarwal, Srinivasan Raghuraman, and Peter Rindal. Fully distributed multi-point functions for PCGs and beyond. Cryptology ePrint Archive, Paper 2025/2294, 2025. URL: https://eprint.iacr.org/2025/2294.

[BB89] Judit Bar-Ilan and Donald Beaver. Non-cryptographic fault-tolerant computing in constant number of rounds of interaction. In Piotr Rudnicki, editor, *Proceedings of the Eighth Annual ACM Symposium on Principles of Distributed Computing, Edmonton, Alberta, Canada, August 14-16, 1989*, pages 201–209. ACM, 1989. doi:10.1145/72981.72995.

[BBC+21] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. Lightweight techniques for private heavy hitters. In *42nd IEEE Symposium*

on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021, pages 762–776. IEEE, 2021. doi:10.1109/SP40001.2021.00048.

[BBC+24] Maxime Bombar, Dung Bui, Geoffroy Couteau, Alain Couvreur, Clément Ducros, and Sacha Servan-Schreiber. FOLEAGE: $\mathbb {F}_{\scriptstyle 4}$ole-based multi-party computation for boolean circuits. In Kai-Min Chung and Yu Sasaki, editors, *Advances in Cryptology - ASIACRYPT 2024 - 30th International Conference on the Theory and Application of Cryptology and Information Security, Kolkata, India, December 9-13, 2024, Proceedings, Part VI*, volume 15489 of *Lecture Notes in Computer Science*, pages 69–101. Springer, 2024. doi:10.1007/978-981-96-0938-3_3.

[BBMS22] Carsten Baum, Lennart Braun, Alexander Munch-Hansen, and Peter Scholl. Moz$\mathbb{Z}_{2^k}$arella: Efficient vector-OLE and zero-knowledge proofs over $\mathbb{Z}_{2^k}$. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15-18, 2022, Proceedings, Part IV*, volume 13510 of *Lecture Notes in Computer Science*, pages 329–358. Springer, 2022. doi:10.1007/978-3-031-15985-5_12.

[BCG+19] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part III*, volume 11694 of *Lecture Notes in Computer Science*, pages 489–518, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Cham, Switzerland. doi:10.1007/978-3-030-26954-8_16.

[BCG+20] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators from ring-LPN. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part II*, volume 12171 of *Lecture Notes in Computer Science*, pages 387–416, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Cham, Switzerland. doi:10.1007/978-3-030-56880-1_14.

[BCGI18] Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. Compressing vector OLE. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018: 25th Conference on Computer and Communications Security*, pages 896–912, Toronto, ON, Canada, October 15–19, 2018. ACM Press. doi:10.1145/3243734.3243868.

[Bea91] Donald Beaver. Efficient multiparty protocols using circuit randomization. In Joan Feigenbaum, editor, *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*, volume 576 of *Lecture Notes in Computer Science*, pages 420–432. Springer, 1991. doi:10.1007/3-540-46766-1_34.

[BGH+25] 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 44–44, Los Alamitos, CA, USA, May 2025. IEEE Computer Society. URL: https://doi.ieeecomputersociety.org/10.1109/SP61157.2025.00044, doi:10.1109/SP61157.2025.00044.

[BGI15] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part II*, volume 9057 of *Lecture Notes in Computer Science*,

pages 337–367, Sofia, Bulgaria, April 26–30, 2015. Springer Berlin Heidelberg, Germany. doi:10.1007/978-3-662-46803-6_12.

[BGI16] 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, *ACM CCS 2016: 23rd Conference on Computer and Communications Security*, pages 1292–1303, Vienna, Austria, October 24–28, 2016. ACM Press. doi:10.1145/2976749.2978429.

[BGIK22] Elette Boyle, Niv Gilboa, Yuval Ishai, and Victor I. Kolobov. Programmable distributed point functions. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15-18, 2022, Proceedings, Part IV*, volume 13510 of *Lecture Notes in Computer Science*, pages 121–151. Springer, 2022. doi:10.1007/978-3-031-15985-5_5.

[BKKO20] Paul Bunn, Jonathan Katz, Eyal Kushilevitz, and Rafail Ostrovsky. Efficient 3-party distributed ORAM. In Clemente Galdi and Vladimir Kolesnikov, editors, *Security and Cryptography for Networks - 12th International Conference, SCN 2020, Amalfi, Italy, September 14-16, 2020, Proceedings*, volume 12238 of *Lecture Notes in Computer Science*, pages 215–232. Springer, 2020. doi:10.1007/978-3-030-57990-6_11.

[BPSY23] 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 Joseph A. Calandrino and Carmela Troncoso, editors, *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023*, pages 301–318. USENIX Association, 2023. URL: https://www.usenix.org/conference/usenixsecurity23/presentation/bienstock.

[CBM15] 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, SP 2015, San Jose, CA, USA, May 17-21, 2015*, pages 321–338. IEEE Computer Society, 2015. doi:10.1109/SP.2015.27.

[CPS08] Jean-Sébastien Coron, Jacques Patarin, and Yannick Seurin. The random oracle model and the ideal cipher model are equivalent. In David Wagner, editor, *Advances in Cryptology – CRYPTO 2008*, volume 5157 of *Lecture Notes in Computer Science*, pages 1–20, Santa Barbara, CA, USA, August 17–21, 2008. Springer Berlin Heidelberg, Germany. doi:10.1007/978-3-540-85174-5_1.

[dCP22] Leo de Castro and Antigoni Polychroniadou. Lightweight, maliciously secure verifiable function secret sharing. In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology - EUROCRYPT 2022 - 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30 - June 3, 2022, Proceedings, Part I*, volume 13275 of *Lecture Notes in Computer Science*, pages 150–179. Springer, 2022. doi:10.1007/978-3-031-06944-4_6.

[DFL+20] Emma Dauterman, Eric Feng, Ellen Luo, Raluca Ada Popa, and Ion Stoica. DORY: an encrypted search system with distributed trust. In *14th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2020,*

Virtual Event, November 4-6, 2020, pages 1101–1119. USENIX Association, 2020. URL: https://www.usenix.org/conference/osdi20/presentation/dauterman-dory.

[DIL+20] Samuel Dittmer, Yuval Ishai, Steve Lu, Rafail Ostrovsky, Mohamed Elsabagh, Nikolaos Kiourtis, Brian Schulte, and Angelos Stavrou. Function secret sharing for PSI-CA: with applications to private contact tracing. *IACR Cryptol. ePrint Arch.*, page 1599, 2020. URL: https://eprint.iacr.org/2020/1599.

[DRPS22] Emma Dauterman, Mayank Rathee, Raluca Ada Popa, and Ion Stoica. Waldo: A private time-series database from function secret sharing. In *43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, May 22-26, 2022*, pages 2450–2468. IEEE, 2022. doi:10.1109/SP46214.2022.9833611.

[DS17] Jack Doerner and Abhi Shelat. Scaling ORAM for secure computation. In Bhavani Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 523–535. ACM, 2017. doi:10.1145/3133956.3133967.

[ECZB21] Saba Eskandarian, Henry Corrigan-Gibbs, Matei Zaharia, and Dan Boneh. Express: Lowering the cost of metadata-hiding communication with cryptographic privacy. In Michael D. Bailey and Rachel Greenstadt, editors, *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, pages 1775–1792. USENIX Association, 2021. URL: https://www.usenix.org/conference/usenixsecurity21/presentation/eskandarian.

[FLO19] Brett Hemenway Falk, Steve Lu, and Rafail Ostrovsky. DURASIFT: A robust, decentralized, encrypted database supporting private searches with complex policy controls. In Lorenzo Cavallaro, Johannes Kinder, and Josep Domingo-Ferrer, editors, *Proceedings of the 18th ACM Workshop on Privacy in the Electronic Society, WPES@CCS 2019, London, UK, November 11, 2019*, pages 26–36. ACM, 2019. doi:10.1145/3338498.3358651.

[GGM24] Gayathri Garimella, Benjamin Goff, and Peihan Miao. Computation efficient structure-aware PSI from incremental function secret sharing. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology - CRYPTO 2024 - 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2024, Proceedings, Part VIII*, volume 14927 of *Lecture Notes in Computer Science*, pages 309–345. Springer, 2024. doi:10.1007/978-3-031-68397-8_10.

[GI14] Niv Gilboa and Yuval Ishai. Distributed point functions and their applications. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 640–658, Copenhagen, Denmark, May 11–15, 2014. Springer Berlin Heidelberg, Germany. doi:10.1007/978-3-642-55220-5_35.

[GJM+24] Kanav Gupta, Neha Jawalkar, Ananta Mukherjee, Nishanth Chandran, Divya Gupta, Ashish Panwar, and Rahul Sharma. SIGMA: secure GPT inference with function secret sharing. *Proc. Priv. Enhancing Technol.*, 2024(4):61–79, 2024. URL: https://doi.org/10.56553/popets-2024-0107, doi:10.56553/POPETS-2024-0107.

[GKWY20] Chun Guo, Jonathan Katz, Xiao Wang, and Yu Yu. Efficient and secure multiparty computation from fixed-key block ciphers. In *2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020*, pages 825–841. IEEE, 2020. doi:10.1109/SP40000.2020.00016.

[GLH+25] Xiaojie Guo, Hanlin Liu, Zhicong Huang, Hongrui Cui, Wenhao Zhang, Cheng Hong, Xiao Wang, Kang Yang, and Yu Yu. Dory: Streaming PCG with small memory. *IACR Cryptol. ePrint Arch.*, 2025:1660, 2025. URL: https://eprint.iacr.org/2025/1660.

[Gol01] Oded Goldreich. *Foundations of Cryptography: Basic Tools*. Cambridge university press, 2001.

[JZZR24] Keyu Ji, Bingsheng Zhang, Hong-Sheng Zhou, and Kui Ren. Distributed point function with constraints, revisited. *IACR Cryptol. ePrint Arch.*, page 937, 2024. URL: https://eprint.iacr.org/2024/937.

[KDK+24] Stanislav Kruglik, Son Hoang Dau, Han Mao Kiah, Huaxiong Wang, and Liang Feng Zhang. Verifiable information-theoretic function secret sharing. *IACR Cryptol. ePrint Arch.*, page 453, 2024. URL: https://eprint.iacr.org/2024/453.

[KKE+26] Erki Külaots, Toomas Krips, Hendrik Eerikson, Darya Pisetskaya, and Pille Pullonen-Raudvere. SLAMP-FSS: Two-party multi-point function secret sharing from simple linear algebra. *IACR Communications in Cryptology*, 3(1), 2026. doi:10.62056/avommpxqi.

[KKEPR24] Erki Külaots, Toomas Krips, Hendrik Eerikson, and Pille Pullonen-Raudvere. SLAMP-FSS: Two-party multi-point function secret sharing from simple linear algebra. Cryptology ePrint Archive, Paper 2024/1394, 2024-09-13, 2024. URL: https://eprint.iacr.org/archive/2024/1394/20240913:055602.

[KL08] Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography*. Chapman and hall/CRC, 2008.

[KL23] Stephan Krenn and Thomas Lorünser. *An Introduction to Secret Sharing: A Systematic Overview and Guide for Protocol Selection*. Springer Nature, 2023.

[KPM24] Chandan Kumar, Sikhar Patranabis, and Debdeep Mukhopadhyay. Compact key function secret sharing with non-linear decoder. *IACR Commun. Cryptol.*, 1(2):8, 2024. URL: https://doi.org/10.62056/a3c3c3w9p, doi:10.62056/A3C3C3W9P.

[MP13] Gary L Mullen and Daniel Panario. *Handbook of finite fields*, volume 17. CRC press Boca Raton, 2013.

[NSD22] Zachary Newman, Sacha Servan-Schreiber, and Srinivas Devadas. Spectrum: High-bandwidth anonymous broadcast. In Amar Phanishayee and Vyas Sekar, editors, *19th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2022, Renton, WA, USA, April 4-6, 2022*, pages 229–248. USENIX Association, 2022. URL: https://www.usenix.org/conference/nsdi22/presentation/newman.

[SGRR19] Phillipp Schoppmann, Adrià Gascón, Leonie Reichert, and Mariana Raykova. Distributed vector-ole: Improved constructions and implementation. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz,

editors, *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, pages 1055–1072. ACM, 2019. doi:10.1145/3319535.3363228.

[Sha79] Adi Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, November 1979. doi:10.1145/359168.359176.

[SVLH23] Kyle Storrier, Adithya Vadapalli, Allan Lyons, and Ryan Henry. Grotto: Screaming fast (2+1)-PC or $\mathcal{F}$2n via (2, 2)-dpfs. In Weizhi Meng, Christian Damsgaard Jensen, Cas Cremers, and Engin Kirda, editors, *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, CCS 2023, Copenhagen, Denmark, November 26-30, 2023*, pages 2143–2157. ACM, 2023. doi:10.1145/3576915.3623147.

[Wag22] Sameer Wagh. Pika: Secure computation using function secret sharing over rings. *Proc. Priv. Enhancing Technol.*, 2022(4):351–377, 2022. URL: https://doi.org/10.56553/popets-2022-0113, doi:10.56553/POPETS-2022-0113.

[WYG+17] Frank Wang, Catherine Yun, Shafi Goldwasser, Vinod Vaikuntanathan, and Matei Zaharia. Splinter: practical private queries on public data. In *Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation*, NSDI'17, page 299–313, USA, 2017. USENIX Association.

[Zha23] Shuqing Zhang. Efficient VOLE based multi-party PSI with lower communication cost. *IACR Cryptol. ePrint Arch.*, page 1690, 2023. URL: https://eprint.iacr.org/2023/1690.