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

Erki Külaots[1], Toomas Krips[1][0000−0003−0981−3553], Hendrik Eerikson[1,2][0009−0009−9298−074X], and Pille Pullonen-Raudvere[2][0000−0002−3255−7001]

[1] University of Tartu, Estonia
{erki.kulaots, toomas.krips}@ut.ee
[2] Cybernetica AS, Estonia
{hendrik.eerikson, pille.pullonen-raudvere}@cyber.ee

Abstract. Multiparty computation (MPC) is an important field of cryptography that deals with protecting the privacy of data, while allowing to do computation on that data. A key part of MPC is the parties involved having correlated randomness that they can use to make the computation or the communication between themselves more efficient, while still preserving the privacy of the data. Examples of these correlations include random oblivious transfer (OT) correlations, oblivious linear-function evaluation (OLE) correlations, multiplication triples (also known as Beaver triples) and one-time truth tables. Multi-point function secret sharing (FSS) has been shown to be a great building block for pseudorandom correlation generators. The main question is how to construct fast and efficient multi-point FSS schemes. Here we propose a natural generalization of the scheme of Boyle et al [BGI16] using a tree structure, a pseudorandom generator and systems of linear equations. Our schemes SLAMP-FSS and SLAMPR-FSS are more efficient in the evaluation phase than other previously proposed multi-point FSS schemes while being also more flexible and being similar in other efficiency parameters.

Keywords: Function secret sharing · secure computation · distributed point functions.

1 Introduction

A two party function secret sharing scheme allows the parties to jointly evaluate a parameterized function on common inputs such that the parties separately do not learn anything about the output or the parameters nor do they need to communicate during such evaluation.

Point functions are functions that evaluate to a predetermined value b at the predetermined special point a and evaluate to 0 everywhere else. Multi-point functions have some constant number of such special points.

Multi-point function secret sharing is part of the broader topic of secure multiparty computation (MPC), which is prevalent in all areas of life, where private

information needs to be processed. If private information has to be processed and no single party can be trusted with the whole data, then the computation can be shared between multiple parties such that no single party learns any significant part of the private data. Examples of this include: medical information for research, handling private financial data or keys for cryptocurrencies [IEE].

Function secret sharing (FSS) was proposed to be useful in the setting of querying and updating distributed databases (private information retrieval – PIR) [BGI15] and it has also been used to construct ORAM [DS17]. In addition to that, multi-point FSS has seen some success in the constructions of other primitives such as vector oblivious linear-function evaluation (VOLE) [BCGI18] and random oblivious transfer [BCG$^+$19] The common denominator is that the FSS schemes can be used to construct efficient pseudorandom correlation generators (PCGs) and then the PCG can provide correlated randomness to the parties involved. Correlated randomness is a precious resource in cryptographic applications. For example, identical random strings distributed among the parties can be used for perfectly secure encryption by using one-time pad encryption. Other examples of more useful types of random correlation in respect to MPC are random oblivious transfer (OT) correlations, oblivious linear-function correlations (OLE), multiplication triples (also known as Beaver triples [Bea91]) and one-time truth tables [BCGI18,BCG$^+$19].

We propose two new schemes for multipoint FSS. We call our main scheme Simple Linear Algebra MultiPoint Function Secret Sharing, or SLAMP-FSS for short. Previous schemes rely on many pseudorandom generator (PRG) evaluations that are by far the most expensive part of the protocol and a large number of PRG calls are needed in the protocol. Therefore, our goal was to bring down the number of required PRG calls in the evaluation step. SLAMP-FSS has similar key size and time complexity to other multi-point function secret sharing schemes and its advantages are a simple mathematical constructions and efficiency of evaluating the shared function at some small constant number of points.

For constructing PCGs, commonly, t (where t is in the approximate order of magnitude of $2^5 - 2^{10}$) applications of a distributed point functions are needed, and often it is necessary to evaluate the function at all points. We refer to evaluating the function on all points as *full evaluation*. So far, the most common approach 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 tradeoffs.

We propose a natural generalization of the [BGI16] for t points. The scheme uses simple linear algebra and is conceptually simple. Like [BGI16], our construction will be based on a binary tree with n levels, with the leaves corresponding naturally with the domain of the DPF. The tree has t leaves where the function will evaluate to non-zero, we will think of these leaves and nodes on the paths leading to them from the root as alive nodes, the other nodes we will consider to

be dead nodes. Evaluating the DPF at a point x corresponds to travelling down to the leaf corresponding to x. The parties will have a secret-shared value with the invariant that the value of this is zero if and only if we are currently at a dead node. Thus it is important to construct the DPF in such a way that if a node is dead, its children will automatically be dead. For alive nodes, at each level of the tree, there can be at most t of them. They will have twice the amount of children some of which will be alive and some will be dead. We will use simple linear algebra to achieve that this property will be satisfied at all levels.

We also propose a variation of the scheme that is more effective for full evaluation, but outputs random values at the predetermined points. We call this scheme Simple Linear Algebra MultiPoint Random Function Secret Sharing, or SLAMPR-FSS for short. The scheme differs from SLAMP-FSS only slightly but leads to twice as effective full evaluation.

We note that unlike batch-code based attempts, which can only be used for full evaluation, our schemes can be used to evaluate on any number of points. For full evaluation, they are up to several times more efficient than previous works while having a key size that is approximately similar.

We currently leave the secure instantiation of the setup as a future work. However, note that while at first glance there may seem to be two large roadblocks that are hard to instantiate securely, namely, evaluating a PRG and solving a system of t linear equations, it is plausible that these problems can be solved much more efficiently than the naive approach. First, note that the PRG is not evaluated on shared values but on shares. Hence, it is possible that a solution like the setup algorithm by Doerner and shelat [DS17] could also be adapted to our case to evaluate all PRG-s locally. Secondly, if we wish to solve a system of linear equations $[A]\boldsymbol{x} = [\boldsymbol{b}]$ over a large field where 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 $\boldsymbol{x}$, in which case solving the rest of the system is equivalent to solving a related system $[A']\boldsymbol{x'} = [\boldsymbol{b'}]$ where A' is with an overwhelming probability a random square matrix, and $[A']$ and $[\boldsymbol{b'}]$ can be computed with no communication (i.e quickly). This can be solved by computing $[{A'}^{-1}]$ and computing $[{A'}^{-1}][\boldsymbol{b'}]$. Computing $[{A'}^{-1}]$ can in turn be solved by a two secure matrix multiplications [BB89].

The outline of this work is as follows. In Section 2, we give a short description of prior work. In Section 3, the preliminary knowledge that the reader needs for understanding the paper is given. In Section 4, algorithms are introduced and explained. In Section 5, the properties of SLAMP-FSS are presented and rigorously proved. In Section 6, we compare SLAMP-FSS and SLAMPR-FSS to prior works. Finally, Section 7 concludes this paper. In Appendix A, we give the missing proofs and games that did not fit into the main body of the paper. In Appendix B, we give the modified version SLAMPR-FSS and show that it satisfies the necessary properties. In Appendix C, we calculate the efficiency of our schemes.

2 Prior work

Our work expands on the work of Boyle et al. [BGI16] in which the construction of a single-point DPF is proposed. This construction requires n evaluations of the PRG to evaluate a single point and $2 \cdot 2^n$ evaluations of the PRG for full evaluation.

Recently, Guo et al. [GYW+23] improved on the construction of the original point function, making the full evaluation 1.33 times faster, making it require $1.5 \cdot 2^n$ evaluations of the PRG instead of $2 \cdot 2^n$ evaluations. Note that this still requires to run the protocol t times to obtain the multipoint DPF, thus the total cost would be $1.5t \cdot 2^n$.

In [BCGI18] two approaches were proposed to obtain the goal of t point functions. The first was to partition the domain of the function into t smaller sections and to apply the original construction to each section separately. This construction 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 has meant, however that a larger value of t is used for applications, for example as in [BBMS22] where the t used is larger by the statistical security parameter.

The second proposal in [BCGI18] was to use combinatorial objects called batch codes. These do give a full evaluation in $O(2^n)$ evaluations of the PRG and have a setup cost of only $O(n)$, but the concrete estimation requires the coefficient in $O(2^n)$ to be quite high. It also allows to only apply the full evaluation.

In [ACLS18,SGRR19], probabilistic batch codes were used. These do achieve the full evaluation computation of $3 \cdot 2^n$ PRG calls. However, they in fact implement a weaker primitive called Known-Index FSS, where one of the parties learns the position of the secret point. Although this is sufficient for some applications, it is not sufficient for all of them. Also, there is a small ($\approx 2^{-40}$) error probability and the construction again can be only used for full evaluation.

3 Preliminaries

This section is meant as a guide to return to, if something in later sections becomes confusing. However, it is recommended to read through sections about secret sharing, functional secret sharing, distributed point functions and multi-point functions as these topics are less well known. Most of the definitions given are classical with some minor changes to fit more tightly with this research paper.

3.1 Notation

Let us list some important notation. For sets X and Y, we denote $\mathcal{F}(X \to Y)$ as the family of all functions $\varphi : X \to Y$. For $n \in \mathbb{N}$, we denote $\{0,1\}^n$ as bit strings with length n and $\{0,1\}^*$ as the set of all bit strings. For bit strings $x \in \{0,1\}^*$ and integers $i \in \mathbb{N}$, we denote x^i as the i-th bit of x. For bit strings $a, b \in \{0,1\}^*$,

we denote $a||b$ as the concatenation of a and b. For bits $a^1, a^2 \in \{0,1\}$, we denote a^1a^2 as the concatenation of a^1 and a^2. For bit strings $a, x \in \{0,1\}^n$, we denote $x^1x^2 \dots x^i = a^1a^2 \dots a^i$ as the first i bits of x coinciding with the first i bits of a. For vectors $x, y \in \mathbb{F}_{2^k}^v$, we denote $\langle x, y \rangle$ as the scalar product of the vectors x and y, defined by

$$\langle x, y \rangle = \sum_{i=1}^{v} x_i y_i.$$

For values a, b from some set X, we denote $[a \stackrel{?}{=} b]$ as the boolean function that checks the equality of two elements of X. In other words

$$[a \stackrel{?}{=} b] = \begin{cases} 0, & \text{if } a \neq b, \\ 1, & \text{if } a = b. \end{cases}$$

3.2 Cryptography

Probabilistic polynomial time (PPT) algorithm Probabilistic polynomial time algorithms are algorithms that run for 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].

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 large value and a uniform large value. More formally,

Definition 1. *Let X and Y be sets such that $|X| < |Y|$. We say that a function $f : X \to Y$ is a $\varepsilon_{\mathsf{PRG}}$-**pseudorandom generator** (PRG), if for every PPT adversary $\mathcal{A}$*

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

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 t parties, where t is a threshold set by the scheme [Sha79,Bla79,KL23]. A piece of data held by one party is called a **secret share**. Let us denote the set of parties with $\mathcal{P} := \{P_1, \dots, P_n\}$.

Definition 2. *A **perfect secret sharing scheme** for a set of secrets S, threshold t and set of parties $\mathcal{P}$ consists of a pair of PPT algorithms (*share, $\mathsf{reconstruct}$*), where*

- share*: On input $s \in S$ outputs shares $\sigma_1, \dots, \sigma_n \in \{0,1\}^*$;*

– reconstruct*: On inputs (i, σ_i) from parties $P_i \in \hat{\mathcal{P}} \subseteq \mathcal{P}$, outputs the secret $s' \in S$ or the error symbol $\perp$.*

This scheme must satisfy the following properties:

– ***Completeness**: For every $s \in S$ and every subset of parties $\hat{\mathcal{P}} \subseteq \mathcal{P}$, that satisfy $\left|\hat{\mathcal{P}}\right| \geq t$*

$$\Pr\left[s' = s \mid (\sigma_1, \ldots, \sigma_n) \leftarrow \mathsf{share}(s); s' \leftarrow \mathsf{reconstruct}(\{P_i() : P_i \in \hat{\mathcal{P}}\})\right] = 1,$$

where $P_i() := (i, \sigma_i)$ denotes the output of the i-th party;

– ***Privacy**: For every subset of parties $\hat{\mathcal{P}} \subset \mathcal{P}$, that satisfy $\left|\hat{\mathcal{P}}\right| < t$, and every PPT algorithm $\mathcal{A}$*

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

where $P_i() := (i, \sigma_i)$ denotes the output of the i-th party and p_{triv} denotes the trivial probability of guessing the secret or guessing the shares of other parties

$$p_{triv} := \max\{\max_{s \in S}\{\Pr[s \leftarrow S]\}, \max_{\substack{P_i \in \mathcal{P} \setminus \hat{\mathcal{P}} \\ \sigma_i \in \{0,1\}^*}}\{\Pr[\sigma_i \leftarrow \mathsf{share}(s)]\}\}.$$

Definition 3. *Let $\mathbb{G}$ be an abelian group. **Additive secret sharing** is a perfect secret sharing scheme with threshold $t = n = |\mathcal{P}|$ and $p_{triv} = \max_{s \in \mathbb{G}}\{\Pr[s \leftarrow \mathbb{G}]\}$, for which*

– share*: On input $s \in \mathbb{G}$, samples $\sigma_1, \ldots, \sigma_{n-1} \xleftarrow{\$} \mathbb{G}$. Then computes*

$$\sigma_n = s - \sum_{i=1}^{n-1} \sigma_i$$

and outputs shares $\sigma_1, \ldots, \sigma_n$.

– reconstruct*: On input of all n shares $\sigma_1, \ldots, \sigma_n$, outputs $\sum_{i=1}^{n} \sigma_i$. On any other subset of input shares outputs $\perp$.*

For secret shared value τ from some abelian group $\mathbb{G}$ and a two-party additive secret sharing scheme, we denote $[\tau]_1$ and $[\tau]_2$ as the secret shares of party 1 and party 2. That is

$$[\tau]_1 + [\tau]_2 = \tau.$$

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 $f : \mathbb{G} \to \mathbb{G}$, a value $\tau \in \mathbb{G}$, and a sharing of τ that is $[\tau] = ([\tau]_1, [\tau]_2)$, 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 $[z]_2 \leftarrow f([\tau]_2)$.

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]. We now will give the more formal definition.

Definition 4 ([BGI15]). *Let $p \in \mathbb{N}$. A **p-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 (Gen, Eval), where*

- Gen*: Gets the description of $\varphi \in \mathcal{F}$ as its input and it outputs p keys $k_1, \ldots, k_p \in \{0,1\}^*$.*
- 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 $i \in \{1, \ldots, p\}$, $k_i \in \{0,1\}^*$ and $x \in X$.*

The scheme must satisfy the following properties:

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

$$\mathsf{Pr}[\mathsf{reconstruct}(\mathsf{Eval}(1, k_1, x), \ldots, \mathsf{Eval}(p, k_p, x)) = \varphi(x) | \\ k_1, \ldots, k_p \leftarrow \mathsf{Gen}(\varphi)] = 1.$$

- ***Security**: We call an FSS scheme $(Q, \varepsilon_{\mathsf{FSS}})$-indistinguishable, if for all corrupted parties $T \subset \{1, \ldots, p\}$, $|T| \leq Q < p$ and for all PPT algorithms $\mathcal{A}$:*

$$\Pr\left[b = \bar{b}\right] - \frac{1}{2} \leq \varepsilon_{\mathsf{FSS}},$$

where b and $\bar{b}$ come from the following experiment:
  - *The adversary outputs $(\varphi_1, \varphi_2, \sigma) \leftarrow \mathcal{A}()$, 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 $b \leftarrow \{0,1\}$ and calculates $(k_1, \ldots, k_p) \leftarrow \mathsf{Gen}(\varphi_b)$.*
  - *The adversary gets the corrupted keys and calculates $\bar{b} \leftarrow \mathcal{A}(\{k_i\}_{i \in T}, \sigma)$.*

  Note that in the following sections we denote $\mathsf{Eval}(i, k_i, x)$ as $\mathsf{Eval}^i(k_i, x)$.

Distributed point functions (DPF)

Definition 5. *Let X be a set and let $\mathbb{G}$ be an abelian group. For $a \in X$ and $b \in \mathbb{G}$ the **point function** $P_{a,b} : X \to \mathbb{G}$ is defined by $P_{a,b}(a) = b$ and $P_{a,b}(x) = 0$ for all $x \neq a$.*

Definition 6 ([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 $\mathbb{G} = \{0,1\}^m$ where the group operation is defined to be as the pointwise XOR.*

Definition 7 ([BCGI18]). *Let X be a set and let $\mathbb{G}$ be an abelian group. For $\boldsymbol{a} \in X^t$ and $\boldsymbol{b} \in \mathbb{G}^t$ the t-**multi-point function** $P_{\boldsymbol{a},\boldsymbol{b}}$ is defined by*

$$P_{\boldsymbol{a},\boldsymbol{b}}(x) = \begin{cases} b_i, & \text{if } x = a_i, \\ 0, & \text{otherwise}, \end{cases}$$

where a_i is the i-th element of $\boldsymbol{a}$ and b_i is the i-th element of $\boldsymbol{b}$.

Ideal Cipher Model We will use the Ideal Cipher Model in this paper. This essentially assumes that the output of a block cipher with a known key is randomly distributed. Note that we will use the model in only 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 regarding some subspaces. We consider this a reasonable assumption to make as this assumption is typically made in this line of work.

Definition 8 ([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 9 ([CPS08]). *The **Ideal Cipher Model** (ICM) is an idealized 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 will 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 will state this in more detail in Section 5.

3.3 Statistical distance and computational distance

In this section statistical distance comes from [Gol01] and computational distance from [KL08].

Definition 10 (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 11. *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*

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

Definition 12. *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:*

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

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

Definition 13. *We call two cryptographic games $\mathcal{G}_0$ and $\mathcal{G}_1$ ε-**close**, if for all PPT algorithms $\mathcal{A}$*

$$|\Pr[\mathcal{G}_0(\mathcal{A}) = 1] - \Pr[\mathcal{G}_1(\mathcal{A}) = 1]| \leq \varepsilon,$$

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:** Two ensembles can also be called ε-**close**, if the statistical distance between them is less or equal than ε.*

Theorem 1 (Bernoulli inequality). *[Car00]. Let $a \in \mathbb{R}$, $n \in \mathbb{N}$. If $a > -1$ and $n \geq 1$, then*

$$(1-a)^n \geq 1 - na.$$

Now let $\mathbb{F}$ be a field and $m, n \in \mathbb{N}$. The next definitions and theorems come from [AR91].

Definition 14. *Let $r \in \mathbb{N}$ and $v_1, \ldots, v_r$ be vectors in a vector space V over the field $\mathbb{F}$. Vector $w \in V$ is called a **linear combination** of the vectors $v_1, \ldots, v_r$, if there exist $k_1, \ldots, k_r \in \mathbb{F}$ such that*

$$w = k_1 v_1 + \ldots + k_r v_r.$$

Definition 15. *Let $r \in \mathbb{N}$ and $v_1, \ldots, v_r$ be vectors in a vector space V. We call subspace $\tilde{V}$ of V the **space spanned by** $v_1, \ldots, v_r$ if every $\tilde{v} \in \tilde{V}$ can be expressed as a linear combination of $v_1, \ldots, v_r$ and every linear combination of $v_1, \ldots, v_r$ is in $\tilde{V}$.*

Definition 16. *If A is a $m \times n$ matrix over the field $\mathbb{F}$, then the subspace of $\mathbb{F}^n$ spanned by the row vectors of A is called the **row space** of A. The subspace of $\mathbb{F}^m$ spanned by the column vectors of A is called the **column space** of A. The solution space of a homogeneous system of linear equations $Ax = 0$ is called the **nullspace** or the **kernel** of A.*

Theorem 2. *For any matrix A over $\mathbb{F}$, the dimension of the row space and the dimension of the column space are equal.*

Definition 17. *The common dimension of the row space and the column space of matrix A is called the **rank** of the matrix. It is denoted by $\mathsf{rank}(A)$.*

Definition 18. *The dimension of the nullspace of A is called **nullity** of A and is denoted by $\mathsf{nullity}(A)$.*

Theorem 3. *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 4 (Dimension theorem of matrices). *If A is a matrix with n columns then*

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

4 Algorithms

From now on let $k, n, t, v \in \mathbb{N}$, $v \geq t+1$, $f : \mathbb{F}_{2^k} \to \mathbb{F}_{2^k}^{v+1}$ be a $\varepsilon_{\mathsf{PRG}}$-PRG, $(a_1, \ldots, a_t) \in (\{0,1\}^n)^t$ and $(b_1, \ldots, b_t) \in \mathbb{F}_{2^k}^t$. We also assume that f to be in the ideal cipher model. We require that if $i, j \in \{1, \ldots, t\}$ and $i \neq j$, then $a_i \neq a_j$ The values a_j, b_j define a multi-point function $\varphi : \{0,1\}^n \to \mathbb{F}_{2^k}$ such that for all $j \in \{1, \ldots, t\}$

$$\varphi(x) = \begin{cases} b_j, \text{ if } x = a_j, \\ 0, \text{ otherwise.} \end{cases}$$

A function secret sharing scheme consists of two algorithms – the key generation algorithm (Gen) and the evaluation algorithm (Eval). The following subsections define these algorithms. The idea is to generalize DPF construction of Boyle et. al [BGI15,BGI16] to the multi-point function case. This means the keys returned by the key generation algorithm should define complete binary trees with k levels.

To evaluate φ on x both parties essentially move bit by bit down the binary trees defined by the keys given to the parties by Gen. This means 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. If a path from the root node to some leaf node follows some a_j, then we call that path **an alive path**. Every other path from root to leaf is called **a dead path**. We call nodes that lie on some alive path **alive nodes** and other nodes **dead nodes**. The goal is to have a shared secret of 0 at the dead nodes and non-zero shared secret at alive nodes.

To achieve this we will use linear algebra. Both parties have to apply a linear function to the values at the leaf nodes to get either a shared secret of 0, if the last node was dead, or b_j, if the alive path followed was a_j. Note that every alive node has to have at least one alive child, but it can have two. Also note that there are at most t alive nodes at each level of the binary tree. Dead nodes cannot have any alive children, therefore our construction must preserve the deadness of nodes. Similarly to [BGI16], the key part we use in accomplishing this goal is that we are working over a field of characteristic 2. This means the sum of two equal elements is always zero and if the sum of two elements is zero, then they are equal. We use a deterministic PRG to guarantee that, if both secret sharings become the same, then they stay the same.

For example, in Figure 1 alive paths are colored green, alive nodes are colored green and dead nodes are colored black and 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 $(1011, 010)$. On the left we can see for each input at the corresponding leaf the values that the parties will obtain by evaluating the algorithm locally and on the right we see the resulting shared secret on all the leaves. Note that on the left the parties locally cannot tell which nodes are alive and which are dead.

4.1 Key generation algorithm

The key generation algorithm (formalised in Algorithm 1 and Algorithm 2) works as follows:

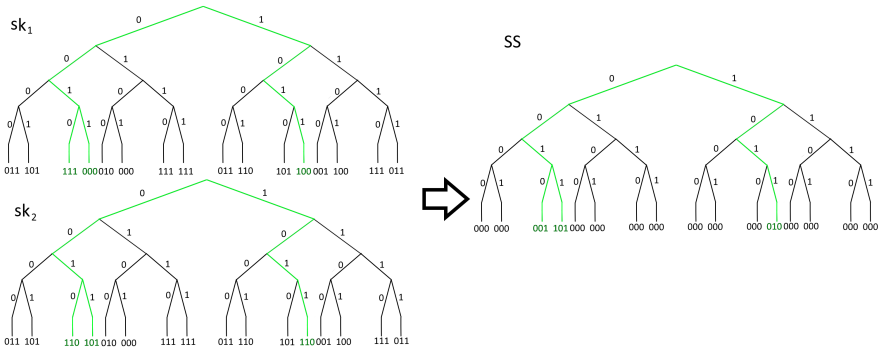

Fig. 1. Illustration of SLAMP-FSS

1. Sample X_ϵ from $\mathbb{F}_{2^k}^v \setminus \{0\}$ and τ_ϵ from $\mathbb{F}_{2^k}$. Sample shares of $(X_\epsilon, \tau_\epsilon)$ for party one and party two. Add ϵ to set R_0.
2. Iterate over $i \in \{1, \ldots, n\}$ and at each iteration do the following:
   (a) Randomly sample two different values $w_{i,0}$ and $w_{i,1}$ from $\mathbb{F}_{2^k}$. They have to be different, because this ensures that alive nodes with exactly one dead child can kill only one child and not both of them.
   (b) Iterate over the set R_{i-1} (set of alive nodes) and separate its nodes' children $r||b$ into alive nodes R_i and dead nodes R'_i. Here $r \in \{0,1\}^{i-1}$ and $b \in \{0,1\}$. Collect the nodes from R_{i-1} with two alive children into the set $\widehat{R}_{i-1}$.
   (c) For each dead node $r||b$ in R'_i, add a constraint $\langle X_r, \boldsymbol{d}_{i-1} \rangle = \tau_r \cdot w_{i,b}$ to a system of linear equations. This kills this dead node. We note that here $\boldsymbol{d}_{i-1}$ denotes the variables in the linear system of equations that we will obtain.
   (d) For each alive node r in $\widehat{R}_{i-1}$ from the previous layer $i-1$ that have two alive children $r||0$ and $r||1$, add a constraint $\langle X_r, \boldsymbol{d}_{i-1} \rangle = \tau_r \cdot w_{r,2}$ 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 with high probability that alive nodes that have an alive sibling are not killed by accident.
   (e) Solve for $\boldsymbol{d}_{i-1} \in \mathbb{F}_{2^k}^v$. Since there are at most t rows in the system of linear equations and $v > t$ variables, then we know there are either no solutions or a lot of solutions. If the system is not solvable, then return $\perp$ and abort. If it is solvable, then sample one of those solutions.
   (f) For each alive node $r||b$ in R_i, set as the new secret shared values

$$
\begin{aligned}
([X_{r||b}]_1, [\tau_{r||b}]_1) &\leftarrow f(\langle [X_r]_1, \boldsymbol{d}_{i-1} \rangle + [\tau_r]_1 \cdot w_{i,b}), \\
([X_{r||b}]_2, [\tau_{r||b}]_2) &\leftarrow f(\langle [X_r]_2, \boldsymbol{d}_{i-1} \rangle + [\tau_r]_2 \cdot w_{i,b}), \\
(X_{r||b}, \tau_{r||b}) &\leftarrow ([X_{r||b}]_1, [\tau_{r||b}]_1) + ([X_{r||b}]_2, [\tau_{r||b}]_2).
\end{aligned}
$$

   Note that we do not have to do this operation for dead nodes, because at each iteration we are only using the secret shares of the values in the alive nodes.

3. Solve linear system of equations, where for every $j \in \{1, \ldots, t\}$

$$\left\langle X_{a_j}, \boldsymbol{g} \right\rangle = b_j + \tau_{a_j},$$

and uniformly sample a solution to this. If the system is not solvable, then return $\perp$ and abort.
4. Set $([X_\epsilon]_1, [\tau_\epsilon]_1, \{w_{i,0}\}_{i=1}^n, \{w_{i,1}\}_{i=1}^n, \{\boldsymbol{d}_i\}_{i=0}^{n-1}, \boldsymbol{g})$ as the first key sk_1 and send it to P_1.
5. Set $([X_\epsilon]_2, [\tau_\epsilon]_2, \{w_{i,0}\}_{i=1}^n, \{w_{i,1}\}_{i=1}^n, \{\boldsymbol{d}_i\}_{i=0}^{n-1}, \boldsymbol{g})$ as the second key sk_2 and send it to P_2.

Algorithm 1: Gen – Key generator for the scheme

Input : $\{(a_1, b_1), \ldots, (a_t, b_t)\}, f$

$[X_\epsilon]_1 \xleftarrow{\$} \mathbb{F}_{2^k}^v$;
$[X_\epsilon]_2 \xleftarrow{\$} \mathbb{F}_{2^k}^v \setminus \{[X_\epsilon]_1\}$;
$X_\epsilon \leftarrow [X_\epsilon]_1 + [X_\epsilon]_2$;
$[\tau_\epsilon]_1, [\tau_\epsilon]_2 \xleftarrow{\$} \mathbb{F}_{2^k}$;
$\tau_\epsilon \leftarrow [\tau_\epsilon]_1 + [\tau_\epsilon]_2$;
$R_0 \leftarrow \{\epsilon\}$, where ϵ is an empty string;
for $i = 1$ **to** n **do**
  $w_{i,0}, w_{i,1}, \boldsymbol{d}_{i-1}, R_i \leftarrow \mathsf{SubGen}()$;
$A \leftarrow \begin{pmatrix} X_{a_1} \\ \vdots \\ X_{a_t} \end{pmatrix}$;
$\boldsymbol{B} \leftarrow \begin{pmatrix} b_1 + \tau_{a_1} \\ \vdots \\ b_t + \tau_{a_t} \end{pmatrix}$;
Solve $A\boldsymbol{g} = \boldsymbol{B}$ and sample $\boldsymbol{g}$ from the solution space ;
$sk_1 \leftarrow [X_\epsilon]_1, [\tau_\epsilon]_1, \{w_{i,0}\}_{i=1}^n, \{w_{i,1}\}_{i=1}^n, \{\boldsymbol{d}_i\}_{i=0}^{n-1}, \boldsymbol{g}$;
$sk_2 \leftarrow [X_\epsilon]_2, [\tau_\epsilon]_2, \{w_{i,0}\}_{i=1}^n, \{w_{i,1}\}_{i=1}^n, \{\boldsymbol{d}_i\}_{i=0}^{n-1}, \boldsymbol{g}$;
Give to P_1 the key sk_1;
Give to P_2 the key sk_2;

4.2 Evaluation algorithm

The evaluation algorithm for party $P \in \{1, 2\}$ on input $x = x^1 || x^2 || \ldots || x^n \in \{0, 1\}^n$ works as follows:

1. Set $([\bar{X}_0]_P, [\bar{\tau}_0]_P) \leftarrow ([X_\epsilon]_P, [\tau_\epsilon]_P)$.
2. Iterate over $i \in \{1, \ldots, n\}$ and do the following:
   (a) Calculate $z_i \leftarrow \left\langle [\bar{X}_{i-1}]_P, \boldsymbol{d}_{i-1} \right\rangle + [\bar{\tau}_{i-1}]_P \cdot w_{i,x^i}$.

Algorithm 2: SubGen – Subroutine for Gen(Algorithm 1)

Input : The state of Gen

Output : $w_{i,0}, w_{i,1}, \boldsymbol{d}_{i-1}, R_i$

$w_{i,0} \xleftarrow{\$} \mathbb{F}_{2^k} \setminus \{0\}$;
$w_{i,1} \xleftarrow{\$} \mathbb{F}_{2^k} \setminus \{0, w_{i,0}\}$;
$R_i \leftarrow \emptyset$;
$R'_i \leftarrow \emptyset$;
$\widehat{R}_{i-1} \leftarrow \emptyset$;
for $r \in R_{i-1}$ **do**
  for $b \in \{0,1\}$ **do**
    if $\exists a_j$ *such that* a_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**
    $\widehat{R}_{i-1} \leftarrow \widehat{R}_{i-1} \cup \{r\}$;
Let A be a $|R_{i-1}| \times v$ matrix of zeroes;
Let $\boldsymbol{B}$ be a zero column vector of length $|R_{i-1}|$;
$j_i \leftarrow 1$;
for $r||b \in R'_i$ **do**
  Set the j_i-th row of A to be X_r;
  Set the j_i-th element of $\boldsymbol{B}$ to be $\tau_r \cdot w_{i,b}$;
  $j_i \leftarrow j_i + 1$;
for $r \in \widehat{R}_{i-1}$ **do**
  $w_{r,2} \xleftarrow{\$} \mathbb{F}_{2^k} \setminus \{w_{i,0}, w_{i,1}\}$;
  Set the j_i-th row of A to be X_r;
  Set the j_i-th element of $\boldsymbol{B}$ to be $\tau_r \cdot w_{r,2}$;
  $j_i \leftarrow j_i + 1$;
Solve $A\boldsymbol{d}_{i-1} = \boldsymbol{B}$ and sample $\boldsymbol{d}_{i-1}$ from the solution space;
for $r||b \in R_i$ **do**
  $([X_{r||b}]_1, [\tau_{r||b}]_1) \leftarrow f(\langle [X_r]_1, \boldsymbol{d}_{i-1} \rangle + [\tau_r]_1 \cdot w_{i,b})$;
  $([X_{r||b}]_2, [\tau_{r||b}]_2) \leftarrow f(\langle [X_r]_2, \boldsymbol{d}_{i-1} \rangle + [\tau_r]_2 \cdot w_{i,b})$;
  $X_{r||b} \leftarrow [X_{r||b}]_1 + [X_{r||b}]_2$;
  $\tau_{r||b} \leftarrow [\tau_{r||b}]_1 + [\tau_{r||b}]_2$;
return $w_{i,0}, w_{i,1}, \boldsymbol{d}_{i-1}, R_i$;

(b) Evaluate $([\bar{X}_i]_P, [\bar{\tau}_i]_P) \leftarrow f([z_i]_P)$. If $r||x^i := x^1||x^2||\dots||x^i$ is an alive node, then we have got $([\bar{X}_i]_P, [\bar{\tau}_i]_P) = ([X_{r||x^i}]_P, [\tau_{r||x^i}]_P)$. If $r||x^i$ is a dead node, then r is either alive or dead. If r is alive then from $\langle X_r, \boldsymbol{d}_{i-1}\rangle = \tau_r \cdot w_{i,x^i}$, we can deduce that

$$\langle [X_r]_1, \boldsymbol{d}_{i-1}\rangle + [\tau_r]_1 \cdot w_{i,x^i} = \langle [X_r]_2, \boldsymbol{d}_{i-1}\rangle + [\tau_r]_2 \cdot w_{i,x^i}.$$

This means both parties use the same argument in f and since f is deterministic, then we know that both parties have the same value as their secret shares, which means that their shared secret is 0. And since the evaluation algorithm for both parties continues exactly the same, we know that both secret shares will stay the same until the end of the protocol. If r is dead, then we know that for some ancestor of r must be dead and the same reasoning follows.

3. Calculate $z \leftarrow \langle X_n, \boldsymbol{g}\rangle + \tau_n$.

We formalise this in Algorithm 4. To make the analysis easier we have separated the subroutine that is evaluated at each level to a separate Algorithm 3.

Algorithm 3: Eval_i^P – Subroutine for the evaluator in Algorithm 4

Input : $[\bar{X}_{i-1}]_P, [\bar{\tau}_{i-1}]_P, w_{i,0}, w_{i,1}, \boldsymbol{d}_{i-1}, f, x^i$

$[z_i]_P \leftarrow \left\langle [\bar{X}_{i-1}]_P, \boldsymbol{d}_{i-1}\right\rangle + [\bar{\tau}_{i-1}]_P \cdot w_{i,x^i}$;

$([\bar{X}_i]_P, [\bar{\tau}_i]_P) \leftarrow f([z_i]_P)$;

return (X_i, τ_i)

Algorithm 4: Eval^P – Evaluation algorithm

Input : $sk_P, f, x = x^1||\dots||x^n$

$[X_\epsilon]_P, [\tau_\epsilon]_P, \{w_{i,0}\}_{i=1}^n, \{w_{i,1}\}_{i=1}^n, \{\boldsymbol{d}_i\}_{i=0}^{n-1}, \boldsymbol{g} \leftarrow sk_P$;

$([\bar{X}_0]_P, [\bar{\tau}_0]_P) \leftarrow ([X_\epsilon]_P, [\tau_\epsilon]_P)$;

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

  $([\bar{X}_i]_P, [\bar{\tau}_i]_P) \leftarrow \mathsf{Eval}_i^P([\bar{X}_{i-1}]_P, [\bar{\tau}_{i-1}]_P, w_{i,0}, w_{i,1}, \boldsymbol{d}_{i-1}, f, x^i)$;

$[z]_P \leftarrow \left\langle [\bar{X}_n]_P, \boldsymbol{g}\right\rangle + \tau_n$;

return z

In Appendix B, we will provide a slightly different version of our algorithm along with altered security proofs. That version of the algorithm will output random values instead of predetermined $(b_1, \dots, b_t)$, but the last evaluation of the PRG is omitted. In the full evaluation, this will halve the amount of PRG-evaluations needed, bringing it down from $2 \cdot 2^n$ to 2^n.

5 Properties of SLAMP-FSS

In this paper, we model the PRG f in the Ideal Cipher Model. Let us assume we have the ideal block cipher $F : \{0,1\}^{k(v+1)} \times \{0,1\}^k \to \{0,1\}^{k(v+1)}$. Sample $x \in \{0,1\}^{k(v+1)}$, fix it, and define an ideal PRG $\hat{f}$ as $F(x,\cdot)$. Since F is sampled randomly from all permutations and also x is uniform, then $\hat{f}$ produces uniformly random elements from $\{0,1\}^{k(v+1)}$. We can interpret these as elements of $\mathbb{F}_{2^k}^{v+1}$. In the following proofs we assume that our PRG f behaves like $\hat{f}$ in all matters that are relevant.

We have to prove that the following properties hold, then we can easily reason that this protocol satisfies the conditions for being an FSS scheme.

1. Algorithm 2 has a low probability of error assuming its inputs are correct.
2. If the secret shared by the parties P_1 and P_2 is [0], then after one evaluation cycle by Algorithm 3 the secret is still [0].
3. If $(X_{i-1}, \tau_{i-1}) \neq 0$, then $x^1 x^2 \dots x^{i-1} = a_j^1 a_j^2 \dots a_j^{i-1}$ for at least one of the a_j.
4. For all i and all j with high probability
$$(X_{a_j^1||\dots||a_j^i}, \tau_{a_j^1||\dots||a_j^i}) \neq 0.$$
5. Algorithm 1 has a low probability of error assuming its inputs are correct.
6. If there exists $j \in \{1,\dots,t\}$ such that $x = a_j$, then executing Algorithms 1 and 4 produces a shared secret of b_j.
7. If there does not exist $j \in \{1,\dots,t\}$ such that $x = a_j$, then executing Algorithms 1 and 4 produces a shared secret of 0.
8. It is not possible to tell any information about a_j or b_j, for any $j \in \{1,\dots,t\}$, from the viewpoint of one party.

5.1 Error probability of SubGen

The problem with solving systems of linear equations is that they might not be solvable. Therefore, there is a small chance that Algorithm 2 will terminate with an error.

Theorem 5. *The probability that Algorithm 2 will end with an error provided that its inputs are correct is*

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

By correct inputs we mean that the values SubGen *uses are correctly defined and for all alive nodes r of the previous level: $(X_r, \tau_r) \neq 0$.*

Proof. Let us analyse what is the probability of success and look at the system of linear equations $A\boldsymbol{d} = \boldsymbol{B}$. At first let us assume that $A \in \mathbb{F}_{2^k}^{t' \times v}$ is fixed, where $t' \leq v$. From Theorem 3 we get that $A\boldsymbol{d} = \boldsymbol{B}$ is solvable iff $\mathsf{rank}(A) = \mathsf{rank}(A \mid \boldsymbol{B})$.

We know that the rank of a matrix shows the number of independent rows in the matrix. Thus, there exist $\mathsf{rank}(A)$ independent rows of A. The same rows must be independent in matrix $(A \mid \boldsymbol{B})$, because adding elements to independent vectors cannot make those vectors dependent. For $\mathsf{rank}(A) = \mathsf{rank}(A \mid \boldsymbol{B})$ to hold, the dependent rows of A must also be dependent in $(A \mid \boldsymbol{B})$. Since these rows are dependent on the independent rows, then there are $\mathsf{rank}(A)$ elements of $\boldsymbol{B}$ that we can choose freely and $t' - \mathsf{rank}(A)$ elements that are determined by them. Therefore, for a fixed A there are $(2^k)^{\mathsf{rank}(A)}$ different vectors $\boldsymbol{B}$ for which $\mathsf{rank}(A) = \mathsf{rank}(A \mid \boldsymbol{B})$ and the probability of getting such vector is thus

$$\Pr_{\boldsymbol{B} \xleftarrow{\$} \mathbb{F}_{2^k}^{t'}} [A\boldsymbol{d} = \boldsymbol{B} \text{ is solvable}] = \frac{(2^k)^{\mathsf{rank}(A)}}{(2^k)^{t'}}.$$

However, A is not fixed, it is also uniformly random. Therefore, if we want to calculate the probability of success we have to sum over all A. Hence

$$\begin{aligned}
&\Pr_{\substack{A \xleftarrow{\$} \mathbb{F}_{2^k}^{t' \times v} \\ \boldsymbol{B} \xleftarrow{\$} \mathbb{F}_{2^k}^{t'}}} [A\boldsymbol{d} = \boldsymbol{B} \text{ is solvable}] = \\
&= \Pr_{\boldsymbol{B} \xleftarrow{\$} \mathbb{F}_{2^k}^{t'}} [A\boldsymbol{d} = \boldsymbol{B} \text{ is solvable} \mid \mathsf{rank}(A) = 0] \cdot \Pr_{A \xleftarrow{\$} \mathbb{F}_{2^k}^{t' \times v}} [\mathsf{rank}(A) = 0] + \ldots + \\
&+ \Pr_{\boldsymbol{B} \xleftarrow{\$} \mathbb{F}_{2^k}^{t'}} [A\boldsymbol{d} = \boldsymbol{B} \text{ is solvable} \mid \mathsf{rank}(A) = t'] \cdot \Pr_{A \xleftarrow{\$} \mathbb{F}_{2^k}^{t' \times v}} [\mathsf{rank}(A) = t'] = \\
&= \sum_{j=0}^{t'} \Pr_{\boldsymbol{B} \xleftarrow{\$} \mathbb{F}_{2^k}^{t'}} [A\boldsymbol{d} = \boldsymbol{B} \text{ is solvable} \mid \mathsf{rank}(A) = j] \cdot \Pr_{A \xleftarrow{\$} \mathbb{F}_{2^k}^{t' \times v}} [\mathsf{rank}(A) = j] = \\
&= \sum_{j=0}^{t'} \left(\frac{(2^k)^j}{(2^k)^{t'}} \cdot \frac{\left| \{A \mid A \in \mathbb{F}_{2^k}^{t' \times v} \wedge \mathsf{rank}(A) = j\} \right|}{(2^k)^{t' \cdot v}} \right).
\end{aligned}$$

The number of A with rank j is

$$\left| \{A \mid A \in \mathbb{F}_{2^k}^{t' \times v} \wedge \mathsf{rank}(A) = j\} \right| = (2^k)^{\frac{j(j-1)}{2}} \cdot \prod_{i=0}^{j-1} \frac{((2^k)^{v-i} - 1)((2^k)^{t'-i} - 1)}{(2^k)^{i+1} - 1}$$

according to [MP13]. Therefore success of a single call of SubGen is

$$\begin{aligned}
&\sum_{j=0}^{t'} \left(\frac{(2^k)^j}{(2^k)^{t'}} \cdot \frac{(2^k)^{\frac{j(j-1)}{2}} \cdot \prod_{i=0}^{j-1} \frac{((2^k)^{v-i}-1)((2^k)^{t'-i}-1)}{(2^k)^{i+1}-1}}{(2^k)^{t' \cdot v}} \right) = \\
&= \frac{1}{(2^k)^{t'(v+1)}} \sum_{j=0}^{t'} \left((2^k)^{\frac{j(j+1)}{2}} \cdot \prod_{i=0}^{j-1} \frac{((2^k)^{v-i} - 1)((2^k)^{t'-i} - 1)}{(2^k)^{i+1} - 1} \right),
\end{aligned}$$

where $t' = |R_{i-1}|$. Let us bound it further to show that this value is close to 1. Notice that the last element of the sum is also the biggest. Hence

$$\Pr[\text{success}] \geq \frac{(2^k)^{\frac{t'(t'+1)}{2}}}{(2^k)^{t'(v+1)}} \prod_{i=0}^{t'-1} \frac{((2^k)^{v-i} - 1)((2^k)^{t'-i} - 1)}{(2^k)^{i+1} - 1}.$$

In the product we divide with the following elements $(2^k)^1-1$, $(2^k)^2-1$, …, $(2^k)^{t'}-1$ and multiply by $(2^k)^1-1$, $(2^k)^2-1$, …, $(2^k)^{t'}-1$. Thus, we get

$$\Pr\left[\text{success}\right] \geq \frac{(2^k)^{\frac{t'(t'+1)}{2}}}{(2^k)^{t'(v+1)}} \prod_{i=0}^{t'-1} \left((2^k)^{v-i}-1\right).$$

Notice that $\frac{t'(t'+1)}{2}$ is the sum of an arithmetic series from 1 to t' with step 1 (see [Gra72] for example). We have

$$\begin{aligned}\Pr\left[\text{success}\right] &\geq \frac{(2^k)^{1+2+\ldots+t'}}{(2^k)^{t'v+t'}} \prod_{i=0}^{t'-1} \left((2^k)^{v-i}-1\right) = \\ &= \frac{(2^k)^0 \cdot \ldots \cdot (2^k)^{t'}}{(2^k)^{t'v}(2^k)^{t'}} \prod_{i=0}^{t'-1} \left((2^k)^{v-i}-1\right) = \\ &= \prod_{j=0}^{t'-1} \frac{(2^k)^j}{(2^k)^v} \prod_{i=0}^{t'-1} \left((2^k)^{v-i}-1\right).\end{aligned}$$

Let us combine the two products and get

$$\Pr\left[\text{success}\right] \geq \prod_{i=0}^{t'-1} \frac{(2^k)^i}{(2^k)^v} \left((2^k)^{v-i}-1\right) = \prod_{i=0}^{t'-1} \left(1-\frac{1}{(2^k)^{v-i}}\right).$$

The smallest term in this product is when $i=t'-1$, thus

$$\Pr\left[\text{success}\right] \geq \prod_{i=0}^{t'-1} \left(1-\frac{1}{(2^k)^{v-t'+1}}\right) = \left(1-\frac{1}{(2^k)^{v-t'+1}}\right)^{t'}.$$

Now from the Bernoulli inequality (1) we get

$$\Pr\left[\text{success}\right] \geq \left(1-\frac{1}{(2^k)^{v-t'+1}}\right)^{t'} \geq 1-\frac{t'}{(2^k)^{v-t'+1}}.$$

Since $t \geq t'$, then

$$\Pr\left[\text{success}\right] \geq 1-\frac{t'}{(2^k)^{v-t'+1}} \geq 1-\frac{t}{(2^k)^{v-t+1}}.$$

And thus the probability of error is

$$\Pr\left[\text{Algorithm 2 returns } \perp\right] \leq 1-1+\frac{t}{(2^k)^{v-t+1}} = \frac{t}{(2^k)^{v-t+1}}.$$

□

5.2 Dead nodes' children stay dead

If the secret shared by the parties P_1 and P_2 is [0], then after one evaluation cycle by Algorithm 3 the secret is still [0]. This can be stated more formally as follows.

Theorem 6. *For all* $i \in \{1, \ldots, n\}$; $x^i \in \{0,1\}$; $[X_{i-1}]_1, [X_{i-1}]_2 \in \mathbb{F}_{2^k}^v$; $[\tau_{i-1}]_1, [\tau_{i-1}]_2 \in \mathbb{F}_{2^k}$ *and all key pairs generated by* Gen*: if* $[X_{i-1}]_1 = [X_{i-1}]_2$ *and* $[\tau_{i-1}]_1 = [\tau_{i-1}]_2$ *and*

$$([X_i]_1, [\tau_i]_1, [X_i]_2, [\tau_i]_2) \leftarrow \alpha_i([X_{i-1}]_1, [\tau_{i-1}]_1, [X_{i-1}]_2, [\tau_{i-1}]_2, w_{i,0}, w_{i,1}, \boldsymbol{d}_{i-1}, f, x^i),$$

then $[X_i]_1 = [X_i]_2$ *and* $[\tau_i]_1 = [\tau_i]_2$*, where* α_i *is Algorithm 5 and* Eval_i^P *is Algorithm 3.*

Algorithm 5: α_i – One cycle of multiparty computation

Input : $[X_{i-1}]_1, [\tau_{i-1}]_1, [X_{i-1}]_2, [\tau_{i-1}]_2, w_{i,0}, w_{i,1}, \boldsymbol{d}_{i-1}, f, x^i$

1 $([X_{i-1}]_1, [\tau_{i-1}]_1) \leftarrow \mathsf{Eval}_i^1([X_{i-1}]_1, [\tau_{i-1}]_1, w_{i,0}, w_{i,1}, \boldsymbol{d}_{i-1}, f, x^i)$;

2 $([X_{i-1}]_2, [\tau_{i-1}]_2) \leftarrow \mathsf{Eval}_i^2([X_{i-1}]_2, [\tau_{i-1}]_2, w_{i,0}, w_{i,1}, \boldsymbol{d}_{i-1}, f, x^i)$;

3 **return** $([X_{i-1}]_1, [\tau_{i-1}]_1, [X_{i-1}]_2, [\tau_{i-1}]_2)$

Proof. Let us fix $i \in \{1, \ldots, n\}$, $x^i \in \{0,1\}$ and a key pair generated by Gen. Assume $[X_{i-1}]_1 = [X_{i-1}]_2$ and $[\tau_{i-1}]_1 = [\tau_{i-1}]_2$. Now let us calculate α_i

$$\begin{aligned}
[z_i]_1 &\leftarrow \langle [X_{i-1}]_1, \boldsymbol{d}_{i-1} \rangle + [\tau_{i-1}]_1 \cdot w_{i,x^i}, \\
([X_i]_1, [\tau_i]_1) &\leftarrow f([z_i]_1), \\
[z_i]_2 &\leftarrow \langle [X_{i-1}]_2, \boldsymbol{d}_{i-1} \rangle + [\tau_{i-1}]_2 \cdot w_{i,x^i} = \\
&\qquad = \langle [X_{i-1}]_1, \boldsymbol{d}_{i-1} \rangle + [\tau_{i-1}]_1 \cdot w_{i,x^i} = [z_i]_1, \\
([X_i]_2, [\tau_i]_2) &\leftarrow f([z_i]_2) = f([z_i]_1) = ([X_i]_1, [\tau_i]_1).
\end{aligned}$$

Therefore, we get

$$[X_i]_1 = [X_i]_2, \quad \text{and} \quad [\tau_i]_1 = [\tau_i]_2,$$

which is exactly what we wanted to show. □

5.3 Node is alive if it is supposed to be alive

This can be stated more formally as follows.

Theorem 7. *For all* $i \in \{0, \ldots, n\}$ *if* $(X_i, \tau_i) \neq 0$*, then there exists* $j \in \{1, \ldots, t\}$ *such that*

$$x^1 x^2 \ldots x^i = a_j^1 a_j^2 \ldots a_j^i.$$

(Notice that due to the construction of Algorithms 1 and 4, we can be sure that the shared value in the evaluation phase $(\bar{X}_{i-1}, \bar{\tau}_{i-1})$ is the same as $(X_{x^1x^2\ldots x^{i-1}}, \tau_{x^1x^2\ldots x^{i-1}})$ in the key generation phase. This is the reason why we can denote (X_{i-1}, τ_{i-1}) as $(X_{x^1x^2\ldots x^{i-1}}, \tau_{x^1x^2\ldots x^{i-1}})$.)

Proof. Let us prove the claim of the theorem by induction.

Base step: Let $i = 0$. The statement holds trivially, because there is no 0-th bit of x, therefore, it coincides with the beginning of every a_j.

Induction step: Let us assume for $i-1$ the statement holds and denote

$$r := x^1x^2\ldots x^{i-1},$$

i.e.

$$(X_r, \tau_r) \neq 0 \quad \Rightarrow \quad \exists j \in \{1,\ldots,t\} : r = a_j^1 a_j^2 \ldots a_j^{i-1}.$$

Let us show the statement holds for i as well and assume towards contradiction that

$$(X_{r||x^i}, \tau_{r||x^i}) \neq 0 \quad \wedge \quad \nexists j \in \{1,\ldots,t\} : r||x^i = a_j^1 a_j^2 \ldots a_j^i.$$

Since $(X_{r||x^i}, \tau_{r||x^i}) \neq 0$, then property 2 implies that $(X_r, \tau_r) \neq 0$. Now we can use the induction assumption and get that $\exists j \in \{1,\ldots,t\} : r = a_j^1 a_j^2 \ldots a_j^{i-1}$. This means that during the execution of SubGen $r \in R_{i-1}$ and since $\nexists j \in \{1,\ldots,t\} : r||x^i = a_j^1 a_j^2 \ldots a_j^i$, then we know that $r||x^i \in R'_i$. In SubGen lines 17 to 20 the following equation is enforced:

$$\langle X_r, \boldsymbol{d}_{i-1}\rangle = \tau_r \cdot w_{i,x^i},$$

which, because of linearity of the scalar product and the fact that

$$[X_r]_1 + [X_r]_2 = X_r \quad \text{and} \quad [\tau_r]_1 + [\tau_r]_2 = \tau_r,$$

is the same as

$$\langle [X_r]_1, \boldsymbol{d}_{i-1}\rangle + \langle [X_r]_2, \boldsymbol{d}_{i-1}\rangle = [\tau_r]_1 \cdot w_{i,x^i} + [\tau_r]_2 \cdot w_{i,x^i},$$

that is

$$\langle [X_r]_1, \boldsymbol{d}_{i-1}\rangle - [\tau_r]_1 \cdot w_{i,x^i} = -\langle [X_r]_2, \boldsymbol{d}_{i-1}\rangle + [\tau_r]_2 \cdot w_{i,x^i}.$$

Since we are working in $\mathbb{F}_{2^k}$, then we get

$$\langle [X_r]_1, \boldsymbol{d}_{i-1}\rangle + [\tau_r]_1 \cdot w_{i,x^i} = \langle [X_r]_2, \boldsymbol{d}_{i-1}\rangle + [\tau_r]_2 \cdot w_{i,x^i}.$$

Now we can calculate $(X_{r||x^i}, \tau_{r||x^i})$:

$$\begin{aligned}(X_{r||x^i},\tau_{r||x^i}) &= ([X_{r||x^i}]_1, [\tau_{r||x^i}]_1) + ([X_{r||x^i}]_2, [\tau_{r||x^i}]_2) = \\ &= f\left(\langle [X_r]_1, \boldsymbol{d}_{i-1}\rangle + [\tau_r]_1 \cdot w_{i,x^i}\right) + f\left(\langle [X_r]_2, \boldsymbol{d}_{i-1}\rangle + [\tau_r]_2 \cdot w_{i,x^i}\right) = \\ &= f\left(\langle [X_r]_1, \boldsymbol{d}_{i-1}\rangle + [\tau_r]_1 \cdot w_{i,x^i}\right) + f\left(\langle [X_r]_1, \boldsymbol{d}_{i-1}\rangle + [\tau_r]_1 \cdot w_{i,x^i}\right) = \\ &= (0,\ldots,0,0).\end{aligned}$$

This is a contradiction, because we assumed $(X_{r||x^i}, \tau_{r||x^i}) \neq 0$. Therefore, there exists $j \in \{1,\ldots,t\} : r||x^i = a_j^1 a_j^2 \ldots a_j^i$. By the principles of mathematical induction, the statement holds for any $i \in \{0,\ldots,n\}$. □

5.4 Alive nodes are probably not dead

Theorem 8. *Denote by* $(f(x))_k$ *the last* k *bits of* $f(x)$*. For all* $i \in \{0, \ldots, n\}$*, for all* $j \in \{1, \ldots, t\}$ *and for all PRG* $f : \mathbb{F}_{2^k} \to \mathbb{F}_{2^k}^{v+1}$ *for which*

$$\Pr\left[(f(x))_k = (f(x'))_k \wedge x \neq x' : x, x' \xleftarrow{\$} \mathbb{F}_{2^k}\right] \leq \varepsilon_{col},$$

it holds

$$\Pr\left[\tau_{a_j^1||\ldots||a_j^i} \neq 0\right] \geq \left(p_{solvable} \cdot (1 - \varepsilon_{col})^t\right)^i,$$

where the probability is taken over the randomness of Gen *and*

$$p_{solvable} := 1 - \frac{t}{(2^k)^{v-t+1}}.$$

Proof. Let us prove this claim by induction over $i \in \{0, \ldots, n\}$.

Base step: Let $i = 0$. In any case $(X_\epsilon, \tau_\epsilon) \neq 0$ and thus the probability is 1, because the construction of Gen ensures that X_ϵ is not 0. Therefore the claim holds trivially.

Induction step: Let us assume that the claim holds for $i - 1$ and show it also holds for i. Denote

$$p_i := \Pr\left[(\tau_{a_j^1||\ldots||a_j^i}) \neq 0\right],$$

and $r_j := a_j^1|| \ldots ||a_j^{i-1}$.

The proof follows the scheme seen in Figure 2. Nodes of the scheme represent different events and edges between them represent the probability of this event happening on the condition that previous events happened. If there is only one outgoing edge, then the next event happened with probability 1 given the previous events. If there are two outgoing edges, then the sum of the probabilities equals 1, or in other words, the child events of the parent are complements of each other.

Let us start to calculate the probability of the success path from Figure 2. The first question is, if $\tau_{r_j} \neq 0$ for all $j \in \{1, \ldots, t\}$. This happens by the induction assumption with probability

$$p_{i-1} \geq \left(p_{solvable} \cdot (1 - \varepsilon_{col})^t\right)^{i-1}.$$

On the other hand, if there exists $j \in \{1, \ldots, t\}$ such that $\tau_{r_j} = 0$, then it is possible that $(X_{r_j}, \tau_{r_j}) = 0$. Then we know from property 2, that $(X_{r_j||a_j^i}, \tau_{r_j||a_j^i}) = 0$. Therefore, for easier calculation, we count this path as a failure.

Now let us assume for all $j \in \{1, \ldots, t\} : \tau_{r_j} \neq 0$. The next question is, whether $A\boldsymbol{d}_{i-1} = \boldsymbol{B}$ is solvable during the i-th step. If it is not, then Gen aborts and we end in failure. We know from Subsection 5.1 that it is solvable with probability at least

$$p_{solvable} := 1 - \frac{t}{(2^k)^{v-t+1}}.$$

If $A\boldsymbol{d}_{i-1} = \boldsymbol{B}$ is solvable, then a random $\boldsymbol{d}_{i-1}$ is picked. Let us fix this value.

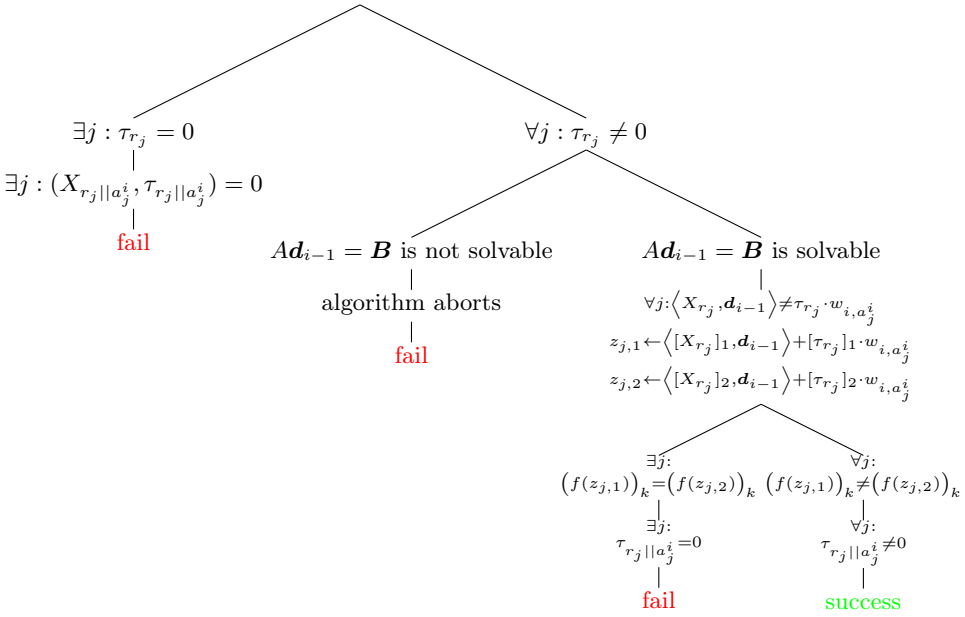

Fig. 2. Proof scheme for property 4.

Let us fix $j \in \{1, \ldots, t\}$. If r has one alive and one dead child, then we know that $r||a_j^i$ is alive and $r||(1-a_j^i)$ is dead. Therefore, the system of linear equations enforces

$$\langle X_{r_j}, \boldsymbol{d}_{i-1}\rangle = \tau_{r_j} \cdot w_{i,(1-a_j^i)}.$$

Since $w_{i,(1-a_j^i)} \neq w_{i,a_j^i}$, then $\langle X_{r_j}, \boldsymbol{d}_{i-1}\rangle \neq \tau_{r_j} \cdot w_{i,a_j^i}$, because $\tau_{r_j} \neq 0$.

If r has two alive children, then we know that the system of linear equations enforces

$$\langle X_{r_j}, \boldsymbol{d}_{i-1}\rangle = \tau_{r_j} \cdot w_{r_j,2}.$$

Since $w_{r_j,2} \neq w_{i,a_j^i}$, then $\langle X_{r_j}, \boldsymbol{d}_{i-1}\rangle \neq \tau_{r_j} \cdot w_{i,a_j^i}$.

Therefore, we know that for all $j \in \{1, \ldots, t\}$ it holds $\langle X_{r_j}, \boldsymbol{d}_{i-1}\rangle \neq \tau_{r_j} \cdot w_{i,a_j^i}$. This also means that

$$\langle [X_{r_j}]_1, \boldsymbol{d}_{i-1}\rangle + [\tau_{r_j}]_1 \cdot w_{i,a_j^i} \neq \langle [X_{r_j}]_2, \boldsymbol{d}_{i-1}\rangle + [\tau_{r_j}]_2 \cdot w_{i,a_j^i}.$$

Let us denote

$$z_{j,1} := \langle [X_{r_j}]_1, \boldsymbol{d}_{i-1}\rangle + [\tau_{r_j}]_1 \cdot w_{i,a_j^i},$$
$$z_{j,2} := \langle [X_{r_j}]_2, \boldsymbol{d}_{i-1}\rangle + [\tau_{r_j}]_2 \cdot w_{i,a_j^i}.$$

During the execution of Gen, $\tau_{r_j||a_j^i}$ is calculated as $(f(z_{j,1}))_k + (f(z_{j,2}))_k$. Now there are two possibilities: on the one hand, if for all $j \in \{1, \ldots, t\} : (f(z_{j,1}))_k \neq$

$(f(z_{j,2}))_k$, which means $\tau_{r_j||a_j^i} \neq 0$, then we succeed. Otherwise, if there exists $j \in \{1,\ldots,t\}$ such that $(f(z_{j,1}))_k = (f(z_{j,2}))_k$, then for that j the equality $\tau_{r_j||a_j^i} = 0$ holds. This is a failure.

What is the probability that for all $j \in \{1,\ldots,t\} : (f(z_{j,1}))_k \neq (f(z_{j,2}))_k$? By our assumption, if we show that the inputs to f are uniformly random and different, then we can conclude that for a fixed j this happens with probability no less than $1 - \varepsilon_{col}$.

Let us thus show that the inputs to f are uniformly random and different. Let us fix $j \in \{1,\ldots,t\}$. We know that $[\tau_{r_j}]_1$ and $[\tau_{r_j}]_2$ are both uniform over $\mathbb{F}_{2^k}$. Therefore, $[\tau_{r_j}]_1 \cdot w_{i,a_j^i}$ and $[\tau_{r_j}]_2 \cdot w_{i,a_j^i}$ are uniform over $\mathbb{F}_{2^k}$, because w_{i,a_j^i} is fixed and nonzero. Thus $z_{j,1}$ and $z_{j,2}$ are also uniform over $\mathbb{F}_{2^k}$. From before we get $z_{j,1} \neq z_{j,2}$. Thus it follows from our assumption about f that for one j the probability of $(f(z_{j,1}))_k \neq (f(z_{j,2}))_k$ is $1 - \varepsilon_{col}$. Assuming all t instances of $z_{j,1}, z_{j,2}$ are independent, the probability that we are looking for is equal or larger than $(1 - \varepsilon_{col})^t$.

Now let us follow the success path:

$$
\begin{aligned}
&p_i \geq p_{i-1} \cdot p_{solvable} \cdot (1 - \varepsilon_{col})^t \geq \\
&\quad \geq \left(p_{solvable} \cdot (1 - \varepsilon_{col})^t\right)^{i-1} \cdot p_{solvable} \cdot (1 - \varepsilon_{col})^t = \left(p_{solvable} \cdot (1 - \varepsilon_{col})^t\right)^i \geq \\
&\quad \geq \left((1 - \varepsilon_{col})^t \cdot \left(1 - \frac{t}{(2^k)^{v-t+1}}\right)\right)^i .
\end{aligned}
$$

Therefore, by the principles of mathematical induction for all $i \in \{0,\ldots,n\}$ the statement holds. □

5.5 The evaluation of an alive path produces correct output

Theorem 9. *For all $x \in \{0,1\}^n$, if there exists $j \in \{1,\ldots,t\}$ such that $x = a_j$, then $[b_j]_1 + [b_j]_2 = b_j$, where $[b_j]_1$ is the output of* Eval^1 *and $[b_j]_2$ 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 $x = a_j$ and that Gen, Eval^1 and Eval^2 succeed. Let $[b_j]_1$ be the output of Eval^1 and $[b_j]_2$ be the output of Eval^2.

$$
\begin{aligned}
[b_j]_1 &= \langle [X_x]_1, \boldsymbol{g} \rangle + [\tau_x]_1, \\
[b_j]_2 &= \langle [X_x]_2, \boldsymbol{g} \rangle + [\tau_x]_2.
\end{aligned}
$$

Since the scalar product is linear,

$$
\begin{aligned}
b_j &= \langle X_x, \boldsymbol{g} \rangle + \tau_x = \langle [X_x]_1 + [X_x]_2, \boldsymbol{g} \rangle + [\tau_x]_1 + [\tau_x]_2 = \\
&= \langle [X_x]_1, \boldsymbol{g} \rangle + [\tau_x]_1 + \langle [X_x]_2, \boldsymbol{g} \rangle + [\tau_x]_2 = [b_j]_1 + [b_j]_2.
\end{aligned}
$$

Therefore, if the protocol succeeds, then P_1 and P_2 have a shared secret of b_j. □

5.6 Error probability of Gen

In the Subsection 5.4, we saw what is the success probability of SubGen in the sense that it does not kill alive nodes and it does not abort. These are the success conditions of Gen as well. However, in the end Gen calculates $\boldsymbol{g}$ as well, which is also a possible point of failure. This subsection captures the entire failure probability of SLAMP-FSS. At first, let us show that, if during the process an alive node has been killed, then Gen aborts (unless that alive path is supposed to evaluate to 0).

Lemma 1. *For all $j \in \{1, \ldots, t\}$, if*

$$(X_{a_j}, \tau_{a_j}) = 0 \quad \textit{and} \quad b_j \neq 0,$$

then Gen *aborts.*

Proof. Let us fix $j \in \{1, \ldots, t\}$ and assume

$$(X_{a_j}, \tau_{a_j}) = 0 \quad \text{and} \quad b_j \neq 0.$$

During the execution of Gen at lines 9 to 11, we try to find $\boldsymbol{g} \in \mathbb{F}_{2^k}^v$ such that

$$\langle X_{a_j}, \boldsymbol{g} \rangle = b_j + \tau_{a_j}.$$

That is

$$\begin{aligned} \langle 0, \boldsymbol{g} \rangle &= b_j + 0, \\ 0 &= b_j. \end{aligned}$$

This is not possible, because $b_j \neq 0$. Therefore, Gen aborts. □

From the previous proof we also get that, if $b_j = 0$, then Gen does not abort, due to this alive node having been killed. This is not a problem, because the end result of the protocol is still correct. The evaluation of desired t-multi-point function on $x = a_j$ is $b_j = 0$.

Theorem 10. *The probability that Algorithm 1 will end with an error provided that its inputs are correct is*

$$\Pr\left[\textit{Algorithm 1 returns } \perp\right] \leq 1 - p_{solvable}^{n+1} \cdot (1 - \varepsilon_{col})^{tn},$$

where ε_{col} is defined as in Theorem 8. and

$$p_{solvable} := 1 - \frac{t}{(2^k)^{v-t+1}}.$$

Proof. From Lemma 1 and Theorem 9 we can conclude that Gen succeeds if Gen does not abort. By succeeding we mean that Gen does not make any keys that do not evaluate to the original multi-point function. Aborting can happen during the executions of SubGen or during the solving of the linear equation $A\boldsymbol{g} = \boldsymbol{B}$. There are three possibilities:

1. For all $j \in \{1, \ldots, t\}$ we have
$$(X_{a_j}, \tau_{a_j}) \neq 0.$$
and for all j in the generation as described through the Figure 2, we got no result fail on all levels $i \in [1, \ldots, n]$.
This holds with probability
$$\Pr\left[(X_{a_j}, \tau_{a_j}) \neq 0\right] \geq \left(p_{solvable} \cdot (1 - \varepsilon_{col})^t\right)^n.$$
From the ideal cipher model we have that in this case (X_{a_j}, τ_{a_j}) are uniformly random for all $j \in \{1, \ldots, t\}$. The question is what is the probability of solving $A\boldsymbol{g} = \boldsymbol{B}$, if A and $\boldsymbol{B}$ are uniformly random. Thus, we have the same case as in Subsection 5.1 and the probability not aborting is greater than $p_{solvable}$.
2. For all $j \in \{1, \ldots, t\}$ we have
$$(X_{a_j}, \tau_{a_j}) \neq 0.$$
but for at least one j at some level $i \in [1, \ldots, n]$, we got the result fail. In this case the algorithm may or may not abort. However, due to it being difficult to evaluate, we will bound the success with 0.
3. There exists $j \in \{1, \ldots, t\}$ such that
$$(X_{a_j}, \tau_{a_j}) = 0.$$
We saw in Lemma 1 that in this case if $b_j \neq 0$, then the algorithm aborts. We do not know what is the probability of $b_j \neq 0$. Thus, it is easier to bound the success with 0.

From previous three branches we get that the probability of success is

$$\Pr[\mathsf{Gen} \text{ succeeds}] \geq p_{solvable}\left(p_{solvable} \cdot (1 - \varepsilon_{col})^t\right)^n + 0 + 0 = p_{solvable}^{n+1} \cdot (1 - \varepsilon_{col})^{tn}.$$

Thus the probability of aborting is

$$\Pr[\mathsf{Gen} \text{ outputs } \perp] \leq 1 - p_{solvable}^{n+1} \cdot (1 - \varepsilon_{col})^{tn}.$$

□

5.7 The evaluation of a dead path produces shared secret of 0

Theorem 11. *For all $x \in \{0,1\}^n$, if there does not exists $j \in \{1, \ldots, t\}$ such that $x = a_j$, then $[z]_1 + [z]_2 = 0$, where $[z]_1$ is the output of Eval^1 and $[z]_2$ is the output of Eval^2, provided that Gen succeeds.*

Proof. Let us fix $x \in \{0,1\}^n$ and assume there does not exists $j \in \{1, \ldots, t\}$ such that $x = a_j$ and that Gen, Eval^1 and Eval^2 succeed. Let $[z_j]_1$ be the output of Eval^1 and $[z_j]_2$ be the output of Eval^2. We know from Gen and property 2, that at some point the shared secret becomes 0 and it stays 0, thus we know that $[X_x]_1 = [X_x]_2$ and $[\tau_x]_1 = [\tau_x]_2$. Let us calculate $[z_j]_1 + [z_j]_2$:

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

This is exactly what we wanted to show. □

5.8 One party cannot evaluate the secret function alone

For readability, the games are moved to Appendix A.2. In the games the notation $\overset{P_1}{\leftarrow}$ denotes the output that is given to P_1 and $\overset{P_2}{\leftarrow}$ denotes the output that is given to P_2.

It is not possible to tell any information about a_j or b_j, for any $j \in \{1, \ldots, t\}$, from the viewpoint of one party. We can define a security definition that should capture property 8.

Definition 19 (FSS real or random indistinguishability). *We call a FSS scheme $\varepsilon_{\mathsf{FSS}}$-ROR-indistinguishable if for every PPT adversary $\mathcal{A}$ it holds*

$$|\Pr[\mathit{Game}\ 1(\mathcal{A}) = 1] - \Pr[\mathit{Game}\ 3(\mathcal{A}) = 1]| \leq \varepsilon_{\mathsf{FSS}},$$

and

$$|\Pr[\mathit{Game}\ 2(\mathcal{A}) = 1] - \Pr[\mathit{Game}\ 3(\mathcal{A}) = 1]| \leq \varepsilon_{\mathsf{FSS}}.$$

Theorem 12. *For all $\varepsilon_{\mathsf{PRG}}$-PRG $f : \mathbb{F}_{2^k} \to \mathbb{F}_{2^k}^{v+1}$:*
SLAMP-FSS is $(t \cdot n \cdot \varepsilon_{\mathsf{PRG}} + (n+1) \cdot \varepsilon_{mat})$-ROR-indistinguishable, where

$$\varepsilon_{mat} = \frac{t}{(2^k)^{v-t+1}}.$$

5.9 Completeness and security

Theorem 13. *SLAMP-FSS is complete. That is, for all $\boldsymbol{a} \in (\{0,1\}^n)^t$, $\boldsymbol{b} \in \mathbb{F}_{2^k}^t$, all input $x \in \{0,1\}^n$ and all PRGs f, if φ is t-multi-point function defined by $\boldsymbol{a}$ and $\boldsymbol{b}$, then*

$$\Pr\left[\mathsf{Eval}^1(sk_1, x) + \mathsf{Eval}^2(sk_2, x) = \varphi(c) \mid sk_1, sk_2 \leftarrow \mathsf{Gen}(\boldsymbol{a}, \boldsymbol{b}, f)\right] = 1.$$

Note that this probability assumes Gen *did not abort.*

Proof. This follows straight from Theorems 9 and 11. □

Theorem 14. *SLAMP-FSS is $(1, t \cdot n \cdot \varepsilon_{\mathsf{PRG}} + (n+1) \cdot \varepsilon_{mat})$-indistinguishable. That is, for all $\varepsilon_{\mathsf{PRG}}$-pseudo random generators f, if P_i is corrupted, $i \in \{1,2\}$, and P_{3-i} is not corrupted, then for all PPT algorithms $\mathcal{A}$*

$$\Pr[\mathit{Game}\ 15(\mathcal{A}) = 1] - \frac{1}{2} \leq t \cdot n \cdot \varepsilon_{\mathsf{PRG}} + (n+1) \cdot \varepsilon_{mat},$$

where

$$\varepsilon_{mat} = \frac{t}{(2^k)^{v-t+1}}.$$

Proof. From Theorem 12 we know that SLAMP-FSS is $(t \cdot n \cdot \varepsilon_{\mathsf{PRG}} + (n+1) \cdot \varepsilon_{mat})$-ROR indistinguishable. Let us assume towards contradiction that SLAMP-FSS is

Algorithm 6: $\mathcal{B}()$

1 $(a_1^0, b_1^0), \ldots, (a_t^0, b_t^0), (a_1^1, b_1^1), \ldots, (a_t^1, b_t^1), \sigma \leftarrow \mathcal{A}()$;
2 $b \xleftarrow{\$} \{0, 1\}$;
3 **return** $(a_1^b, b_1^b), \ldots, (a_t^b, b_t^b)$;

Algorithm 7: $\mathcal{B}(sk_i)$

1 $\bar{b} \leftarrow \mathcal{A}(sk_i, \sigma)$;
2 **return** $[\bar{b} \stackrel{?}{=} b]$;

not $(1, t \cdot n \cdot \varepsilon_{\mathsf{PRG}} + (n+1) \cdot \varepsilon_{mat})$-indistinguishable. Thus, there exists PPT algorithm $\mathcal{A}$ such that

$$\Pr\left[\text{Game } 15(\mathcal{A}) = 1\right] - \frac{1}{2} > t \cdot n \cdot \varepsilon_{\mathsf{PRG}} + (n+1) \cdot \varepsilon_{mat}.$$

Let us define a new adversary $\mathcal{B}$ against ROR indistinguishably in Algorithms 6 and 7:

Without loss of generality, let us assume $i = 1$. Now let us inline $\boldsymbol{B}$ into Game 1 and 3. We get Game 16 and 17. In Game 17, we can delete line 3, because we do not use these values and move line 2 down. We get Game 18.

Let us calculate $|\Pr\left[\text{Game } 1(\mathcal{B}) = 1\right] - \Pr\left[\text{Game } 3(\mathcal{B}) = 1\right]|$. For this notice that Game 16 $\equiv$ Game 15. Therefore,

$$\Pr\left[\text{Game } 1(\mathcal{B}) = 1\right] = \Pr\left[\text{Game } 16(\mathcal{B}) = 1\right] = \Pr\left[\text{Game } 15(\mathcal{A}) = 1\right] > \\ > \frac{1}{2} + t \cdot n \cdot \varepsilon_{\mathsf{PRG}} + (n+1) \cdot \varepsilon_{mat}.$$

Since in Game 18 b is sampled randomly just before the comparison with $\bar{b}$, then the probability that $b = \bar{b}$ must be $\frac{1}{2}$ i.e

$$\Pr\left[\text{Game } 3(\mathcal{B}) = 1\right] = \Pr\left[\text{Game } 18(\mathcal{B}) = 1\right] = \frac{1}{2}.$$

That is

$$|\Pr\left[\text{Game } 1(\mathcal{B}) = 1\right] - \Pr\left[\text{Game } 3(\mathcal{B}) = 1\right]| > t \cdot n \cdot \varepsilon_{\mathsf{PRG}} + (n+1) \cdot \varepsilon_{mat}.$$

We have reached a contradiction. Thus, SLAMP-FSS is $(1, t \cdot n \cdot \varepsilon_{\mathsf{PRG}} + (n+1) \cdot \varepsilon_{mat})$-indistinguishable. □

5.10 Efficiency

Here we give results about the efficiency of our schemes. The detailed results can be found in Appendix C. Let $t_{\mathbb{F}}$ denote the time for one field operation over $\mathbb{F}_{2^k}$, $t_{\mathbb{F}\,vec}$ the time for one operation over vector space $\mathbb{F}_{2^k}^v$ and t_{PRG} the time for one PRG evaluation.

Proposition 1. *Algorithm 1 produces keys with size* $vnk + 2vk + 2nk + k$ *bits.*

Lemma 2. *For* SubGen *to check if a child node* $r||b$ *of an alive node* r *is supposed to be dead or alive, takes* $O(t\lceil \log t \rceil)$ *steps per level.*

Proposition 2. *Algorithm 2 (*SubGen*) is* $O(vt^2 \cdot t_{\mathbb{F}} + t \cdot t_{\mathbb{F}\,vec} + t \cdot t_{\mathsf{PRG}})$. *In fact it takes at most* $2t$ *PRG evaluations.*

Proposition 3. *Algorithm 1 (*Gen*) is* $O(nvt^2 \cdot t_{\mathbb{F}} + nt \cdot t_{\mathsf{PRG}})$. *In fact it takes at most* $2tn$ *PRG evaluations.*

Proposition 4. *Algorithm 4 (*Eval^P*) has time complexity* $\Theta(nvt_{\mathbb{F}} + nt_{\mathsf{PRG}})$. *In fact it takes* n *PRG evaluations.*

Proposition 5. *Full evaluation of SLAMP-FSS requires* $2 \cdot 2^n$ *calls to the PRG.*

Proposition 6. *Full evaluation of SLAMPR-FSS that is given in Appendix B requires* 2^n *calls to the PRG.*

6 Comparison to prior works

In this section we compare our results with some prior works. In Table 1 we can see how the three schemes compare to each other. Parameters t, n, k come from that t-multi-point functions are taken from the function family $\mathcal{F}(\{0,1\}^n \to \mathbb{F}_{2^k})$, $\epsilon > 0$ is a constant used in the batch-code scheme and v is the security parameter.

We compare mainly the key size and the number of PRG calls in various stages. We note that the PRG evaluation is by far the most expensive step and thus we measure that.

We compare our results to the original results of Boyle et al. [BGI16], the two techniques proposed in Boyle et al. [BCGI18] and to Schoppmann et al. [SGRR19].

Concerning key size, we note that other schemes tend to have it somewhat bigger than $O(tvn)$ while our solution is $O(vnk)$ and is in practice dominated by vnk as can be seen from Proposition 1. Thus our solutions are competitive in the area of the key size, with the specific most effective scheme depending on the relations between t, k and other parameters.

We can see that our approach is competitive in the amount of PRG calls in the generation procedure with the exception of the batch code solution. However, the full evaluation of the batch code solution is many times as costly as our solutions, because of the difficulty of finding good batch code parameters.

We note that we estimate the key sizes and the number of PRG calls in the table with big O notation. This is largely due it being difficult to find more precise values to various code-based solutions, as those values depend on the code parametres. We note however, that in those categories, we are not hiding a large constant in our big O, as the key sizes for SLAMP-FSS and SLAMPR-FSSis dominated by the term vnk, and the number of PRG calls in Gen is $2tn$.

Scheme	Key size	Number of PRG calls in Gen	Number of PRG calls in Eval	Number of PRG calls in FullEval	Known-point FSS
Simple [BGI16]	$O(t(vn+k))$	$O(tn)$	tn	$2t2^n$	No
[BCGI18], batch codes	$O(t^{1+\epsilon}(v\lceil n-\log t\rceil+k))$	$O(n)$	-	$O(2^n)$	No
[BCGI18], RSD [1]	$O(t(vn+k))$	$O(tn)$	n	$(1+\lceil\frac{k}{v+2}\rceil)2^n$	No
PBC [SGRR19][2]	$O(t(vn+k))$	$O(tn)$	-	$3\cdot 2^n$	Yes
SLAMP-FSS	$O(vnk)$	$O(tn)$	n	$2\cdot 2^n$	No
SLAMPR-FSS[3]	$O(vnk)$	$O(tn)$	n	2^n	No

[1] The algorithm requires a stronger assumption, thus the t under question should be probably larger, compared to others.
[2] The algorithm has a small $\approx 2^{-40}$ error probability.
[3] The output values are randomly distributed.

Table 1. Multi-point function secret sharing scheme comparison

We note that an issue with our scheme is that in Gen, we require solving n systems of linear equations. However, as explained in the introduction, this boils down to $2n$ private $t \times t$ matrix multiplications and n private matrix-vector multiplications which is not too expensive.

Compared to the RSD solution of [BCGI18], we note that its amount of PRG calls in FullEval is at minimum $2 \cdot 2^k$, thus being tied in this regard with SLAMP-FSS but being twice as costly as SLAMPR-FSS. In the other parametres, the results are similar, although the respective value of t should be larger due to the stronger assumption.

7 Conclusions

We presented a new multi-point function secret sharing schemes SLAMP-FSS and SLAMPR-FSS and proved that they is complete and secure with a generation function that passes with overwhelming probability. The scheme has the biggest advantage over its competitors that in the evaluation process, the number of PRG evaluations needed is much smaller while keeping the other efficiency numbers competitive with the other schemes. Moreover, it is built from simple mathematical constructions, and is conceptually simple. It seems likely that this approach can be combined with other techniques in the future.

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.

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

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.

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.

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 *Advances in Cryptology–CRYPTO 2019: 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2019, Proceedings, Part III 39*, pages 489–518. Springer, 2019.

BCGI18. Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. Compressing vector OLE. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 896–912, 2018.

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.

BGI15. Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 337–367. Springer, 2015.

BGI16. Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1292–1303, 2016.

Bla79. G. R. Blakley. Safeguarding cryptographic keys. In *1979 International Workshop on Managing Requirements Knowledge, MARK 1979, New York, NY, USA, June 4-7, 1979*, pages 313–318. IEEE, 1979.

Car00. Neal L Carothers. *Real analysis*. Cambridge University Press, 2000.

CPS08. Jean-Sébastien Coron, Jacques Patarin, and Yannick Seurin. The random oracle model and the ideal cipher model are equivalent. In *Advances in Cryptology–CRYPTO 2008: 28th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2008. Proceedings 28*, pages 1–20. Springer, 2008.

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.

GI14. Niv Gilboa and Yuval Ishai. Distributed point functions and their applications. In *Advances in Cryptology–EUROCRYPT 2014: 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings 33*, pages 640–658. Springer, 2014.

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.

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

Gra72. Mary W Gray. *Calculus with finite mathematics for social sciences*. Addison-Wesley Publishing Company, Inc., 1972.

GYW+23. Xiaojie Guo, Kang Yang, Xiao Wang, Wenhao Zhang, Xiang Xie, Jiang Zhang, and Zheli Liu. Half-tree: Halving the cost of tree expansion in COT and DPF. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part I*, volume 14004 of *Lecture Notes in Computer Science*, pages 330–362. Springer, 2023.

IEE. What is multiparty computation? `https://digitalprivacy.ieee.org/publications/topics/what-is-multiparty-computation`. Accessed on 14.05.2024.

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.

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

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.

Sha79. Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

A Missing proofs

Time complexity of Gaussian elimination

Theorem 15. *Let $t, v, k \in \mathbb{N}$, $v > t$, $A \in \mathbb{F}_{2^k}^{t \times v}$ and $\boldsymbol{B} \in \mathbb{F}_{2^k}^{t}$. Gaussian elimination on the matrix equation $Ax = \boldsymbol{B}$ takes $O(vt^2)$.*

Proof. Let us describe Gaussian elimination [AR91] line by line and bound the time complexity of each step. Operations on rows are done on the augmented matrix $(A \mid \boldsymbol{B})$.

Start iterating over columns of A from left to right. Let j be the column index. And S be an empty set at the beginning.

1. Search for a row $i \notin S$ with non-zero element in j-th column. It takes at most t field operations.
2. If all elements were 0, then move onto the next column and go back to the previous step.
3. If a non-zero element a was found, find its inverse a^{-1} and add i to set S. This takes one field operation and some constant number of steps.
4. Multiply the elements of this row i with a^{-1} and replace the i-th row with these values. This takes $O(v)$ field operations.
5. For each row $i' \notin S$ take the j-th element b. Multiply each element of the i-th row with b and subtract these values from the elements of the i'-th row and replace the i'-th row with the result. This takes $O(tv)$ field operations.
6. Repeat for column $j+1$ until $j = v$ or $|S| = t$.

After this the matrix is in the row-echelon form. The worst case scenario is if first $v - t$ columns are all zero columns and the last t are linearly independent of each other. In this case, we do $O(t(v-t)) = O(tv)$ field operations to check the zero columns and $O(t(t+1+tv+1)) = O(vt^2)$ field operations to get the row-echelon form.

To get to the reduced row-echelon form, we do similar triangular process, but look at column indices in S. This is similar enough that it also takes $O(vt^2)$ field operations. □

A.1 Theorem 12

Theorem 16. *For all $\varepsilon_{\mathsf{PRG}}$-PRG $f : \mathbb{F}_{2^k} \to \mathbb{F}_{2^k}^{v+1}$:*
SLAMP-FSS is $(t \cdot n \cdot \varepsilon_{\mathsf{PRG}} + (n+1) \cdot \varepsilon_{mat})$-ROR-indistinguishable, where

$$\varepsilon_{mat} = \frac{t}{(2^k)^{v-t+1}}.$$

Proof. In this proof, we color changes in code magenta.
Without loss of generality, let us assume that adversary $\mathcal{A}$ is playing as P_2. Also, let us assume we have $\varepsilon_{\mathsf{PRG}}$-PRG $f : \mathbb{F}_{2^k} \to \mathbb{F}_{2^k}^{v+1}$. This means that for every PPT adversary $\mathcal{B}$

$$|\Pr[\text{Game } 4(\mathcal{B}) = 1] - \Pr[\text{Game } 5(\mathcal{B}) = 1]| \leq \varepsilon_{\mathsf{PRG}}.$$

Now let us look at Game 2 and the first occurrence of the line (1)

$$([X_{r||b}]_1, [\tau_{r||b}]_1) \leftarrow f(\langle [X_r]_1, \boldsymbol{d}_{i-1} \rangle + [\tau_r]_1 \cdot w_{i,b}). \tag{1}$$

Since $\epsilon \in R_0$, then at least one of the corresponding node's children must be in R_1. Let us denote this child by b. Now let us look at $\langle [X_\epsilon]_1, \boldsymbol{d}_0 \rangle + [\tau_\epsilon]_1 \cdot w_{1,b}$.

Since $w_{1,b} \neq 0$ and $[\tau_\epsilon]_1$ is uniform over $\mathbb{F}_{2^k}$, then $[\tau_\epsilon]_1 \cdot w_{1,b}$ is also uniform over $\mathbb{F}_{2^k}$. From this we can also conclude that $\langle [X_\epsilon]_1, \boldsymbol{d}_0 \rangle + [\tau_\epsilon]_1 \cdot w_{1,b}$ is also uniform over $\mathbb{F}_{2^k}$. Notice that $\mathcal{A}$ does not know this value, because they do not know $[\tau_\epsilon]_1$. Even if they know all the other parts of this expression $[\tau_\epsilon]_1$ hides it.

Therefore, we have reached Game 4. We can replace the first occurrence of line (1) with

$$([X_{r||b}]_1, [\tau_{r||b}]_1) \xleftarrow{\$} \mathbb{F}_{2^k}^{v+1},$$

and get a new game called Game 2-1, which differs by only this line and is $\varepsilon_{\mathsf{PRG}}$-close to Game 2. Repeating this procedure at every occurrence of line (1) or in other words at every alive node, we can replace all such lines and reach Game 2-t', which is $t \cdot n \cdot \varepsilon_{\mathsf{PRG}}$-close to Game 2. Let us write out Game 2-t' and rename it to Game 6. This can be further simplified, because $X_{r||b}$ and $\tau_{r||b}$ on lines 30 and 31 of SubGen are now also uniformly random. We get the new game Game 7. We can do the sampling right before we need sampled values and get Game 8, which turns into Game 9, because the aim of the sampling is to sample random matrices and vectors. **Note** that the symbol "..." denotes the lines that have not been changed and are the same as in Game 2.

Since $\boldsymbol{d}_{i-1}$ 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 $\boldsymbol{d} \in \mathbb{F}_{2^k}^v$. Let us show that the statistical distance between Game 10 and Game 12 is small. For this let us consider Game 11. This game is like Game 10, but it returns $\perp$, if $A\boldsymbol{d} = \boldsymbol{B}$ is not solvable. In fact because of Theorem 3, Game 10 returns $\perp$ if and only if Game 11 returns $\perp$. Thus,

$$SD(\text{Game } 10, \text{Game } 12) = SD(\text{Game } 11, \text{Game } 12).$$

Let us calculate $SD(\text{Game } 11, \text{Game } 12)$:

$$\begin{aligned}
&SD(\text{Game } 11, \text{Game } 12) = \\
&= \frac{1}{2} \sum_{\alpha \in \mathbb{F}_{2^k}^v \cup \{\perp\}} |\Pr[\text{Game } 11 = \alpha] - \Pr[\text{Game } 12 = \alpha]| = \\
&= \frac{1}{2} \sum_{\alpha \in \mathbb{F}_{2^k}^v} \Bigg| \sum_{A \in \mathbb{F}_{2^k}^{|R_{i-1}| \times v}} \sum_{\boldsymbol{B} \in \mathbb{F}_{2^k}^{|R_{i-1}|}} \Pr[A] \Pr[\boldsymbol{B} \mid A] \cdot \\
&\qquad \cdot \Pr_{\substack{\boldsymbol{d} \xleftarrow{\$} \text{solutions to} \\ A\boldsymbol{d} = \boldsymbol{B}}} [\boldsymbol{d} = \alpha \mid \boldsymbol{B} \cap A] - \frac{1}{(2^k)^v} \Bigg| + \\
&+ \frac{1}{2} \sum_{A \in \mathbb{F}_{2^k}^{|R_{i-1}| \times v}} \sum_{\boldsymbol{B} \in \mathbb{F}_{2^k}^{|R_{i-1}|}} \Pr[A] \Pr[\boldsymbol{B} \mid A] \Pr_{\substack{\boldsymbol{d} \xleftarrow{\$} \text{solutions to} \\ A\boldsymbol{d} = \boldsymbol{B}}} [\boldsymbol{d} = \perp \mid \boldsymbol{B} \cap A].
\end{aligned}$$

Since $\boldsymbol{B}$ is independent of A, then

$$\Pr[A] = \frac{1}{(2^k)^{|R_{i-1}| \cdot v}}, \quad \Pr[\boldsymbol{B}] = \frac{1}{(2^k)^{|R_{i-1}|}}.$$

Now we can see that $\boldsymbol{d} = \perp$ iff $\mathsf{rank}(A) < \mathsf{rank}(A|\boldsymbol{B})$. The analysis for this probability can be seen in Section 5.1. Therefore, the second summand is

$$\frac{1}{2} \cdot \sum_{A \in \mathbb{F}_{2^k}^{|R_{i-1}| \times v}} \sum_{\boldsymbol{B} \in \mathbb{F}_{2^k}^{|R_{i-1}|}} \Pr[A] \Pr[\boldsymbol{B}] \Pr_{\substack{\boldsymbol{d} \xleftarrow{\$} \text{solutions to} \\ A\boldsymbol{d} = \boldsymbol{B}}} [\boldsymbol{d} = \perp \mid \boldsymbol{B} \cap A] =$$

$$= \frac{1}{2} \cdot \sum_{A \in \mathbb{F}_{2^k}^{|R_{i-1}| \times v}} \sum_{\substack{\boldsymbol{B} \in \mathbb{F}_{2^k}^{|R_{i-1}|} \\ \mathsf{rank}(A) < \mathsf{rank}(A|\boldsymbol{B})}} \Pr[A] \Pr[\boldsymbol{B}] = \frac{1}{2} \cdot \frac{\#\text{ of } A \text{ and } \boldsymbol{B} \text{ such that } \mathsf{rank}(A) < \mathsf{rank}(A|\boldsymbol{B})}{\#\text{ of } A \text{ and } \boldsymbol{B}} =$$

$$= \frac{1}{2} \Pr[A\boldsymbol{d} = \boldsymbol{B} \text{ is not solvable}].$$

Let us fix α and look at the first summand. More precisely let us calculate

$$\sum_{A \in \mathbb{F}_{2^k}^{|R_{i-1}| \times v}} \sum_{\boldsymbol{B} \in \mathbb{F}_{2^k}^{|R_{i-1}|}} \Pr[A] \Pr[\boldsymbol{B}] \Pr_{\substack{\boldsymbol{d} \xleftarrow{\$} \text{solutions to} \\ A\boldsymbol{d} = \boldsymbol{B}}} [\boldsymbol{d} = \alpha \mid \boldsymbol{B} \cap A].$$

If $A\alpha \neq \boldsymbol{B}$, then we know that $\Pr_{\substack{\boldsymbol{d} \xleftarrow{\$} \text{solutions to} \\ A\boldsymbol{d} = \boldsymbol{B}}} [\boldsymbol{d} = \alpha \mid \boldsymbol{B} \cap A] = 0$. If $A\alpha = \boldsymbol{B}$, then $A\boldsymbol{d} = \boldsymbol{B}$ is solvable and there are $(2^k)^{v - \mathsf{rank}(A)}$ solutions to it. The probability of uniformly sampling $\boldsymbol{d}$ such that $\boldsymbol{d} = \alpha$ is thus

$$\Pr_{\substack{\boldsymbol{d} \xleftarrow{\$} \text{solutions to} \\ A\boldsymbol{d} = \boldsymbol{B}}} [\boldsymbol{d} = \alpha \mid \boldsymbol{B} \cap A] = \frac{1}{(2^k)^{v - \mathsf{rank}(A)}}.$$

We get

$$\sum_{A \in \mathbb{F}_{2^k}^{|R_{i-1}| \times v}} \sum_{\boldsymbol{B} \in \mathbb{F}_{2^k}^{|R_{i-1}|}} \Pr[A] \Pr[\boldsymbol{B}] \Pr_{\substack{\boldsymbol{d} \xleftarrow{\$} \text{solutions to} \\ A\boldsymbol{d} = \boldsymbol{B}}} [\boldsymbol{d} = \alpha \mid \boldsymbol{B} \cap A] =$$

$$= \sum_{A \in \mathbb{F}_{2^k}^{|R_{i-1}| \times v}} \sum_{\substack{\boldsymbol{B} \in \mathbb{F}_{2^k}^{|R_{i-1}|} \\ A\alpha = \boldsymbol{B}}} \frac{1}{(2^k)^{|R_{i-1}|(v+1)}} \cdot \frac{1}{(2^k)^{v - \mathsf{rank}(A)}} =$$

$$= \sum_{j=0}^{|R_{i-1}|} \left(\sum_{\substack{A \in \mathbb{F}_{2^k}^{|R_{i-1}| \times v} \\ \mathsf{rank}(A) = j}} \sum_{\substack{\boldsymbol{B} \in \mathbb{F}_{2^k}^{|R_{i-1}|} \\ A\alpha = \boldsymbol{B}}} \frac{1}{(2^k)^{|R_{i-1}|(v+1)}} \cdot \frac{1}{(2^k)^{v-j}} \right).$$

If A and α are fixed, then there is exactly one $\boldsymbol{B}$ for which $A\alpha = \boldsymbol{B}$. Therefore,

$$\sum_{j=0}^{|R_{i-1}|} \left(\sum_{\substack{A \in \mathbb{F}_{2^k}^{|R_{i-1}| \times v} \\ \mathsf{rank}(A) = j}} \sum_{\substack{\boldsymbol{B} \in \mathbb{F}_{2^k}^{|R_{i-1}|} \\ A\alpha = \boldsymbol{B}}} \frac{1}{(2^k)^{|R_{i-1}|(v+1)}} \cdot \frac{1}{(2^k)^{v-j}} \right) =$$

$$= \sum_{j=0}^{|R_{i-1}|} \left(\sum_{\substack{A \in \mathbb{F}_{2^k}^{|R_{i-1}| \times v} \\ \mathsf{rank}(A)=j}} \frac{1}{(2^k)^{|R_{i-1}|(v+1)}} \cdot \frac{1}{(2^k)^{v-j}} \right) =$$

$$= \sum_{j=0}^{|R_{i-1}|} \left(\frac{\left|\{A \mid A \in \mathbb{F}_{2^k}^{|R_{i-1}| \times v} \wedge \mathsf{rank}(A) = j\}\right|}{(2^k)^{|R_{i-1}|(v+1)} \cdot (2^k)^{v-j}} \right) =$$

$$= \frac{1}{(2^k)^v} \sum_{j=0}^{|R_{i-1}|} \left((2^k)^j \cdot \frac{\left|\{A \mid A \in \mathbb{F}_{2^k}^{|R_{i-1}| \times v} \wedge \mathsf{rank}(A) = j\}\right|}{(2^k)^{|R_{i-1}|(v+1)}} \right) =$$

$$= \frac{1}{(2^k)^v} \Pr\left[A\boldsymbol{d} = \boldsymbol{B} \text{ is solvable}\right],$$

where the last equality comes from Subsection 5.1. Replacing this result into the first summand we get

$$\frac{1}{2} \sum_{\alpha \in \mathbb{F}_{2^k}^v} \left| \frac{1}{(2^k)^v} \Pr\left[A\boldsymbol{d} = \boldsymbol{B} \text{ is solvable}\right] - \frac{1}{(2^k)^v} \right| =$$

$$= \frac{1}{2} \cdot \frac{1}{(2^k)^v} \sum_{\alpha \in \mathbb{F}_{2^k}^v} \left| \Pr\left[A\boldsymbol{d} = \boldsymbol{B} \text{ is solvable}\right] - 1 \right| =$$

$$= \frac{1}{2} \cdot \frac{1}{(2^k)^v} \cdot (2^k)^v \cdot (1 - \Pr\left[A\boldsymbol{d} = \boldsymbol{B} \text{ is solvable}\right]) =$$

$$= \frac{1}{2} \cdot (1 - \Pr\left[A\boldsymbol{d} = \boldsymbol{B} \text{ is solvable}\right]).$$

We can now calculate the statistical distance

$$SD(\text{Game } 11, \text{Game } 12) =$$

$$= \frac{1}{2} (1 - \Pr\left[A\boldsymbol{d} = \boldsymbol{B} \text{ is solvable}\right]) + \frac{1}{2} \Pr\left[A\boldsymbol{d} = \boldsymbol{B} \text{ is not solvable}\right] =$$

$$= 1 - \Pr\left[A\boldsymbol{d} = \boldsymbol{B} \text{ is solvable}\right] \leq \frac{t}{(2^k)^{v-t+1}}.$$

Thus, the statistical distance between Game 10 and Game 12 is at most

$$\varepsilon_{mat} := \frac{t}{(2^k)^{v-t+1}}.$$

Now we can replace the linear equations in Game 9 with random $\boldsymbol{d}$ and get Game 13 that is $k \cdot \varepsilon_{mat}$ close to Game 9.

The same analysis can be done for finding $\boldsymbol{g}$ and thus we can replace it as well and get Game 14 that is ε_{mat} close to Game 13. We can also notice that Game 14 $\equiv$ Game 3. This means we have found the computational distance between Game 3 and Game 2:

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

If $\mathcal{A}$ plays the role of P_1, then the proof is analogous. □

A.2 Games

In the following security games Game 1, 2, the notation $\overset{P_1}{\leftarrow}$ denotes the output that is given to P_1 and $\overset{P_2}{\leftarrow}$ denotes the output that is given to P_2.

Game 1:
Input : $\mathcal{A}$ - the adversarial algorithm
$(a_1, b_1), \ldots, (a_t, b_t) \leftarrow \mathcal{A}()$;
$sk_1 \overset{P_1}{\leftarrow} \mathsf{Gen}((a_1, b_1), \ldots, (a_t, b_t), f)$;
return $\mathcal{A}(sk_1)$;

Game 2:
Input : $\mathcal{A}$ - the adversarial algorithm
$(a_1, b_1), \ldots, (a_t, b_t) \leftarrow \mathcal{A}()$;
$sk_2 \overset{P_2}{\leftarrow} \mathsf{Gen}((a_1, b_1), \ldots, (a_t, b_t), f)$;
return $\mathcal{A}(sk_2)$;

Game 3:
Input : $\mathcal{A}$ - the adversarial algorithm
$(a_1, b_1), \ldots, (a_t, b_t) \leftarrow \mathcal{A}()$;
$X_\epsilon \overset{\$}{\leftarrow} \mathbb{F}_{2^k}^v$;
$\tau_\epsilon \overset{\$}{\leftarrow} \mathbb{F}_{2^k}$;
for $i = 1$ **to** n **do**
  $w_{i,0} \overset{\$}{\leftarrow} \mathbb{F}_{2^k} \setminus \{0\}$;
  $w_{i,1} \overset{\$}{\leftarrow} \mathbb{F}_{2^k} \setminus \{0, w_{i,0}\}$;
  $\boldsymbol{d}_{i-1} \overset{\$}{\leftarrow} \mathbb{F}_{2^k}^v$;
$\boldsymbol{g} \overset{\$}{\leftarrow} \mathbb{F}_{2^k}^v$;
$sk \leftarrow X_\epsilon, \tau_\epsilon, \{w_{i,0}\}_{i=1}^n, \{w_{i,1}\}_{i=1}^n, \{\boldsymbol{d}_i\}_{i=0}^{n-1}, \boldsymbol{g}$;
return $\mathcal{A}(sk)$;

Game 4:
Input : $\mathcal{B}$ - the adversarial algorithm
$x \overset{\$}{\leftarrow} \mathbb{F}_{2^k}$;
$y \leftarrow f(x)$;
return $\mathcal{B}(y)$;

Game 5:
Input : $\mathcal{B}$ - the adversarial algorithm
$y \overset{\$}{\leftarrow} \mathbb{F}_{2^k}^{v+1}$;
return $\mathcal{B}(y)$;

Game 6:
Input : $\mathcal{A}$ - the adversarial algorithm
$(a_1, b_1), \ldots, (a_t, b_t) \leftarrow \mathcal{A}()$;
$\cdots$;
for $r||b \in R_i$ **do**
  $([X_{r||b}]_1, [\tau_{r||b}]_1) \xleftarrow{\$} \mathbb{F}_{2^k}^{v+1}$;
  $([X_{r||b}]_2, [\tau_{r||b}]_2) \leftarrow f(\langle [X_r]_2, \boldsymbol{d}_{i-1} \rangle + [\tau_r]_2 \cdot w_{i,b})$;
  $X_{r||b} \leftarrow [X_{r||b}]_1 + [X_{r||b}]_2$;
  $\tau_{r||b} \leftarrow [\tau_{r||b}]_1 + [\tau_{r||b}]_2$;
$\cdots$;
return $\mathcal{A}(sk_2)$;

Game 7:
$\cdots$;
for $r||b \in R_i$ **do**
  $(X_{r||b}, \tau_{r||b}) \xleftarrow{\$} \mathbb{F}_{2^k}^{v+1}$;
$\cdots$;

Game 10:
$A \xleftarrow{\$} \mathbb{F}_{2^k}^{|R_{i-1}| \times v}$;
$\boldsymbol{B} \xleftarrow{\$} \mathbb{F}_{2^k}^{|R_{i-1}|}$;
Solve $A\boldsymbol{d} = \boldsymbol{B}$ and choose a $\boldsymbol{d}$;
return $\boldsymbol{d}$;

Game 11:
$A \xleftarrow{\$} \mathbb{F}_{2^k}^{|R_{i-1}| \times v}$;
$\boldsymbol{B} \xleftarrow{\$} \mathbb{F}_{2^k}^{|R_{i-1}|}$;
if $\mathsf{rank}(A) < \mathsf{rank}(A|\boldsymbol{B})$ **then**
  return $\perp$;
Solve $A\boldsymbol{d} = \boldsymbol{B}$ and choose a $\boldsymbol{d}$;
return $\boldsymbol{d}$;

Game 12:
$\boldsymbol{d} \xleftarrow{\$} \mathbb{F}_{2^k}^{v}$;
return $\boldsymbol{d}$;

Game 8:

Input : $\mathcal{A}$ - the adversarial algorithm

$(a_1, b_1), \ldots, (a_t, b_t) \leftarrow \mathcal{A}()$;

$[X_\epsilon]_2 \xleftarrow{\$} \mathbb{F}_{2^k}^v$;

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

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

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

  $\cdots$;

  Let A be a $|R_{i-1}| \times v$ matrix of zeroes;

  Let $\boldsymbol{B}$ be a zero column vector of length $|R_{i-1}|$;

  $j_i \leftarrow 1$;

  for $r||b \in R'_i$ **do**

    $(X_r, \tau_r) \xleftarrow{\$} \mathbb{F}_{2^k}^{v+1}$;

    Set the j_i-th row of A to be X_r;

    Set the j_i-th element of $\boldsymbol{B}$ to be $\tau_r \cdot w_{i,b}$;

    $j_i \leftarrow j_i + 1$;

  for $r \in \widehat{R}_{i-1}$ **do**

    $w_{r,2} \xleftarrow{\$} \mathbb{F}_{2^k} \setminus \{w_{i,0}, w_{i,1}\}$;

    $(X_r, \tau_r) \xleftarrow{\$} \mathbb{F}_{2^k}^{v+1}$;

    Set the j_i-th row of A to be X_r;

    Set the j_i-th element of $\boldsymbol{B}$ to be $\tau_r \cdot w_{r,2}$;

    $j_i \leftarrow j_i + 1$;

  Solve $A\boldsymbol{d}_{i-1} = \boldsymbol{B}$ and sample $\boldsymbol{d}_{i-1}$ from the solution space;

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

  $(X_{a_i}, \tau_{a_i}) \xleftarrow{\$} \mathbb{F}_{2^k}^{v+1}$;

$$A \leftarrow \begin{pmatrix} X_{a_1} \\ \vdots \\ X_{a_t} \end{pmatrix};$$

$\boldsymbol{B} \leftarrow (b_1 + \tau_{a_1}, \cdots, b_t + \tau_{a_t})^T$;

Solve $A\boldsymbol{g} = \boldsymbol{B}$ and sample $\boldsymbol{g}$ from the solution space;

return $\mathcal{A}([X_\epsilon]_2, [\tau_\epsilon]_2, \{w_{i,0}\}_{i=1}^n, \{w_{i,1}\}_{i=1}^n, \{\boldsymbol{d}_i\}_{i=0}^{n-1}, \boldsymbol{g})$;

Game 9:

Input : $\mathcal{A}$ - the adversarial algorithm

$(a_1, b_1), \ldots, (a_t, b_t) \leftarrow \mathcal{A}()$;

$[X_\epsilon]_2 \xleftarrow{\$} \mathbb{F}_{2^k}^v$;

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

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

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 $\exists a_j$ *such that* a_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**

      $\widehat{R}_{i-1} \leftarrow \widehat{R}_{i-1} \cup \{r\}$;

  $A \xleftarrow{\$} \mathbb{F}_{2^k}^{|R_{i-1}| \times v}$;

  $\boldsymbol{B} \xleftarrow{\$} \mathbb{F}_{2^k}^{|R_{i-1}|}$;

  Solve $A\boldsymbol{d}_{i-1} = \boldsymbol{B}$ and sample $\boldsymbol{d}_{i-1}$ from the solution space;

$A \xleftarrow{\$} \mathbb{F}_{2^k}^{t \times v}$;

$\boldsymbol{B} \xleftarrow{\$} \mathbb{F}_{2^k}^{t}$;

Solve $A\boldsymbol{g} = \boldsymbol{B}$ and sample $\boldsymbol{g}$ from the solution space;

return $\mathcal{A}([X_\epsilon]_2, [\tau_\epsilon]_2, \{w_{i,0}\}_{i=1}^n, \{w_{i,1}\}_{i=1}^n, \{\boldsymbol{d}_i\}_{i=0}^{n-1}, \boldsymbol{g})$;

Game 13:
Input : $\mathcal{A}$ - the adversarial algorithm

$(a_1, b_1), \ldots, (a_t, b_t) \leftarrow \mathcal{A}()$;
$[X_\epsilon]_2 \xleftarrow{\$} \mathbb{F}_{2^k}^v$;
$[\tau_\epsilon]_2 \xleftarrow{\$} \mathbb{F}_{2^k}$;
for $i = 1$ **to** n **do**
  $w_{i,0} \xleftarrow{\$} \mathbb{F}_{2^k} \setminus \{0\}$;
  $w_{i,1} \xleftarrow{\$} \mathbb{F}_{2^k} \setminus \{0, w_{i,0}\}$;
  $\boldsymbol{d}_{i-1} \xleftarrow{\$} \mathbb{F}_{2^k}^v$;
$A \xleftarrow{\$} \mathbb{F}_{2^k}^{t \times v}$;
$\boldsymbol{B} \xleftarrow{\$} \mathbb{F}_{2^k}^t$;
Solve $A\boldsymbol{g} = \boldsymbol{B}$ and sample $\boldsymbol{g}$ from the solution space;
return $\mathcal{A}([X_\epsilon]_2, [\tau_\epsilon]_2, \{w_{i,0}\}_{i=1}^n, \{w_{i,1}\}_{i=1}^n, \{\boldsymbol{d}_i\}_{i=0}^{n-1}, \boldsymbol{g})$;

Game 14:
Input : $\mathcal{A}$ - the adversarial algorithm

$(a_1, b_1), \ldots, (a_t, b_t) \leftarrow \mathcal{A}()$;
$[X_\epsilon]_2 \xleftarrow{\$} \mathbb{F}_{2^k}^v$;
$[\tau_\epsilon]_2 \xleftarrow{\$} \mathbb{F}_{2^k}$;
for $i = 1$ **to** n **do**
  $w_{i,0} \xleftarrow{\$} \mathbb{F}_{2^k} \setminus \{0\}$;
  $w_{i,1} \xleftarrow{\$} \mathbb{F}_{2^k} \setminus \{0, w_{i,0}\}$;
  $\boldsymbol{d}_{i-1} \xleftarrow{\$} \mathbb{F}_{2^k}^v$;
$\boldsymbol{g} \xleftarrow{\$} \mathbb{F}_{2^k}^v$;
return $\mathcal{A}([X_\epsilon]_2, [\tau_\epsilon]_2, \{w_{i,0}\}_{i=1}^n, \{w_{i,1}\}_{i=1}^n, \{\boldsymbol{d}_i\}_{i=0}^{n-1}, \boldsymbol{g})$;

Game 15:
Input : $\mathcal{A}$ - the adversarial algorithm

$(a_1^0, b_1^0), \ldots, (a_t^0, b_t^0), (a_1^1, b_1^1), \ldots, (a_t^1, b_t^1), \sigma \leftarrow \mathcal{A}()$;
$b \xleftarrow{\$} \{0, 1\}$;
$sk_1, sk_2 \leftarrow \mathsf{Gen}((a_1^b, b_1^b), \ldots, (a_t^b, b_t^b), f)$;
$\bar{b} \leftarrow \mathcal{A}(sk_i, \sigma)$;
return $[\bar{b} \stackrel{?}{=} b]$;

Game 16:
Input : $\mathcal{A}$ - the adversarial algorithm

$(a_1^0, b_1^0), \ldots, (a_t^0, b_t^0), (a_1^1, b_1^1), \ldots, (a_t^1, b_t^1), \sigma \leftarrow \mathcal{A}()$;
$b \xleftarrow{\$} \{0, 1\}$;
$sk_1 \xleftarrow{P_1} \mathsf{Gen}((a_1^b, b_1^b), \ldots, (a_t^b, b_t^b), f)$;
$\bar{b} \leftarrow \mathcal{A}(sk_1, \sigma)$;
return $[\bar{b} \stackrel{?}{=} b]$;

Game 17:
Input : $\mathcal{A}$ - the adversarial algorithm

$(a_1^0, b_1^0), \ldots, (a_t^0, b_t^0), (a_1^1, b_1^1), \ldots, (a_t^1, b_t^1), \sigma \leftarrow \mathcal{A}()$;
$b \xleftarrow{\$} \{0, 1\}$;
$(a_1, b_1), \ldots, (a_t, b_t) \leftarrow (a_1^b, b_1^b), \ldots, (a_t^b, b_t^b)$;
$X_\epsilon \xleftarrow{\$} \mathbb{F}_{2^k}^v$;
$\tau_\epsilon \xleftarrow{\$} \mathbb{F}_{2^k}$;
for $i = 1$ **to** n **do**
  $w_{i,0} \xleftarrow{\$} \mathbb{F}_{2^k} \setminus \{0\}$;
  $w_{i,1} \xleftarrow{\$} \mathbb{F}_{2^k} \setminus \{0, w_{i,0}\}$;
  $\boldsymbol{d}_{i-1} \xleftarrow{\$} \mathbb{F}_{2^k}^v$;
$\boldsymbol{g} \xleftarrow{\$} \mathbb{F}_{2^k}^v$;
$sk \leftarrow X_\epsilon, \tau_\epsilon, \{w_{i,0}\}_{i=1}^n, \{w_{i,1}\}_{i=1}^n, \{\boldsymbol{d}_i\}_{i=0}^{n-1}, \boldsymbol{g}$;
$\bar{b} \leftarrow \mathcal{A}(sk, \sigma)$;
return $[\bar{b} \stackrel{?}{=} b]$;

Game 18:
Input : $\mathcal{A}$ - the adversarial algorithm

$(a_1^0, b_1^0), \ldots, (a_t^0, b_t^0), (a_1^1, b_1^1), \ldots, (a_t^1, b_t^1), \sigma \leftarrow \mathcal{A}()$;
$X_\epsilon \xleftarrow{\$} \mathbb{F}_{2^k}^v$;
$\tau_\epsilon \xleftarrow{\$} \mathbb{F}_{2^k}$;
for $i = 1$ **to** n **do**
  $w_{i,0} \xleftarrow{\$} \mathbb{F}_{2^k} \setminus \{0\}$;
  $w_{i,1} \xleftarrow{\$} \mathbb{F}_{2^k} \setminus \{0, w_{i,0}\}$;
  $\boldsymbol{d}_{i-1} \xleftarrow{\$} \mathbb{F}_{2^k}^v$;
$\boldsymbol{g} \xleftarrow{\$} \mathbb{F}_{2^k}^v$;
$sk \leftarrow X_\epsilon, \tau_\epsilon, \{w_{i,0}\}_{i=1}^n, \{w_{i,1}\}_{i=1}^n, \{\boldsymbol{d}_i\}_{i=0}^{n-1}, \boldsymbol{g}$;
$\bar{b} \leftarrow \mathcal{A}(sk, \sigma)$;
$b \xleftarrow{\$} \{0, 1\}$;
return $[\bar{b} \stackrel{?}{=} b]$;

B SLAMPR-FSS

Here we present another version of our scheme along with its security and correctness proofs. This is very similar to the algorithm described in the main body with two differences. First, here the values b_i will be random, which is sufficient for many applications. Secondly, in the iteration, the last evaluation of SubGen is omitted. This is a small difference when evaluating the scheme just once, but will make the full evaluation of the scheme twice as fast.

The scheme is presented in the Algorithms 8,9,10 and 11. More precisely, Algorithm 8 will describe the key generation algorithm. It is very similar to

Algorithm 8 but solving the last system of equations is omitted, as is the last call to the PRG. To model this we have defined the algorithm 9 which is identical to the primitive SubGen, except at the last iteration, it does not call the PRG on the values $\langle [X_r]_P, \boldsymbol{d}_{i-1} \rangle + [\tau_r]_P \cdot w_{i,b}$ and does not output any new $[X_{r\|b}]_P$ and $[\tau_{r\|b}]_P$, as these values will not be needed any more. The Algorithm 10 is similar to Algorithm 4, except that for $i = n$, we do not call the PRG on the value $\langle [X_r]_P, \boldsymbol{d}_{i-1} \rangle + [\tau_r]_P \cdot w_{i,x^n}$, but simply output it.

We will see that this modified scheme satisfies the necessary properties mutatis mutandis.

The altered requirements are, roughly speaking that a single evaluating party still does not know the values of the indices $\{a_i\}_{i=1}^t$, that the function evaluates to zero on inputs that are not equal to any of the a_i and that it evaluates to a nonzero value at the a_i with an overwhelming probability. In addition, we also will show that the values obtained by evaluating at a_i will be pseudorandom.

Algorithm 8: Gen – Key generator for the scheme SLAMPR-FSS

Input : $\{(a_1, \ldots, a_t)\}, f$

$[X_\epsilon]_1 \xleftarrow{\$} \mathbb{F}_{2^k}^v$;
$[X_\epsilon]_2 \xleftarrow{\$} \mathbb{F}_{2^k}^v \setminus \{[X_\epsilon]_1\}$;
$X_\epsilon \leftarrow [X_\epsilon]_1 + [X_\epsilon]_2$;
$[\tau_\epsilon]_1, [\tau_\epsilon]_2 \xleftarrow{\$} \mathbb{F}_{2^k}$;
$\tau_\epsilon \leftarrow [\tau_\epsilon]_1 + [\tau_\epsilon]_2$;
$R_0 \leftarrow \{\epsilon\}$, where ϵ is an empty string;
for $i = 1$ **to** $n - 1$ **do**
  $w_{i,0}, w_{i,1}, \boldsymbol{d}_{i-1}, R_i \leftarrow$ SubGen();
$w_{n,0}, w_{n,1}, \boldsymbol{d}_{n-1}, R_n \leftarrow$ SubGenPrim();
$sk_1 \leftarrow [X_\epsilon]_1, [\tau_\epsilon]_1, \{w_{i,0}\}_{i=1}^n, \{w_{i,1}\}_{i=1}^n, \{\boldsymbol{d}_i\}_{i=0}^{n-1}$;
$sk_2 \leftarrow [X_\epsilon]_2, [\tau_\epsilon]_2, \{w_{i,0}\}_{i=1}^n, \{w_{i,1}\}_{i=1}^n, \{\boldsymbol{d}_i\}_{i=0}^{n-1}$;
Give to P_1 the key sk_1;
Give to P_2 the key sk_2;

Let us consider the properties outlined in Section 5. We note that as GenPrim algorithm coincides with Gen until the calling of SubGenPrim, which in turn coincides with SubGen, except for not including the last five lines before the **return** statement. Thus we can assume that the properties in hand hold until SubGenPrim is called. Thus we have that for all $r \in R_{n-1}$, we have that $\tau_r \neq 0$ with probability no smaller than

$$\left((1 - \frac{t}{(2^k)^{v-t+1}}) \cdot (1 - \varepsilon_{col})^t \right)^{n-1}$$

by Theorem 8. We also have that if a path r of length $n-1$ it holds that $r \notin R_{n-1}$, then $(X_r, \tau_r) = 0$, by Theorem 7.

Algorithm 9: SubGenPrim – Subroutine for Gen(Algorithm 1)

Input : The state of Gen

Output : $w_{i,0}, w_{i,1}, \boldsymbol{d}_{i-1}, R_i$

$w_{i,0} \xleftarrow{\$} \mathbb{F}_{2^k} \setminus \{0\}$;
$w_{i,1} \xleftarrow{\$} \mathbb{F}_{2^k} \setminus \{0, w_{i,0}\}$;
$R_i \leftarrow \emptyset$;
$R'_i \leftarrow \emptyset$;
$\widehat{R}_{i-1} \leftarrow \emptyset$;
for $r \in R_{i-1}$ **do**
  for $b \in \{0,1\}$ **do**
    if $\exists a_j$ *such that* a_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**
    $\widehat{R}_{i-1} \leftarrow \widehat{R}_{i-1} \cup \{r\}$;
Let A be a $|R_{i-1}| \times v$ matrix of zeroes;
Let $\boldsymbol{B}$ be a zero column vector of length $|R_{i-1}|$;
$j_i \leftarrow 1$;
for $r||b \in R'_i$ **do**
  Set the j_i-th row of A to be X_r;
  Set the j_i-th element of $\boldsymbol{B}$ to be $\tau_r \cdot w_{i,b}$;
  $j_i \leftarrow j_i + 1$;
for $r \in \widehat{R}_{i-1}$ **do**
  $w_{r,2} \xleftarrow{\$} \mathbb{F}_{2^k} \setminus \{w_{i,0}, w_{i,1}\}$;
  Set the j_i-th row of A to be X_r;
  Set the j_i-th element of $\boldsymbol{B}$ to be $\tau_r \cdot w_{r,2}$;
  $j_i \leftarrow j_i + 1$;
Solve $A\boldsymbol{d}_{i-1} = \boldsymbol{B}$ and sample $\boldsymbol{d}_{i-1}$ from the solution space;
return $w_{i,0}, w_{i,1}, \boldsymbol{d}_{i-1}, R_i$;

Algorithm 10: Eval^P – Another way to define Eval^P

Input : $sk_P, f, x = x^1||\dots||x^n$

$[X_\epsilon]_P, [\tau_\epsilon]_P, \{w_{i,0}\}_{i=1}^n, \{w_{i,1}\}_{i=1}^n, \{\boldsymbol{d}_i\}_{i=0}^{n-1}, \boldsymbol{g} \leftarrow sk_P$;
$([\bar{X}_0]_P, [\bar{\tau}_0]_P) \leftarrow ([X_\epsilon]_P, [\tau_\epsilon]_P)$;
for $i = 1$ **to** $n-1$ **do**
  $([\bar{X}_i]_P, [\bar{\tau}_i]_P) \leftarrow \mathsf{Eval}_i^P([\bar{X}_{i-1}]_P, [\bar{\tau}_{i-1}]_P, w_{i,0}, w_{i,1}, \boldsymbol{d}_{i-1}, f, x^i)$;
$z \leftarrow \left\langle [\bar{X}_n]_P, \boldsymbol{d}_{n-1} \right\rangle + [\bar{\tau}_{n-1}]_P w_{n,x^n}$;
return z

Algorithm 11: Eval_i^P – Subroutine for the evaluator in Algorithm 10

Input : $X_{i-1}, \tau_{i-1}, w_{i,0}, w_{i,1}, \boldsymbol{d}_{i-1}, f, x^i$

$[z_i]_P \leftarrow \left\langle [\bar{X}_{i-1}]_P, \boldsymbol{d}_{i-1} \right\rangle + [\bar{\tau}_{i-1}]_P \cdot w_{i,x^i}$;

$([\bar{X}_i]_P, [\bar{\tau}_i]_P) \leftarrow f([z_i]_P)$;

return $([\bar{X}_i]_P, [\bar{\tau}_i]_P)$

Now, if $(X_r, \tau_r) = 0$, that is $[X_r]_1 = [X_r]_2$ and $[\tau_r]_1 = [\tau_r]_2$, then also $\langle X_r, \boldsymbol{d}_{n-1} \rangle + \tau_r w_{n,0} = \langle X_r, \boldsymbol{d}_{n-1} \rangle + \tau_r w_{n,1} = 0$. Thus the children of a dead node are dead also in this case, and thus those nodes will be correctly evaluated.

Now, if $r \in R_{n-1}$ and also $r \in \widehat{R_{n-1}}$, then we have that $\langle X_r, \boldsymbol{d}_{n-1} \rangle = \tau_r w_{r,2}$. Thus the value at $r\|0$ is set to $\tau_r w_{r,2} + \tau_r w_{i,0} = \tau_r(w_{r,2} + w_{i,0})$ and the value at $r\|1$ is set to $\tau_r w_{r,2} + \tau_r w_{i,1} = \tau_r(w_{r,2} + w_{i,1})$. These values are 0 only if $\tau_r = 0$ which happens with probability ε_{col}.

Now, if $r \in R_{n-1}$ and $r \notin \widehat{R_{n-1}}$, then we have that one of the children of r is alive and one is dead. Let us denote the dead child by $r\|b$ and the alive one by $r\|\bar{b}$. We note that by construction, $\langle X_r, \boldsymbol{d}_{n-1} \rangle = \tau_r w_{r,b}$. Thus we have that the evaluation of z at the dead node $r\|b$ is $z \leftarrow \langle X_r, \boldsymbol{d}_{n-1} \rangle + \tau_r w_{n,b} = \tau_r w_{n,b} + \tau_r w_{n,b} = 0$. Thus the evaluation at this point is correct. The evaluation at the alive node is $z \leftarrow \langle X_r, \boldsymbol{d}_{n-1} \rangle + \tau_r w_{n,\bar{b}} = \tau_r w_{n,b} + \tau_r w_{n,\bar{b}} = \tau_r(w_{n,b} + w_{n,\bar{b}})$. This is 0 only if $\tau_r = 0$ which which happens with probability ε_{col}. As the number of $r \in R_{n-1}$ can be no greater than t, then we have that the probability that some a_i evaluates to zero is no greater than

$$(1 - \varepsilon_{col})^t \left((1 - \frac{t}{(2^k)^{v-t+1}}) \cdot (1 - \varepsilon_{col})^t \right)^{n-1}.$$

To show the security of our scheme, we will need to redefine the security games so that they do not contain the $(b_1, \ldots, b_n)$ any more.

Thus our security will be captured by the following proposition.

Proposition 7. *We have that*

$$|\Pr[\mathit{Game}\ 19(\mathcal{A}) = 1] - \Pr[\mathit{Game}\ 21(\mathcal{A}) = 1]| \leq \varepsilon_{\mathsf{FSS}},$$

and

$$|\Pr[\mathit{Game}\ 20(\mathcal{A}) = 1] - \Pr[\mathit{Game}\ 21(\mathcal{A}) = 1]| \leq \varepsilon_{\mathsf{FSS}},$$

where $\overset{P_1}{\leftarrow}$ *denotes the output that is given to* P_1 *and* $\overset{P_2}{\leftarrow}$ *denotes the output that is given to* P_2 *and where* $\varepsilon_{\mathsf{FSS}} = (t \cdot n \cdot \varepsilon_{\mathsf{PRG}} + (n+1) \cdot \frac{t}{(2^k)^{v-t+1}}$

We will now show that Game 19 is ε_{FSS}-close to Game 21.

To see this, we note that due to the construction of the games 1 and 19, and games 3 and 21, the values given to $\mathcal{A}$ in 19 have the same distribution as the first five values given to $\mathcal{A}$ in 1. Likewisethe values given to $\mathcal{A}$ in 21 have the same distribution as the first five values given to $\mathcal{A}$ in 3. Thus, if there existed an adversary $\mathcal{B}$ who was better at distinguishing between games 19 and 21, then

Game 19:
Input : $\mathcal{A}$ - the adversarial algorithm
1 $(a_1, \ldots, a_t) \leftarrow \mathcal{A}()$;
2 $sk_1 \overset{P_1}{\leftarrow} \mathsf{GenPrim}((a_1, \ldots, a_t), f)$;
3 **return** $\mathcal{A}(sk_1)$;

Game 20:
Input : $\mathcal{A}$ - the adversarial algorithm
1 $(a_1, \ldots, a_t) \leftarrow \mathcal{A}()$;
2 $sk_2 \overset{P_2}{\leftarrow} \mathsf{GenPrim}((a_1, \ldots, a_t), f)$;
3 **return** $\mathcal{A}(sk_2)$;

Game 21:
Input : $\mathcal{A}$ - the adversarial algorithm
1 $(a_1, \ldots, a_t) \leftarrow \mathcal{A}()$;
2 $X_\epsilon \overset{\$}{\leftarrow} \mathbb{F}_{2^k}^v$;
3 $\tau_\epsilon \overset{\$}{\leftarrow} \mathbb{F}_{2^k}$;
4 **for** $i = 1$ **to** n **do**
5 $\quad w_{i,0} \overset{\$}{\leftarrow} \mathbb{F}_{2^k} \setminus \{0\}$;
6 $\quad w_{i,1} \overset{\$}{\leftarrow} \mathbb{F}_{2^k} \setminus \{0, w_{i,0}\}$;
7 $\quad \boldsymbol{d}_{i-1} \overset{\$}{\leftarrow} \mathbb{F}_{2^k}^v$;
8 $\boldsymbol{g} \overset{\$}{\leftarrow} \mathbb{F}_{2^k}^v$;
9 $sk \leftarrow X_\epsilon, \tau_\epsilon, \{w_{i,0}\}_{i=1}^n, \{w_{i,1}\}_{i=1}^n, \{\boldsymbol{d}_i\}_{i=0}^{n-1}$;
10 **return** $\mathcal{A}(sk)$;

an adversary $\mathcal{A}$ whose task it was to distinguish between 1 and 3 could just omit his last input and give the inputs to $\mathcal{B}$ to distinguish. Thus we have that

$$|\Pr[\text{Game } 19(\mathcal{A}) = 1] - \Pr[\text{Game } 21(\mathcal{A}) = 1]| \leq \varepsilon_{\mathsf{FSS}},$$

We also note that SLAMPR-FSS has the property that the values at the points $\{a_i\}$ are pseudorandom. More precisely, consider the Games 22 and 23. We will claim that the difference between these two games is negligible.

For that we will use the following natural assumption about our PRG f.

Assumption 1. *Let $f : M \to N$ be a function. Fix a one-dimensional subspace B of N. We say that f is a ε_{sPRG}-subspace-PRG if for any PPT adversary $\mathcal{A}$,*

$$|\Pr[b = 1 | x \xleftarrow{\$} M, y \leftarrow f(x), b \leftarrow \mathcal{A}(y) \wedge y \in B] - \Pr[b = 1 | b \leftarrow \mathcal{A}(y), y \xleftarrow{\$} B]| \leq \varepsilon_{sPRG}$$

Essentially we assume that an adversary cannot distinguish between a random element of a subspace B or a pseudorandom element on the condition that it is an element of B.

Game 22:
Input : $\mathcal{A}$ - the adversarial algorithm

$(a_1, \ldots, a_t) \leftarrow \mathcal{A}()$;
$sk_1, sk_2 \leftarrow \mathsf{Gen}((a_1, \ldots, a_t), f)$;
for $i = 1$ **to** t **do**
  $[b_i]_1 \leftarrow \mathsf{Eval}^1(sk_1, f, a_i)$;
  $[b_i]_2 \leftarrow \mathsf{Eval}^2(sk_2, f, a_i)$;
return $\mathcal{A}(sk_1, \{[b_i]_1\}_{i=1}^t, \{[b_i]_2\}_{i=1}^t)$;

Game 23:
Input : $\mathcal{A}$ - the adversarial algorithm

$(a_1, \ldots, a_t) \leftarrow \mathcal{A}()$;
$sk_1, sk_2 \leftarrow \mathsf{Gen}((a_1, \ldots, a_t), f)$;
for $i = 1$ **to** t **do**
  $[b_i]_1 \leftarrow \mathsf{Eval}^1(sk_1, f, a_i)$;
  $[b_i]_2 \xleftarrow{\$} \mathbb{F}_{2^k}^v$;
return $\mathcal{A}(sk_1, \{[b_i]_1\}_{i=1}^t, \{[b_i]_2\}_{i=1}^t)$;

Proposition 8. *Suppose that f is a ε_{sPRG}-subspace-PRG and ε_{sPRG}-PRG. Then*

$$|\Pr[\textit{Game } 22(\mathcal{A}) = 1] - \Pr[\textit{Game } 23(\mathcal{A}) = 1]| \leq t\varepsilon_{sPRG},$$

Game 24:

Input : $\mathcal{A}$ - the adversarial algorithm

$(a_1, \ldots, a_t) \leftarrow \mathcal{A}()$;

$sk_1, sk_2 \leftarrow \mathsf{Gen}((a_1, \ldots, a_t), f)$;

For $i \in [1, \ldots, n]$, compute $R_i, R'_i, \widehat{R_{i-1}}$ as in SubGenand SubGenPrim;

For all $r \in \bigcup_{i=1}^{n} R_i \cup \bigcup_{i=1}^{n} R'_i$, compute $[X_r]_1, [\tau_r]_1$, using sk_1 and the EvalPrim algorithm ;

Using the EvalPrim algorithm, compute the $(\{[b_i]_1\}_{i=1}^{t})$ from the $\{a_i\}_{i=1}^{t}$ and the $[X_r]_1$ and the $[\tau_r]_1$;

For all $r \in \bigcup_{i=1}^{n} R_i \cup \bigcup_{i=1}^{n} R'_i$, compute $[X_r]_2, [\tau_r]_2$, using sk_2 and the EvalPrim algorithm ;

Using the EvalPrim algorithm, compute the $(\{[b_i]_2\}_{i=1}^{t})$ from the $\{a_i\}_{i=1}^{t}$ and the $[X_r]_2$ and the $[\tau_r]_2$;

return $\mathcal{A}(sk_1, \{[b_i]_1\}_{i=1}^{t}, \{[b_i]_2\}_{i=1}^{t})$;

Game 25:

Input : $\mathcal{A}$ - the adversarial algorithm

$(a_1, \ldots, a_t) \leftarrow \mathcal{A}()$;

$sk_1, sk_2 \leftarrow \mathsf{Gen}((a_1, \ldots, a_t), f)$;

For $i \in [1, \ldots, n]$, compute $R_i, R'_i, \widehat{R_{i-1}}$ as in SubGenand SubGenPrim;

For all $r \in \bigcup_{i=1}^{n} R_i \cup \bigcup_{i=1}^{n} R'_i$, compute $[X_r]_1, [\tau_r]_1$, using sk_1 and the EvalPrim algorithm ;

Using the EvalPrim algorithm, compute the $(\{[b_i]_1\}_{i=1}^{t})$ from the $\{a_i\}_{i=1}^{t}$ and the $[X_r]_1$ and the $[\tau_r]_1$;

For all $r \in \bigcup_{i=1}^{n-2} R_i \cup \bigcup_{i=1}^{n-2} R'_i$, compute $[X_r]_2, [\tau_r]_2$, using sk_2 and the EvalPrim algorithm ;

For all $r \in R_{n-1}$:

if $r \notin \widehat{R_{n-1}}$ **then**

  if $r||0 \in R'_n$ **then**

    Sample randomly $[X_r]_2, [\tau_r]_2$ on condition $\langle [X_r]_2, \boldsymbol{d}_{n-1} \rangle + [\tau_r]_2 \cdot w_{n,0} = \langle [X_r]_1, \boldsymbol{d}_{n-1} \rangle + [\tau_r]_1 \cdot w_{n,0}$;

  else

    Sample randomly $[X_r]_2, [\tau_r]_2$ on condition $\langle [X_r]_2, \boldsymbol{d}_{n-1} \rangle + [\tau_r]_2 \cdot w_{n,1} = \langle [X_r]_1, \boldsymbol{d}_{n-1} \rangle + [\tau_r]_1 \cdot w_{n,1}$;

else

  $[X_r]_2 \xleftarrow{\$} \mathbb{F}_{2^k}^{v}$;

  $[\tau_r]_2 \xleftarrow{\$} \mathbb{F}_{2^k}$

for $r \in R_{n-1}$: **do**

  if $\exists a_i : r||0 = a_i$ **then**

    $[b_i]_2 \leftarrow \langle [X_r]_2, \boldsymbol{d}_{n-1} \rangle + [\tau_r]_2 \cdot w_{n,0}$;

  if $\exists a_i : r||1 = a_i$ **then**

    $[b_i]_2 \leftarrow \langle [X_r]_2, \boldsymbol{d}_{n-1} \rangle + [\tau_r]_2 \cdot w_{n,1}$;

return $\mathcal{A}(sk_1, \{[b_i]_1\}_{i=1}^{t}, \{[b_i]_2\}_{i=1}^{t})$;

Proof. For syntactic sugar, we restate Game 22 as Game 24.

We will now first argue that Games 24 and 25 are $t\varepsilon_{sPRG}$-close.

We note that Game 25 only differs from Game 24 is that here, on lines 7 to 15, instead of using the PRG f with the adversary knowing that the result would satisfy $\langle [X_{r\|b}]_2, \boldsymbol{d}_{n-1}\rangle + [\tau_{r\|b}]_2 \cdot w_{n-1,b} = \langle [X_{r\|b}]_1, \boldsymbol{d}_{n-1}\rangle + [\tau_{r\|b}]_1 \cdot w_{n-1,b}$ where $r\|b \in R'_{n-1}$, we sampled a random vector where this property holds, and that for $r \in \widehat{R}_{i-1}$, we sampled a random value instead of using the output of f. By our assumption that f is a ε_{sPRG}-subspace-PRG (this concerns the changes made for $r\|b \in R'_{n-1}$) and ε_{sPRG}-PRG (this concerns the changes made for $r \in \widehat{R}_{i-1}$) and that there are together at most t instances where these two changes are made, no PPT adversary can distinguish between Games 24 and 25 better than $t\varepsilon_{sPRG}$.

Let us now analyze the output values of the $[b_i]_2$ in Game 25. We will show that they are negligibly close to being uniformly randomly distributed.

Let $r \in R_{n-1}$. If $r \in \widehat{R_{n-1}}$, then we note that the distributions of $\langle [X_r]_2, \boldsymbol{d}_{n-1}\rangle + [\tau_r]_2 \cdot w_{n,0}$ and $\langle [X_r]_2, \boldsymbol{d}_{n-1}\rangle + [\tau_r]_2 \cdot w_{n,1}$ are uniformly at randomly distributed. Thus in those cases $[b_i]_2$ are indeed uniformly at randomly distributed.

If $r \notin \widehat{R_{n-1}}$, then exactly one of $r\|0$ and $r\|1$ is alive and one is dead. Let us denote the dead child by $r\|b$ and the alive one by $r\|\bar{b}$. Now we have that the condition

$$\langle [X_r]_2, \boldsymbol{d}_{n-1}\rangle + [\tau_r]_2 \cdot w_{n,b} = \langle [X_r]_1, \boldsymbol{d}_{n-1}\rangle + [\tau_r]_1 \cdot w_{n,b} \tag{2}$$

holds.

Let the last nonzero index of $\boldsymbol{d}_{n-1}$ be δ that is at position j, denote the vector of the first $j-1$ positions of $\boldsymbol{d}_{n-1}$ by $\bar{d}$.

Let the value at jth position in $[X_r]_2$ be x_r, denote the vector of the first $j-1$ positions of $[X_r]_2$ by $\bar{X_r}$.

We note that $\langle [X_r]_2, \boldsymbol{d}_{n-1}\rangle = \langle \bar{X_r}, \bar{d}\rangle + \delta x_r$.

Denote $\beta := \langle [X_r]_1, \boldsymbol{d}_{n-1}\rangle + [\tau_r]_1 \cdot w_{n,b} - \langle \bar{X_r}, \bar{d}\rangle$. Thus we can rewrite the condition 2 as $\delta x_r + [\tau_r]_2 \cdot w_{n,b} = \beta$.

Note that we have that for every choice of $\bar{X_r}$, we have that the pair $(x_2, [\tau_r]_2)$ satisfies condition 2 iff $\delta x_r = [\tau_r]_2 \cdot w_{n,b} - \beta$.

Note that from the viewpoint of the adversary who knows sk_1 and thus $[X_r]_1, [\tau_r]_1, \boldsymbol{d}_{n-1}$ and $w_{n,b}$, fixing the value of $\bar{X_r}$ also fixes the value of β. We thus have that $[\tau_r]_2 = (\delta x_r + \beta) w_{n,b}^{-1}$. Note that for every value of x_r, the value of $[\tau_r]_2$ is different, thus each value of $[\tau_r]_2$ is equally likely. Now, by construction we have that $r\|\bar{b}$ must be equal to some a_i. Thus we have that $[b_i]_2 = \langle [X_r]_2, \boldsymbol{d}_{n-1}\rangle + [\tau_r]_2 \cdot w_{n,\bar{b}}$. We also have that $[b_i]_1 = \langle [X_r]_1, \boldsymbol{d}_{n-1}\rangle + [\tau_r]_1 \cdot w_{n,\bar{b}}$. Thus

$$b_i = [b_i]_1 + [b_i]_2 = \langle [X_r]_1, \boldsymbol{d}_{n-1}\rangle + [\tau_r]_1 \cdot w_{n,\bar{b}} + \langle [X_r]_2, \boldsymbol{d}_{n-1}\rangle + [\tau_r]_2 \cdot w_{n,\bar{b}} =$$

$$\beta + [\tau_r]_1 \cdot w_{n,\bar{b}} + \delta x_r + [\tau_r]_2 \cdot w_{n,\bar{b}} = \beta + [\tau_r]_1 \cdot w_{n,\bar{b}} + [\tau_r]_2 \cdot w_{n,b} - \beta + [\tau_r]_2 \cdot w_{n,\bar{b}} =$$

$$[\tau_r]_1 \cdot w_{n,\bar{b}} + [\tau_r]_2 \cdot (w_{n,b} + w_{n,\bar{b}}).$$

Now note that from the viewpoint of an adversary who knows sk_1, the values $[\tau_r]_1 \cdot w_{n,\bar{b}}$ is fixed. However $(w_{n,b} + w_{n,\bar{b}})$ is nonzero and possible values of $[\tau_r]_2$

are uniformly at random over $\mathbb{F}_{2^k}$. Thus also the distribution of b_i is uniformly random over $\mathbb{F}_{2^k}$, which, due to $[b_i]_1$ being known to the adversary, means that $[b_i]_2$ is distributed uniformly random. Thus we have shown that in the output of Game 25 the values $[b_i]_2$ are uniformly random. Hence we can say that the outputs of Games 25 and 23 are the same. The claim hence follows. □

C Efficiency Calculations

Let $t_{\mathbb{F}}$ denote the time for one field operation over $\mathbb{F}_{2^k}$, $t_{\mathbb{F}\,vec}$ the time for one operation over vector space $\mathbb{F}_{2^k}^v$ and t_{PRG} the time for one PRG evaluation.

Proposition 9. *Algorithm 1 produces keys with size $\Theta(vkn)$ bits.*

Proof. The key for P_i, $i \in \{1,2\}$ is defined as $([X_\epsilon]_i, [\tau_\epsilon]_i, \{w_{i,0}\}_{i=1}^n, \{w_{i,1}\}_{i=1}^n, \{\boldsymbol{d}_i\}_{i=0}^{n-1}, \boldsymbol{g})$. Element of $\mathbb{F}_{2^k}$ takes k bits to store. Let us analyze the key size element wise:

- $[X_\epsilon]_i \in \mathbb{F}_{2^k}^v$ takes $v \cdot k$ bits.
- $[\tau_\epsilon]_i \in \mathbb{F}_{2^k}$ takes k bits.
- $\{w_{i,0}\}_{i=1}^n \in \mathbb{F}_{2^k}^n$ takes $n \cdot k$ bits.
- $\{w_{i,1}\}_{i=1}^n \in \mathbb{F}_{2^k}^n$ takes $n \cdot k$ bits.
- $\{\boldsymbol{d}_i\}_{i=0}^{n-1} \in \mathbb{F}_{2^k}^{v \cdot n}$ takes $v \cdot n \cdot k$ bits.
- $\boldsymbol{g} \in \mathbb{F}_{2^k}^v$ takes $v \cdot k$ bits.

Let us sum this together and get

$$v \cdot k + k + n \cdot k + n \cdot k + v \cdot n \cdot k + v \cdot k = (v \cdot n + 2v + 2n + 1) \cdot k = \Theta(vkn).$$

□

Lemma 3. *For* SubGen *to check, if a child node $r||b$ of an alive node r is supposed to be dead or alive, takes $O(t \lceil \log t \rceil)$ steps per level.*

Proof. Let us describe a possible algorithm for this operation.

1. For each alive node r keep a list of indices j for which $a_j^1 || \ldots || a_j^{|r|} = r$. Denote it by L_r.
2. During the check, iterate over L_r and check if $(|r|+1)$-th bit of a_j is equal to b. Return this truth value.
3. While iterating, divide indices into two lists $L_{r||0}$ and $L_{r||1}$.

Each index takes $\lceil \log t \rceil$ space and $\lceil \log t \rceil$ time steps to be read. On any given level there are t indices distributed among L_r and since only one bit of each a_j is looked at, the time complexity per level sums up to $O(t \lceil \log t \rceil)$. This means that when evaluating for all $r \in R_{i-1}$, then the total cost is $O(t \lceil \log t \rceil)$. □

Proposition 10. *Algorithm 2 (*SubGen*) is $O(vt^2 \cdot t_{\mathbb{F}} + t \cdot t_{\mathbb{F}\,vec} + t \cdot t_{\mathsf{PRG}})$. In fact it takes at most $2t$ PRG evaluations.*

Proof. Let us count the operations made by SubGen.

- Lines 1 to 5 take $\Theta(t_{\mathbb{F}})$ steps, because the operations are either over $\mathbb{F}_{2^k}$ or constant.
- Lines 6 to 13 take $O(t\lceil \log t \rceil)$, due to check on line 8 taking $O(t\lceil \log t \rceil)$ for all $r \in R_{i-1}$ and other operations can be done in constant time.
- Lines 14 to 25 take $O(t(t_{\mathbb{F}\,vec} + t_{\mathbb{F}}))$ steps, because there are at most t alive nodes in $R'_i \cup \widehat{R}_{i-1}$ and for each alive node a constant number of field and vector space operations are done.
- Line 26 takes $O(vt^2 \cdot t_{\mathbb{F}} + t_{\mathbb{F}\,vec})$ steps. We can use Gaussian elimination on the system of linear equations, which takes $O(vt^2)$ field operations (see Theorem 15 in the Appendix). Choosing $\boldsymbol{d}_{i-1}$ is an operation in the vector space.
- Lines 27 to 32 take $O(t(2t_{\mathsf{PRG}} + t_{\mathbb{F}\,vec} + t_{\mathbb{F}}))$ steps. There are at most t alive nodes in R_i and for each alive node the PRG is called twice and some constant number of field operations and vector space operations are done.

Summing these together we get

$$\Theta(t_{\mathbb{F}}) + O(t\lceil \log t \rceil) + O(t(t_{\mathbb{F}\,vec} + t_{\mathbb{F}})) + O(vt^2 \cdot t_{\mathbb{F}} + t_{\mathbb{F}\,vec}) + \\ + O(t(2t_{\mathsf{PRG}} + t_{\mathbb{F}\,vec} + t_{\mathbb{F}})) = O(vt^2 \cdot t_{\mathbb{F}} + t \cdot t_{\mathbb{F}\,vec} + t \cdot t_{\mathsf{PRG}}).$$

□

Proposition 11. *Algorithm 1 (*Gen*) is $O(nvt^2 \cdot t_{\mathbb{F}} + nt \cdot t_{\mathsf{PRG}})$. In fact it takes at most $2tn$ PRG evaluations.*

Proof. Let us count the operations made by Gen.

- Line 1 to 6 take $\Theta(t_{\mathbb{F}\,vec} + t_{\mathbb{F}})$ steps, because there are constant number of field operations and vector space operations.
- Line 7 and 8 take $O(nt(vt \cdot t_{\mathbb{F}} + t_{\mathbb{F}\,vec} + t_{\mathsf{PRG}}))$ steps, due to n evocations of SubGen.
- Line 9 and 10 take $\Theta(t_{\mathbb{F}\,vec} + t_{\mathbb{F}})$ steps, because there are constant number of field operations and vector space operations.
- Line 11 takes $O(vt^2 \cdot t_{\mathbb{F}} + t_{\mathbb{F}\,vec})$ steps. We can use Gaussian elimination on the system of linear equations, which takes $O(vt^2)$ field operations. Choosing $\boldsymbol{g}$ is an operation in the vector space.
- Lines 12 to 15 take $\Theta(vkn)$ steps, due to the key size.

Summing these together we get

$$\Theta(t_{\mathbb{F}\,vec} + t_{\mathbb{F}}) + O(nt(vt \cdot t_{\mathbb{F}} + t_{\mathbb{F}\,vec} + t_{\mathsf{PRG}})) + \Theta(t_{\mathbb{F}\,vec} + t_{\mathbb{F}}) + \\ + O(vt^2 \cdot t_{\mathbb{F}} + t_{\mathbb{F}\,vec}) + \Theta(vkn) = O(nt(vt \cdot t_{\mathbb{F}} + t_{\mathbb{F}\,vec} + t_{\mathsf{PRG}}) + vkn).$$

Let us assume that $t_{\mathbb{F}\,vec} \approx v \cdot t_{\mathbb{F}}$ and $t_{\mathbb{F}} \geq k$. Then

$$O(nt(vt \cdot t_{\mathbb{F}} + t_{\mathbb{F}\,vec} + t_{\mathsf{PRG}}) + vkn) = O(nvt^2 \cdot t_{\mathbb{F}} + nt \cdot t_{\mathsf{PRG}}).$$

□

Proposition 12. *Algorithm 4 (*Eval^P*) has time complexity* $\Theta(nvt_{\mathbb{F}} + nt_{\mathsf{PRG}})$*. In fact it takes* n *PRG evaluations.*

Proof. Let us count the operations made by Eval^P.

- Line 1 takes $\Theta(vkn)$ steps, due to the key size.
- Lines 2 to 4 take $\Theta(n(t_{\mathbb{F}\,vec} + t_{\mathbb{F}} + t_{\mathsf{PRG}}))$ steps, because lines 3 and 4 are repeated n times.
  - Lines 3 and 4 take $\Theta(t_{\mathbb{F}\,vec} + t_{\mathbb{F}} + t_{\mathsf{PRG}})$ steps, because the scalar product is a vector space operation, addition and multiplication are field operations and the PRG is called once.
- Line 5 takes $\Theta(t_{\mathbb{F}\,vec} + t_{\mathbb{F}})$ steps, because of the scalar product and addition.
- Line 6 takes some constant number of steps.

Let us add these values together and get

$$\Theta(vkn) + \Theta(n(t_{\mathbb{F}\,vec} + t_{\mathbb{F}} + t_{\mathsf{PRG}})) + \Theta(t_{\mathbb{F}\,vec} + t_{\mathbb{F}}) + \Theta(1) = \\ = \Theta(n(t_{\mathbb{F}\,vec} + t_{\mathbb{F}} + t_{\mathsf{PRG}} + vk)).$$

Let us assume that $t_{\mathbb{F}\,vec} \approx v \cdot t_{\mathbb{F}}$ and $t_{\mathbb{F}} \geq k$. Then

$$\Theta(n(t_{\mathbb{F}\,vec} + t_{\mathbb{F}} + t_{\mathsf{PRG}} + vk)) = \Theta(n(v \cdot t_{\mathbb{F}} + t_{\mathbb{F}} + t_{\mathsf{PRG}} + vk)) = \Theta(nvt_{\mathbb{F}} + nt_{\mathsf{PRG}}).$$

□

Proposition 13. *Full evaluation of SLAMP-FSS requires* $2 \cdot 2^n$ *calls to the PRG.*

Proof. We note that a full evaluation of the scheme SLAMP-FSS can be obtained by recursively computing for each level i, for all bitstrings r of length i the values $[X_r]_P, [\tau_r]_P$. Let the parent node of r be r' with $r = r' \| b$. We note that we can compute $[X_r]_P, [\tau_r]_P$ by computing $\langle [X_r]_P, \boldsymbol{d}_{i-1} \rangle + [\tau_r]_P w_{i,b}$ and then applying f to the result. Thus, for full evaluation, we need one application of f for every non-root node in the tree. There are $2 \cdot 2^n - 2$ non-root nodes in the tree which we round to $2 \cdot 2^n$ for easier presentation. □

Proposition 14. *Full evaluation of SLAMPR-FSS requires* 2^n *calls to the PRG.*

Proof. The proof of this proposition is analogous to the previous one but with the difference that we do not need to evaluate f for the leaves. There are $2^n - 2$ non-root-non-leaf nodes in the tree which we round to 2^n for easier presentation. □