1 Changes from previous version

This is a major rewrite from the previous version of the same eprint.

The central algorithm has significantly changed, although it still similar to the one in the previous version. Additionally, we have changed the assumptions introduced, and added a discussion of those assumptions. We also rewrote the proof that it is infeasible to learn certain secrets from the public keys provided that factoring is hard.

More specifically, we still have the construction that the parties hold a secret-shared vector $[\![z_0]\!]$ (or a pair of secret-shared vectors) over $\mathbb{Z}_N$, $2n$ matrices $\{A_{i,b}\}_{i=1,b\in\{0,1\}}^{n}$, and a vector $[\boldsymbol{e}]$ that is given in the exponent of some group $\mathbb{G}$ and that to evaluate the function at point $\alpha = \alpha_1\alpha_2\ldots\alpha_n$, the parties iteratively compute $[\![z_i]\!] \leftarrow B_{i,\alpha_i}[\![z_{i-1}]\!]$ and in the end compute $\prod_{i=1}^{k}[\boldsymbol{e}_i]^{[\![z_{n,i}]\!]}$ to get the output. We also continue with the same idea that if we work over $\mathbb{Z}_N$ where N is a product of two unknown large primes, then if we duplicate our secret variables

However, the construction of the matrices is different. In the previous version, we had secret $2k$ values ${\boldsymbol{z}_i}^{(b)}$ and secret k values ${\boldsymbol{w}_i}^{(b)}$ and the matrices satisfied the property that $A_{i,\alpha_i}{\boldsymbol{w}_{i-1}}^{(b)}$ and $A_{i,\overline{\alpha_i}}{\boldsymbol{w}_{i-1}}^{(b)}$, $A_{i,\overline{\alpha_i}}{\boldsymbol{z}_{i-1}}^{(b)}$ are all collinear to ${\boldsymbol{w}_i}^{(b)}$, and that $A_{i,\alpha_i}{\boldsymbol{z}_{i-1}}^{(b)} = {\boldsymbol{z}_i}^{(b)}$.

In this version we generalize the concept, in that we note that we can consider a vector being collinear to some ${\boldsymbol{w}_i}^{(b)}$ as it belonging to a one-dimensional submodule. In the current version we replace this by the predicates that the secret-shared vector belongs to some secret higher-dimensional submodules. We also construct the matrices somewhat differently, by first considering matrices that preserve certain submodules and then left-and-right multiplying them with matrices that also preserve those submodules. We also use different assumptions and discuss their security.

Multi-Party Distributed Point Functions with Polylogarithmic Key Size from Invariants of Matrices

Toomas Krips[1][0000−0003−0981−3553] and Pille Pullonen-Raudvere[2][0000−0002−3255−7001]

[1] University of Tartu, Estonia
toomas.krips@ut.ee
[2] Cybernetica AS, Estonia
pille.pullonen-raudvere@cyber.ee

Abstract. Distributed point functions (DPFs), introduced in 2014, are a widely used primitive in secure computation for a wide variety of applications. However, until recently, constructions for DPFs with polylogarithmic keys were known only for the two-party setting, multi-party schemes have key sizes exponential in the number of parties or the domain size. We generalize the efficient tree-based two-party DPF approach and get a scheme for a polylogarithmic-size DPF for an any number of parties. We use a technique where we have the invariant for off-path leaves such that secret-shared vector is mapped to secret submodules by public matrices. We show, using a technique by Shamir, that these vectors are hard to compute over $\mathbb{Z}_{pq}$ if factoring is hard. Our scheme is a secure DPF under two new assumptions related to Generic Group Model and the Linear Code Equivalence. The output of our scheme is in the exponent in some group where Diffie-Hellman type problems are hard which limits the usability of the scheme. Still, it is the first multi-party DPF generalizing the tree-based two-party DPF approach. Our scheme is the first where the key is polylogarithmic in the domain size and independent of the number of parties and the key generation and evaluation can be computed efficiently independently of the number of parties.

Keywords: Distributed point functions · Function secret sharing

1 Introduction

Function secret sharing (FSS) [6] is in a sense dual to usual secret sharing. In secret sharing, we have a known function f and a secret-shared value $[\![x]\!]$, we use secure multi-party computation to compute $[\![z]\!]$, where $z = f(x)$. In FSS, the input x is public, but the function f is shared. Thus, one might depict it as $[\![z]\!] \leftarrow [\![f]\!](x)$ – the computing parties know the input x, but they do not know f nor z. An important property of FSS schemes is that the evaluation of the function is a local operation. The only step requiring communication is distributing the shares of the function to the respective parties.

The FSS constructions focus on some specific function families as general efficient constructions are not known. A popular class is secret parametrized functions. We know that the function comes from a certain kind of parametrized functions, and we are satisfied with merely hiding the secret parameter(s). The most common of these schemes is a Distributed Point Function (DPF) [23], where we define a FSS scheme for a point function

$$f_{\alpha,\beta}(x) = \begin{cases} \beta, & \text{if } x = \alpha \\ 0 & \text{otherwise.} \end{cases}$$

Despite their siplicity, point functions quite useful in constructing various primitives such as pseudorandom correlation generators [4], oblivious RAM [11], anonymous messaging [33], secure search [22,19,20], and private histograms [2].

Efficient DPF constructions exist for the 2-party case [6,7,25] . The underlying idea replies on having two parties and computing in a field of characteristic 2. In this field, some secret shared value a is 0 exactly if the shares that the parties have are equal. Hence, these approaches cannot be straightforwardly generalized to more than 2 parties. The idea of DPF in [6] revolves around representing a function as a full binary tree where each level represents an input bit and leaves are the output values. Evaluation means traversing the tree using the input bit-by-bit to choose the next node. In a DPF scheme, both participants receive a secret shared seed for the root node and specific correction words for each level of the tree. At each level, the parties use a pseudorandom generator (PRG) to expand the current seed. They adjust the PRG output using the provided correction word to derive the seeds for the next level and pick the seed based on the next input bit. This can be seen as, for level i, applying an operator $\mathcal{F}_{i,b}$ if the next bit is b. The correction words are designed to ensure that if the output is zero, both participants end up with the same seed for the leaf. However, if the output is non-zero, then their seeds differ, leading to non-zero outputs when the shares are XOR-ed together. The final level of correction words allows to adjust the leaf nodes, to transform of a non-zero shared secret output into the desired value β, while preserving any zero outputs as zeroes. We call the node active if it is on a path to output β and inactive otherwise. Respectively, the path corresponding to the bits of α and leading to leaf β is the active path.

Note that [6] also introduces a basic idea for multi-party DPF, but the construction is based on sharing the truth table of the function rather than extending the tree-based approach. Recently, other works have extended this approach [14,29,24,18]. We introduce an idea where the overall logic follows a similar schema as the 2-party tree-based DPF in [6] but it is applicable to the multi-party case. More related work is discussed in Section 7.

1.1 Our Contributions

In order to generalize the tree-traversal approach to function evaluation, there are three tasks. The first is to generalize the invariant of both parties having the same share for inactive nodes and different for active nodes to some other

conditions to distinguish the active and inactive shared states. Secondly, defining locally evaluatable operators $\mathcal{F}_{i,b}$ for level i that preserve these conditions and can be used to derive the values for the next level of the binary tree. Thirdly, defining a locally evaluatable operator that maps the final state to zero for inactive leaves or the desired output value for the active leaf.

Our first contribution is proposing a general schema in Section 3.2, this proposes concrete properties that these three components need to maintain to be the components of a DPF. Secondly, we provide a concrete instantiation of this schema in Section 4, defining the full secure multi-party DPF construction. We put forward two new assumptions in Section 2.3, one of which is related to Linear Code Equivalence and the other is a heuristic in the style of Generic Group Model. We show the security of our scheme under these assumptions. We note that these assumptions are new and discuss their plausibility in Sections 6. We also rely on DDH assumption and the hardness of factoring (Section C.1). We note that the techniques we introduce are novel, the natural assumptions that relate to them are naturally also less studied.

Our generalization of [6] in Section 3.2 follows the tree traversal idea. However, rather than ensuring that parties have equal values, we define some other invariants for inactive nodes. An invariant is checked by a predicate. We can have several different invariants checked by different predicates. At level i and input bit value $\overline{\alpha_i}$, the operator $\mathcal{F}_{i,\overline{\alpha_i}}$ maps the state of the active node into a value such that the i-th predicate holds and that all operators $\mathcal{F}_{j,0}$, $\mathcal{F}_{j,1}$ with $j > i$ preserve the predicate. Hence, if y satisfies the predicate, then $\mathcal{F}_{j,b}(y)$ will also satisfy that predicate. This ensures that the children of inactive nodes are inactive. For the state at node α, none of the predicates is satisfied. In the end, we apply a final operator mapping all values that satisfy any of the predicates to 0 and the value at the node α to the desired value β.

Concretely, for every level $i \in [0, n]$, we have a secret submodule of $\mathbb{Z}_N^k$ denoted by $\tilde{C}'_i$. First consider the case where our operators are public square $k \times k$ matrices $A_{j,b}$ and the state is a secret-shared vector $[\![\overline{\boldsymbol{z}}_i]\!]$. At every level i, we compute $[\![\overline{\boldsymbol{z}}_i]\!] \leftarrow A_{j,b}[\![\overline{\boldsymbol{z}}_{i-1}]\!]$, where $b \in \{0, 1\}$. The predicate is satisfied if $\overline{\boldsymbol{z}}_i \in \tilde{C}'_i$. The matrices $A_{j,b}$ are constructed in such a way that they map $\tilde{C}'_{i-1}$ to $\tilde{C}'_i$. This construction ensures that if the state vector at a node at level belongs to $\tilde{C}'_i$, then also all its descendants at level $j > i-1$ will belong to $\tilde{C}'_j$. At active nodes, at level i the secret-shared vector is some other vector $\boldsymbol{z}_i$ that does not belong to $\tilde{C}'_i$. However, we must have that $A_{i+1,\overline{\alpha_{i+1}}}\boldsymbol{z}_i \in \tilde{C}'_{i+1}$.

If all the $\tilde{C}'_i$ are 1-dimensional, then at one level of our scheme we obtain a pair of matrices $A_{i,0}$ and $A_{i,1}$ that map the same vector to collinear vectors. This is what the condition that $A_{i,0}$ and $A_{i,1}$ must map $\tilde{C}'_{i-1}$ to $\tilde{C}'_i$ means for $\tau = 1$. We note that in this case this can be characterized as an instance of a MinRank problem with two matrices. MinRank is a type of hardness assumption that is used in multivariate cryptography. However, MinRank is typically more studied with more matrices than two, and it could be that with two matrices, the problem is not that hard. An attack against MinRank suggests that with two matrices, there could be an attack of complexity approximately around $O(n^6)$.

However, if we increase τ, then we can, with some more work, also be turned into a MinRank issue, however, for that issue, the complexity of that algorithm becomes roughly $O((n\tau)^{\tau 2.37})$. We discuss this in Appendix E. Due to this reason we require that our submodules have dimension at least $\tau > 1$.

We chose to have a more concise description of the matrices $A_{i,b}$ and the secret submodules. Thus, we have second secret submodules $\tilde{C}_i$ that have a somewhat more simple description. For every level we also sample a random invertible matrix F_i, and have that $\tilde{C}'_i = F_i\tilde{C}_i$. To get a more simple description of the submodules $\tilde{C}_i$, we sample random vectors $\boldsymbol{c_1}, \ldots, \boldsymbol{c_{\tau+n}}$ and let $\tilde{C}_i = \langle \boldsymbol{c_1}, \ldots, \boldsymbol{c_{\tau+i}} \rangle$. We also sample some more matrices that behave well on the $\tilde{C}_i$, along with some other constraints, to build the publicly known matrices $A_{i,0}$ and $A_{i,1}$ from multiplying and adding those matrices and the F_i and F_{i-1}^{-1}. More precisely, we have the central matrix B_0 that is built to take the on-path values to off-path values, that is, to the $\tilde{C}_i$. Since there are only n on-path values, we can hard-code the behavior of B_0 for all of them. On the other hand, we have the other central matrix B_1 that is built so that it takes on-path values to on-path values.

However, if our construction was only that $(A_{i,0}, A_{i,1}) = (F_iB_0F_{i-1}^{-1}, F_iB_1F_{i-1}^{-1})$ or $(A_{i,0}, A_{i,1}) = (F_iB_1F_{i-1}^{-1}, F_iB_0F_{i-1}^{-1})$, depending on the value of α_i, then our construction would be problematic. Namely, in that case one could consider consecutive levels where F_i is repeated, albeit in inverse form. In that case, our solution would be very close to a *code equivalence problem with repeat use of a secret*, such as 2-LCE. Budroni et al showed in [10] that such a construction can be attacked and that one can recover the secret. For that reason we add to the left and right of B_0 and B_1 some more matrices that are chosen freely except for the constraint that they should preserve the submodules $\tilde{C}_{i-1}$ and $\tilde{C}_i$ respectively and that they should map the on-path values to some vectors that we specify.

This means that the central part of the $A_{i,0}$ and $A_{i,1}$ do not describe equivalent codes, they merely have a (small) common subcode. Hence, the attack in [10] does not work, as it requires building a large matrix from certain known matrices that all map one vector (which is a matrix in column form) to zero. When there are more instances of the same code, one can get a system of linear equations with more equations than unknowns. When the common part is a subcode, one would not get such one vector, as one would need to change that vector by the projection to a subcode, for every case. Thus, the attack would not work, as there is no common vector in the kernel of all the cases, but all the vectors are different and the part that makes them different is unknown.

We can slightly linearly mix the matrices B_0 and B_1. For that reason we have

$$\begin{pmatrix} A_{i,0} \\ A_{i,1} \end{pmatrix} \leftarrow \begin{pmatrix} F_i & 0 \\ 0 & F_i \end{pmatrix} \begin{pmatrix} \overline{E}_{i,1+3\pi_i(0)} & \overline{E}_{i,2+\pi_i(0)} \\ \overline{E}_{i,2+\pi_i(1)} & \overline{E}_{i,1+3+\pi_i(1)} \end{pmatrix} \begin{pmatrix} B_{\pi_i(0)} \\ B_{\pi_i(1)} \end{pmatrix} E_iF_{i-1}^{-1}.$$

The permutation π_i depends on the α_i and is either the identity permutation on $\{0,1\}$ or the switching permutation. On the active path, we specify the vectors $\boldsymbol{d_i}$ for $i \in [0, 3n]$ and the intermediate vectors that the matrices E_i, B_1 and $\overline{E}_{i,4}$ take

them to. For vectors $\boldsymbol{d_i}$ for $i \in [0, 3n]$, we need that for all $i \in [1, n]$, $E_i \boldsymbol{d_{3(i-1)}} \in \boldsymbol{d_{3(i-1)+1}} + \tilde{C}_{i-1}$, that $B_1 \boldsymbol{d_{3(i-1)+1}} \in \boldsymbol{d_{3(i-1)+2}} + \tilde{C}_i$ and that $\overline{E}_{i,4} \boldsymbol{d_{3(i-1)+2}} \in \boldsymbol{d_{3i}} + \tilde{C}_i$, on one hand, and $B_0 \boldsymbol{d_{3(i-1)+1}} \in \tilde{C}_i$ and $\overline{E}_{i,2} \boldsymbol{d_{3(i-1)+2}} \in \tilde{C}_i$, on the other. These requirements mean that we control how the input $\boldsymbol{d_0}$ behaves on the active path. On the active path, at ith level, our secret vector is an element of $F_i(\boldsymbol{d_{3i}} + \tilde{C}_i)$ and if we leave the active path, it is mapped to the respective $F_i \tilde{C}_i$.

The final operator is scalar multiplication by a vector $\boldsymbol{e}$ that belongs to the orthogonal module of $\tilde{C}_n$, thus causing the output at all inactive leaves to be 0, and the value at the active leaf to be nonzero. We scale the input $[\![\boldsymbol{z}_0]\!]$ accordingly so that the output there would be β. However, if we give the final vector just as elements of $\mathbb{Z}_N$, then it is possible to detect the active leaf in polynomial time, since one can compute certain vectors corresponding to the leaves such that a matrix constructed from these vectors has a full rank if and only if it contains the active leaf. We describe this attack in Appendix D. Due to this we put this final vector $\boldsymbol{e}$ into the exponent of some group where DDH-style problems are hard, thus making it hard to distinguish whether a matrix given in the exponent has full rank or not. In proofs we consider an assumption to remove this vector-in-the-exponent. It states that if we can distinguish this $[\boldsymbol{e}]$ from a random vector, then it we are able to produce an element of $F_n \tilde{C}_n$ and, thus, there must be an extractor who can extract it. We tie this assumption to the Generic Group Model. However, for our security to hold, we need that it is hard to produce an element of $F_n \tilde{C}_n$.

Our keys consist of the shared vector $\boldsymbol{z}_0$, the square matrices $\{A_{i,b}\}_{i=1,b=0}^{n,1}$ and $[\boldsymbol{e}]$. The key size is dominated by $A_{i,b}$. We require that each matrix satisfies $\tau + 2n$ constraints, thus the number of rows and columns in the matrices must be at least $\tau + 2n$. Since τ is a small constant, the key size is $O(n^3)$.

Concerning the hardness of producing elements of the secret submodules, it turns out that if we construct our scheme over $\mathbb{Z}_N$ where $N = pq$ where p and q are two (possibly unknown) large primes, then we can reduce the hardness of finding any element of $F_n \tilde{C}_n$ to the hardness of factoring. We use a powerful technique by Shamir [37] that lets one show that when in a multivariate system of polynomial equations over $\mathbb{Z}_{pq}$, the system is invariant under some nontrivial rational transformation between the variables and the number of solutions is polynomial, then finding one solution is equivalent to factoring the modulus. In order to use this result, we double the number of secret variables while still keeping the number of matrices the same. For example, for every level, we define two submodules $\tilde{C}_i^{(1)}$ and $\tilde{C}_i^{(2)}$. This allows us to define a nontrivial transformation that satisfies the requirements of Shamir's result. Meaning that finding the relevant secret values is as hard as factoring. Moreover, we specify our result so that learning any part of the secret allows us to learn the whole secret and thus to factor N. This result, and the construction where all the variables are doubled, are rather technical and we give them in Appendix C.

We note that the only constraints that we have on matrices $\overline{E}_{i,c}$ and E_i are that they preserve the submodules $\tilde{C}_i$ and that they behave in a specific

way on one of the $\boldsymbol{d_j}$. Thus, we want to argue that if one does not manage to find any element of the $\tilde{C}_i$,$\tilde{C}_i^{\perp}$ or any $\boldsymbol{d_j}$, then it is infeasible to distinguish $A_{i,0}$ and $A_{i,1}$ from random, in a similar way to code equivalence problems. More specifically, in order to show the security of distinguishing our system, we have two code-equivalence style assumptions. The security of our scheme follows from the second one, that is somewhat involved. However, we argue that the second assumption can be considered a repeated version of the first assumption code-equivalence assumption that we introduce (with some extra hints). The first assumption states that $\begin{pmatrix} \overline{E}_{i,1} & \overline{E}_{i,2} \\ \overline{E}_{i,3} & \overline{E}_{i,4} \end{pmatrix} \begin{pmatrix} B_{\pi_i(0)} \\ B_{\pi_i(1)} \end{pmatrix} E_i F_{i-1}^{-1}$ (where B_0 and B_1 are known F_i in a random invertible matrix and the $\overline{E}_{i,c}$ and E_i are chosen at random under given constraints) either the adversary cannot guess the π_i better than random chance or an extractor can produce an element from one of the constraints that the $\overline{E}_{i,c}$ or E_i must satisfy.

We contrast this with the linear code equivalence problem, where the assumption states that $E \begin{pmatrix} B_0 \\ B_1 \end{pmatrix} F$ is indistinguishable from random where B_0, B_1 are known, F is a random invertible matrix and E is a monomial matrix, that is, a permutation matrix where instead of 1-s one can have arbitrary nonzero values. In general, in code equivalence problems, code generator matrices are hidden from left and right with some other hiding matrices that may have some structure. In our case, the structure of those hiding matrices is such that they must satisfy some constraints related to the code generator matrix, otherwise they are random. We note that these constraints might be a security issue, thus, we modify our assumption to state that if an adversary is able to break that assumption then it must have figured out some element subject to that constraint.

The set from where E is chosen in the linear code equivalence problem is much smaller than the set from where the matrices $E_{i,c}$ and $\overline{E}_i$ are chosen. However, our matrices are constrained to behave in a certain way under certain inputs. Thus, our assumption states that *if* the adversary manages to distinguish our case from random, they must know some element of that set (i.e. there is an extractor that produces an element of that set). We use the Shamir-technique to show that producing one of those elements implies factoring N. In the Submodule Vector Discrete Logarithm Assumption, if we sample a vector randomly from the orthogonal complement of some submodule $\tilde{B}$, and put it into the exponent, then the only way we can distinguish it from random is to produce an element of $\tilde{B}$. We discuss this assumption further in Section 6.

As a result we propose a multi-party DPF based on the hardness of factoring, two new assumptions and linear secret sharing in Section 4 for ring $\mathbb{Z}_{pq}$ with key size $O(n^3)$ for input domain of bitlength n. Our scheme is applicable to any number of $m \geq 2$ parties and secure for up to $m-1$ corrupted parties in the honest-but-curious setting. The final evaluation output can be reconstructed using multiplication and decoded if the domain for β is small.

2 Notation and preliminaries

2.1 General notation

We use a **bold** typeface to denote vectors and we denote matrices by capital letters. The vectors are column vectors by default. The exceptions are ring R, group $\mathbb{G}$, i-th party P_i of the m parties and the domain of the functions D_i. We denote by $[a, b]$ the set $\{a, a+1, \ldots, b\}$.

For a vector $\boldsymbol{x}$ of length k, we denote the diagonal matrix where the elements on the diagonal are the elements of $\boldsymbol{x}$ by $\mathsf{diag}(\boldsymbol{x})$ the $k \times k$. Given ℓ vectors $\boldsymbol{x_1}, \ldots, \boldsymbol{x_\ell} \in \mathbb{Z}_{pq}^k$, we use $\mathsf{ToMat}(\boldsymbol{x_1}, \ldots, \boldsymbol{x_\ell})$ to denote the matrix with the column vectors $\boldsymbol{x_1}, \ldots, \boldsymbol{x_\ell}$ in that order. In a group $\mathbb{G}$ with a generator g, for an integer a, we denote by $[a]$ the element g^a. We assume that the size of $\mathbb{G}$ is N. If discrete logarithm is hard, then $[a]$ hides the value a.

For a bitstring b of length ℓ, we denote its bit representation as a sequence $b_1, \ldots, b_\ell$. A binary string a of length ℓ is a prefix of a binary string b, if $a_i = b_i$ for all $i \leq \ell$. In a full binary tree, each node has two or zero children. It can be seen as representing bitstrings. The root node corresponds to the empty bitstring and the children of a node correspond to appending 0 or 1 bit to the value of the node. Hence, the bitstring corresponding to a node is the prefix of all its descendants. A node has length ℓ, if the corresponding bitstring has length ℓ.

For $\{0, 1\}$, we denote the identity permutation as π_{id} and the switching permutation as π_{sw}, where $\pi_{sw}(0) = 1$ and $\pi_{sw}(1) = 0$. A predicate is a mapping from some set to $\{0, 1\}$. The predicate holds if the output is 1. We use κ as the security parameter, $\mathcal{F}_{i,b}$ as the operator at level i and next input bit b, $N = pq$ where p and q are large primes. By $\mathbb{Z}_{pq}$ we denote the ring of elements modulo N. We denote the point function outputting β for input α and 0 elsewhere as $f_{\alpha,\beta}$. We use the abbreviation PPT for probabilistic polynomial-time.

2.2 Mathematics

We use a capital letter with a tilde to denote submodules, for example $\tilde{C}$. For a $k \times m$ matrix A over some ring $\mathbb{Z}_{pq}$, and a submodule $\tilde{C} \in R^m$, we denote $A\tilde{C} := \{A\boldsymbol{x} | \boldsymbol{x} \in \tilde{C}\}$. For a vector $\boldsymbol{d} \in R^m$ and a submodule $\tilde{C} \in R^m$, we denote $\boldsymbol{d} + \tilde{C} = \{\boldsymbol{d} + \boldsymbol{x} | \boldsymbol{x} \in \tilde{C}\}$.

We say that $G \in R^{m \times m'}$ is a generator matrix for a submodule $\tilde{C} \in R^m$ if $\tilde{C} = \{G\boldsymbol{x} | \boldsymbol{x} \in R^{m'}\}$. A submodule is sometimes also called a *code* and its generator matrix the generator matrix of a code. We note that the generator matrix of a code is usually given in the way where we multiply $\boldsymbol{x}$ by G from the right, not the left, along with other necessary changes, however, in this paper, we give the version where we multiply $\boldsymbol{x}$ by G from the left. For a square $k \times k$ matrix A over a ring R, an eigenvector is a vector $\boldsymbol{x}$ such that there exists an element $\lambda \in R$ such that $A\boldsymbol{x} = \lambda\boldsymbol{x}$. The value λ is called an *eigenvalue* corresponding to $\boldsymbol{x}$, the tuple $(\lambda, \boldsymbol{x})$ is called an eigenpair.

Let the column vectors of an invertible square matrix Q be $\boldsymbol{q_1}, \ldots, \boldsymbol{q_k}$. Let $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_k)$ be a vector and $\Lambda := \mathsf{diag}(\boldsymbol{\lambda})$. Then the eigenvectors of $A :=$

$Q\Lambda Q^{-1}$ are $\boldsymbol{q_1}, \ldots, \boldsymbol{q_k}$ and the corresponding eigenvalues are $(\lambda_1, \ldots, \lambda_k)$. For a square matrix A, the sum of the elements on its diagonal is said to be the trace of A and it is denoted as $\mathsf{tr}(A)$. It is known that the sum of the eigenvalues of A is equal to the trace of A [13]. Let B be a submodule of A, the codimension of a submodule is defined as $\operatorname{codim} B = \dim A - \dim B$.

$GL_m(R)$ is the set of $m \times m$ invertible matrices with elements from the ring R. $\mathsf{Perm}_m(R)$ is the set of $m \times m$ permutation matrices with elements from the ring R, that is, matrices where there is exactly one 1 on each row and column and all the other elements are 0. $\mathsf{Mono}_m(R)$ is the set of $m \times m$ monomial matrices with elements from the ring R that is, matrices where there is exactly one nonzero element on each row and column and all the other elements are 0. We say that a matrix is chosen with v constraints when the matrix is chosen so that it must map some v linearly independent vectors to certain outputs. We may require that the outputs are other vectors, but we may also be more lax, and simply require that they are mapped to a certain submodule or coset.

If $\boldsymbol{a}$ is an eigenvector to a square matrix A and if $\boldsymbol{b}$ is collinear with $\boldsymbol{a}$, then $\boldsymbol{b}$ is an eigenvector of A. We denote size k identity matrix by I_k. If the size is not important or if it is obvious, we just write I. We say two vectors $\boldsymbol{x}, \boldsymbol{y}$ are orthogonal, if $\langle \boldsymbol{x}, \boldsymbol{y} \rangle = 0$. We denote the orthogonal complement of $\tilde{B} \subseteq R^m$ as $\tilde{B}^{\perp}$, that is $\tilde{B}^{\perp} = \{\boldsymbol{x} \in R^m | \forall \boldsymbol{y} \in \tilde{B}, \langle \boldsymbol{y}, \boldsymbol{x} \rangle = 0\}$. If $\{\boldsymbol{x_i}\}_{i=1}^t$ is a set of vectors of equal length over a ring R, then we denote the set $\{\sum_{i=1}^t a_i \boldsymbol{x_i} | a_i \in R \forall i \in [1,t]\}$ by $\mathsf{span}(\{\boldsymbol{x_i}_{i=1}^t\})$. For a given submodule $\tilde{C}$, if we are given a basis for it, then we can reduce it to the Reduced Row Echelon form, which we denote by $\mathsf{RREF}(\tilde{C})$.

2.3 Cryptographic assumptions

For the proof of our scheme we need two new assumptions, but we additionally consider three more assumptions that are variations of one of them, to get a better understanding of it. The first assumption is a heuristic that claims that when a vector $\boldsymbol{e}$ is randomly chosen from some submodule $\tilde{B}^{\perp}$ where $\tilde{B}$ is a submodule of $\mathbb{Z}_N^k$ with dimension ρ (about which the adversary is given partial information) and the adversary is given $[\boldsymbol{e}]$, then the adversary can only distinguish $[\boldsymbol{e}]$ from random if it can find some vector from the $\tilde{B}$. In short, if the adversary can distinguish $[\boldsymbol{e}]$ from random, then there must exist an extractor who outputs an element from $\tilde{B}$. We hope to get a better understanding of the security in the future, and to tie this heuristic to the Generic Group Model [38].

Assumption 1 ($(\mathbb{G}, \mathcal{F}, \varrho)$-Submodule Discrete-Log Vector Assumption). *Let $\mathbb{G}$ be a group of order N with a generator g. Let $\mathcal{F}$ be a class of functions. Let $f \in \mathcal{F}$. Group $\mathbb{G}$ satisfies Submodule Discrete-Log Vector Oracle Heuristic for $\mathcal{F}$, if for any $f \in \mathcal{F}$ and adversary $\mathcal{A}$, there exists an extractor $\mathtt{Extr}_{\mathcal{A}}$, such that if*

$$|Pr[\mathcal{A}([\boldsymbol{e}], f(\boldsymbol{b_1}, \ldots, \boldsymbol{b_\varrho})) | \boldsymbol{b_1}, \ldots, \boldsymbol{b_\varrho} \xleftarrow{\$} (\mathbb{Z}_N)^k, \boldsymbol{e} \xleftarrow{\$} \mathsf{span}(\boldsymbol{b_1}, \ldots, \boldsymbol{b_\varrho})^{\perp}]$$
$$- \Pr[\mathcal{A}([\boldsymbol{r}], f(\boldsymbol{b_1}, \ldots, \boldsymbol{b_\varrho})) | \boldsymbol{r}, \boldsymbol{b_1}, \ldots, \boldsymbol{b_\varrho} \xleftarrow{\$} (\mathbb{Z}_N)^k]| \geq \epsilon(\kappa)$$

for some non-negligible function $\epsilon()$, then

$$\Pr[\boldsymbol{b'} \in \mathsf{span}(\boldsymbol{b_1}, \ldots, \boldsymbol{b_\varrho}) | \boldsymbol{b'} \leftarrow \mathtt{Extr}_{\mathcal{A}}\big([\boldsymbol{e}], f(\boldsymbol{b_1}, \ldots, \boldsymbol{b_\varrho})\big), \boldsymbol{b_1} \ldots, \boldsymbol{b_\varrho} \xleftarrow{\$} (\mathbb{Z}_N)^k, \\ \boldsymbol{e} \xleftarrow{\$} \mathsf{span}(\boldsymbol{b_1}, \ldots, \boldsymbol{b_\varrho})^{\perp}] \geq \mu(\kappa)$$

where $\mu()$ is some non-negligible function.

Our code-based assumptions are similar to Linear Code Equivalence (see, for example [40]), in a way that we will discuss in Section 6.

Assumption 2 (Decisional Linear Code Equivalence). *Let R be a ring. Let $G \in R^{m \times m'}$ be a generator matrix of a linear codes of dimension m'. The Decisional Linear Code Equivalence requires to distinguish $(G, G' | G' \xleftarrow{\$} R^{m \times m'})$ from $(G, G' | S \xleftarrow{\$} GL_{m'}(N), Q \xleftarrow{\$} \mathsf{Mono}_m(N), G' \leftarrow QGS)$.*

We note that we give the matrices in the assumption in the transposed way compared to the way that this assumption is usually stated. This is because code generator matrices usually multiply vectors from the right, but we use matrices to multiply vectors from the left. Thus, to get an equivalent statement we had to reverse the usual order in which this assumption is commonly given.

Assumption 3 ((R, k, τ, n, i, u)-Switching-Decisional Linear Code Equivalence With Common Submodules). *Let R be a ring. Let $\boldsymbol{c_1}, \ldots, \boldsymbol{c_{\tau+n}} \xleftarrow{\$} R^k$. Let $B_0, B_1 \in R^{k \times k}$ be randomly sampled under the condition that $B_c \langle \boldsymbol{c_1}, \ldots, \boldsymbol{c_{\tau+i}} \rangle \subseteq \langle \boldsymbol{c_1}, \ldots, \boldsymbol{c_{\tau+i+1}} \rangle$ for any $i \in [0, n-1]$. Let $\boldsymbol{d} \in R^k$ be an arbitrary element such that $B_0 \boldsymbol{d} \in \langle \boldsymbol{c_1}, \ldots, \boldsymbol{c_{\tau+i}} \rangle$ and $B_1 \boldsymbol{d} \notin \langle \boldsymbol{c_1}, \ldots, \boldsymbol{c_{\tau+i}} \rangle$.*

Choose $\overline{E}_1, \overline{E}_2, \overline{E}_3, \overline{E}_4$ randomly under the constraint that their codimension size is u, that for all $c \in [1, 4]$, $\overline{E}_c \langle \boldsymbol{c_1}, \ldots, \boldsymbol{c_{\tau+i+1}} \rangle \subseteq \langle \boldsymbol{c_1}, \ldots, \boldsymbol{c_{\tau+i+1}} \rangle$ and that $\overline{E}_2 B_1 \boldsymbol{d} \in \langle \boldsymbol{c_1}, \ldots, \boldsymbol{c_{\tau+i}} \rangle$. Choose E randomly under the constraint that $E \langle \boldsymbol{c_1}, \ldots, \boldsymbol{c_{\tau+i}} \rangle \subseteq \langle \boldsymbol{c_1}, \ldots, \boldsymbol{c_{\tau+i}} \rangle$ and that $\boldsymbol{d} \in \operatorname{Im} E$. Choose $F \xleftarrow{\$} GL_k(N)$ at random.

Choose, at random, a bit α_i. If $\alpha_i = 1$, set $\pi_i = \pi_{id}$, else set $\pi_i = \pi_{sw}$. Compute

$$\begin{pmatrix} A_{i,0} \\ A_{i,1} \end{pmatrix} \leftarrow \begin{pmatrix} \overline{E}_{1+3\pi_i(0)} & \overline{E}_{2+\pi_i(0)} \\ \overline{E}_{2+\pi_i(1)} & \overline{E}_{1+3+\pi_i(1)} \end{pmatrix} \begin{pmatrix} B_{\pi_i(0)} \\ B_{\pi_i(1)} \end{pmatrix} E_i F_i^{-1}.$$

Then the (R, k, τ, n, j, u)-Switching-Decisional Linear Code Equivalence With Common Submodules Assumption states that if an adversary who is given B_0, B_1, $A_{i,0}$ and $A_{i,1}$ can guess α_i better than random chance, then there is an extractor $\mathtt{Extr}$ who can output either an element of $\langle \boldsymbol{c_1}, \ldots, \boldsymbol{c_{\tau+i}} \rangle$, some element of $\boldsymbol{d} + \langle \boldsymbol{c_1}, \ldots, \boldsymbol{c_{\tau+i}} \rangle$, an element $\boldsymbol{d'}$ that $E\boldsymbol{d'} \in \boldsymbol{d} + \langle \boldsymbol{c_1}, \ldots, \boldsymbol{c_{\tau+i}} \rangle$ or an element from $\langle \boldsymbol{c_1}, \ldots, \boldsymbol{c_{\tau+i}} \rangle^{\perp}$.

Assumption 4 ((R, k, τ, n, u)-Multi-Switching-Decisional Linear Code Equivalence With Common Submodules). *Let R be a ring. Let $\boldsymbol{c_1}, \ldots, \boldsymbol{c_{\tau+n}} \xleftarrow{\$}$*

R^k. *Let* $B_0, B_1 \in R^{k\times k}$ *be such that* $B_b\langle \boldsymbol{c_1}, \ldots, \boldsymbol{c_{\tau+i}}\rangle \subseteq \langle \boldsymbol{c_1}, \ldots, \boldsymbol{c_{\tau+i+1}}\rangle$ *for any* $i \in [0, n-1]$.

For $i \in [1, n]$, *let* $\boldsymbol{d_{3(i-1)+1}} \in R^k$ *be an arbitrary element such that* $B_0\boldsymbol{d_{3(i-1)+1}} \in \langle \boldsymbol{c_1}, \ldots, \boldsymbol{c_{\tau+i}}\rangle$ *and* $B_1\boldsymbol{d_{3(i-1)+1}} \notin \langle \boldsymbol{c_1}, \ldots, \boldsymbol{c_{\tau+i}}\rangle$.

For $i \in [1, n]$, *choose* $\overline{E}_{i,1}, \overline{E}_{i,2}, \overline{E}_{i,3}, \overline{E}_{i,4}$ *randomly under the constraint that their codimension size is* u, *that for all* $c \in [1, 4]$, $\overline{E}_{i,c}\langle \boldsymbol{c_1}, \ldots, \boldsymbol{c_{\tau+i+1}}\rangle \subseteq \langle \boldsymbol{c_1}, \ldots, \boldsymbol{c_{\tau+i+1}}\rangle$ *and that* $\overline{E}_{i,2}B_1\boldsymbol{d} \in \langle \boldsymbol{c_1}, \ldots, \boldsymbol{c_{\tau+i}}\rangle$. *For* $i \in [1, n]$, *choose* E_i *randomly under the constraint that* $E_i\langle \boldsymbol{c_1}, \ldots, \boldsymbol{c_{\tau+i}}\rangle \subseteq \langle \boldsymbol{c_1}, \ldots, \boldsymbol{c_{\tau+i}}\rangle$ *and that* $\boldsymbol{d_{3(i-1)+1}} \in \operatorname{Im} E_i$. *For* $i \in [1, n]$, *choose* $F_i \xleftarrow{\$} GL_k(N)$ *at random.*

For $i \in [1, n]$, *denote by* $\boldsymbol{d_{3(i-1)}}$ *some such vector that* $E_i\boldsymbol{d_{3(i-1)}} \in \boldsymbol{d_{3(i-1)+1}} + \langle \boldsymbol{c_1}, \ldots, \boldsymbol{c_{\tau+i}}\rangle$. *Now choose for* $i \in [1, n]$ *a* $H_i \xleftarrow{\$} GL_k(n)$ *under the constraint that* $H_iF_i^{-1}\overline{E}_{i,4}B_1\boldsymbol{d_{3(i-1)+1}} = F_{i+1}^{-1}\boldsymbol{d_{3i}}$.

Let the adversary $\mathcal{A}$ *choose two* n*-bit bitstrings* $\alpha^{[1]}$ *and* $\alpha^{[2]}$. *Choose at random* $\overline{b} \leftarrow \{1, 2\}$. *Let* $\alpha = \alpha^{[b]}$ *and let* $\alpha = \alpha_1, \ldots, \alpha_n$.

Now, for every $i \in [1, n]$, *if* $\alpha_i = 1$, *set* $\pi_i = \pi_{id}$, *else set* $\pi_i = \pi_{sw}$.

Now compute

$$\begin{pmatrix} A_{i,0} \\ A_{i,1} \end{pmatrix} \leftarrow \begin{pmatrix} \overline{E}_{i,1+3\pi_i(0)} & \overline{E}_{i,2+\pi_i(0)} \\ \overline{E}_{i,2+\pi_i(1)} & \overline{E}_{i,1+3+\pi_i(1)} \end{pmatrix} \begin{pmatrix} B_{\pi_i(0)} \\ B_{\pi_i(1)} \end{pmatrix} E_iF_i^{-1}.$$

The adversary is given $\{A_{i,0}\}_{i=1}^n$,$\{A_{i,1}\}_{i=1}^n$ *and* $\{H_iF_i^{-1}\}_{i=1}^n$. *The* (R, k, τ, n)*-Multi-Switching-Decisional Linear Code Equivalence With Common Submodules-Assumption states that if the adversary can guess* $\overline{b}$ *better than random, then there exists an extractor* `Extr`, *who, given* $\{A_{i,0}\}_{i=1}^n$,$\{A_{i,1}\}_{i=1}^n$ *and* $\{H_iF_i^{-1}\}_{i=1}^n$ *can produce either an element of* $\langle \boldsymbol{c_1}, \ldots, \boldsymbol{c_{\tau+i}}\rangle$, *some* $\boldsymbol{d_{3(i-1)}} + \langle \boldsymbol{c_1}, \ldots, \boldsymbol{c_{\tau+i}}\rangle$ *or some* $\boldsymbol{d_{3(i-1)+1}} + \langle \boldsymbol{c_1}, \ldots, \boldsymbol{c_{\tau+i}}\rangle$ *or an element from* $\langle \boldsymbol{c_1}, \ldots, \boldsymbol{c_{\tau+i}}\rangle^{\perp}$.

In Appendix A, we give two extensions of these code-based assumptions. In these extra assumptions, certain unknown variables are duplicated as we will need this duplication in the final scheme.

2.4 Secret sharing

A secret sharing scheme has two operations (Share, Reconstruct) where the former generates secret shares and the latter restores the value from the shares. In this paper, we consider additive secret sharing where Share picks $m-1$ random values $[\![x]\!]_i \in R$ and sets $[\![x]\!]_m = x - \sum_{n=1}^{m-1}[\![x]\!]_i$, party P_i gets $[\![x]\!]_i$. The Reconstruct algorithm sums all the shares. Additive secret sharing scheme over a ring R is perfectly hiding, meaning that all shares are needed to learn anything about the secret. Linear combinations of the shares can be computed locally. Notably, for a public matrix M and a secret shared vector $[\![\boldsymbol{s}]\!]$, the operation $M \cdot [\![\boldsymbol{s}]\!]$ can be computed by each party P_i locally computing $M \cdot [\![\boldsymbol{s}]\!]_i$. Hence, in the following algorithms, $[\![a]\!] \leftarrow b \cdot [\![c]\!]$ is equivalent to each P_i computing $[\![a]\!]_i \leftarrow b \cdot [\![c]\!]_i$.

Given $[a] = g^a \in \mathbb{G}$, and $x \in \mathbb{Z}_N$, we denote by $x[a]$ the value $[a]^x = [ax]$. Likewise, given $[\boldsymbol{a}] \in \mathbb{G}^k$ and $\boldsymbol{x} \in \mathbb{Z}_N^k$, we interpret $\langle [\boldsymbol{a}], \boldsymbol{x}\rangle$ as $[a_1x_1] \cdot [a_2x_2] \ldots [a_kx_k] = [\sum a_ix_i]$. We interpret $\langle [\boldsymbol{a}], [\![\boldsymbol{x}]\!]\rangle$ in the natural way, that is,

each party j computes $\langle [\boldsymbol{a}], [\![\boldsymbol{x}]\!]_j \rangle$. This gives us an output in a secret sharing scheme with multiplicative reconstruction. Denote by $\langle\!\langle b \rangle\!\rangle$ the perfectly hiding secret sharing scheme over cyclic group $\mathbb{G}$ where $b = \prod \langle\!\langle b \rangle\!\rangle_i$.

The following definition of DPF is the multi-party equivalent of [23] and similar to the FSS definition in [6]. Note that for FSS schemes, the indistinguishability and simulatability based definitions are both used and equivalent [6].

Definition 1 (Distributed Point Function (DPF)). *Let $f_{\alpha,\beta}(x) : D \to \mathbb{G}$ be a point function such that $f_{\alpha,\beta}(\alpha) = \beta$ and $f_{\alpha,\beta}(x) = 0$ for $x \neq \alpha$. Let there be a secret sharing scheme with share decoding algorithm* Reconstruct*. A distributed m-party point function is a pair of PPT algorithms* $(\mathsf{Gen}, \mathsf{Eval})$ *such that:*

- $\mathsf{Gen}(1^\kappa, f_{\alpha,\beta}(x))$*, a randomized algorithm that for security parameter κ and point function $f_{\alpha,\beta}(x)$, outputs of set of keys $\{k_1, \ldots, k_m\}$,*
- $\mathsf{Eval}(i, k_i, x)$*, for party P_i, key k_i and input $x \in D$, it outputs $[\![f_{\alpha,\beta}(x)]\!]_i$.*
- ***Correctness:*** *For all $f_{\alpha,\beta}$ and x,*

$$Pr[(k_1, \ldots, k_m) \leftarrow \mathsf{Gen}(1^\kappa, f_{\alpha,\beta}) : \\ \mathsf{Reconstruct}(\mathsf{Eval}(1, k_1, x), \ldots, \mathsf{Eval}(m, k_m, x)) = f_{\alpha,\beta}(x)] = 1$$

- ***Security:*** *For any allowed set of corrupted parties $T \subset \{1, \ldots, m\}$ we can run the following experiment for the following experiment:*
  1. *The adversary gives $(\alpha_0, \beta_0, \alpha_1, \beta_1) \leftarrow \mathcal{A}(1^\kappa)$ where $\alpha_i \in D$ and $\beta_i \in \mathbb{G}$.*
  2. *The challenger samples $b \leftarrow \{0, 1\}$ and sets $(k_1, \ldots, k_m) \leftarrow \mathsf{Gen}(1^\kappa, f_{\alpha_b,\beta_b})$.*
  3. *The adversary guesses $b' \leftarrow \mathcal{A}(\{k_i\}_{i \in T}, \sigma)$ using the corrupted keys.*

  We say that the DPF is T-secure if there exists a negligible function μ such that for all PPT adversaries $\mathcal{A}$ it holds that

$$Pr[b = b'] - \frac{1}{2} \leq \mu(\kappa).$$

3 Schema for DPFs

We first describe the 2-party DPF [6] and then give our multi-party generalization instantiated as our DPF construction in Section 4.

3.1 2-party DPF

First, note that α can be considered as a bit-sequence $\alpha_1, \ldots, \alpha_n$. Hence, the function $f_{\alpha,\beta}$ can be thought of as a n-level binary tree where each edge is labeled by a bit value and the leaf corresponding to input α has the value β and other leaves have 0. We call the path defined by α to be *active* and all other paths *inactive*. A node is active, if it is on the path that is active. Note that sometimes these are called accepting and non-accepting paths.

During Gen, the algorithm traverses the binary tree and, for each level of the tree, computes correction words based on the node where the two children of the node differ - one corresponding to the active and other to the inactive path.

The crucial step in Gen is distinguishing the inactive and active children of the active node. For that, Gen has operators $\mathcal{F}_{1,0}, \mathcal{F}_{1,1}, \mathcal{F}_{2,0}, \mathcal{F}_{2,1}, \ldots \mathcal{F}_{n,0}, \mathcal{F}_{n,1}$, two for each of the n levels. These operators and a secret-shared state $[\![\sigma_\varepsilon]\!]_i$ are the key of party P_i. The operators are known to both parties.

Given the keys, both parties can locally run Eval. During Eval both parties compute secret shared values for each node in the binary tree. If the input value is α then the final outcome is $[\![\beta]\!]$ and otherwise it is $[\![0]\!]$. When evaluating at input x, the parties compute $\mathcal{F}_{n,x_n}(\ldots \mathcal{F}_{2,x_2}(\mathcal{F}_{1,x_1}([\![\sigma_\varepsilon]\!]))\ldots)$.

For the two-party case, the underlying secret sharing scheme is additive scheme over bitstrings, also known as the XOR-sharing where $x = [\![x]\!]_1 \oplus [\![x]\!]_2$. Hence, if both parties have the same share, then the secret is zero and, respectively, if the shares are not equal the the shared value is not zero. Hence, the operators are chosen in such a way that if the shares of both parties become equal after applying some $\mathcal{F}_{i,x_i}$ for $i \leq n$, they will stay equal for any application of any of the subsequent operators $\mathcal{F}_{j,\cdot}$. Put another way, if the shares are equal in some node of the binary tree, the operators are chosen in such a way that for all the descendants of that node, the shares are equal as well. This ensures that for all binary vectors $x \neq \alpha$, it holds that the shares of both parties are equal at the end of the evaluation and the output is 0.

On the other hand, for nodes on the active path, the shares of the two parties are different. There are only $n+1$ such nodes, one for each level. Thus, the operators are chosen so that at node r applying the operator corresponding to next bit b makes the shares of the parties equal if $r||b$ is not the prefix of α but the other operator for bit $1-b$ does not. For security, it is crucial that one party is unable to tell which operator is which.

For [6], the key has two random seeds, two random bits, and two correction words per level. Both parties have the same correction words, but different seeds and bits. During Eval, each party expands their seed to the next level seed and sign bits using a PRG. Both parties apply the correction word corresponding to their sign bit. If the seeds became the same, they always derive the same value and apply the same correction words ensuring that all the following seeds become equal. The initial keys are chosen uniformly randomly, but ensuring that the least significant bit of the seed differs and that the sign bits are different to ensure a non-zero state at the root node.

3.2 Multi-party DPF

We generalize the XOR-shared seed and sign bit from [6] to a shared state $[\![\sigma_\varepsilon]\!]$. Parties locally apply operators to their secret-shared values, like the the PRG and correction value use in [6]. There is some invariant for inactive nodes such that applying the operators will preserve that invariant. In [6], this invariant is that the shares of the two parties are equal but we need something more general to ensure the secret shared state is a zero state.

We say that a node with index r, the last living ancestor is on level i, if the longest common prefix or r and α has length i. We write this as $\mathsf{LLA}(r) = i$. For any inactive r $0 \geq \mathsf{LLA}(a) < n$ and we use it to classify inactive nodes to

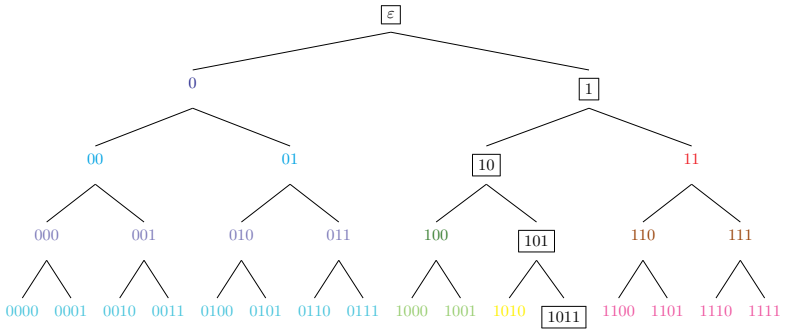


Fig. 1. Predicates $Pred_{i,j}(\cdot)$ for $\alpha = 1011$. Inactive nodes are colored based on their LLA-value (e.g. the blue nodes have $\mathsf{LLA} = 0$). Different tones distinguish the level, e.g, nodes 0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111 where $Pred_{4,0}(\cdot) = 1$ and nodes 110, 111 where $Pred_{3,1}(\cdot) = 1$.

n sets. In principle, we need some invariants to distinguish inactive and active nodes for each level of the tree. In general, we represent these invariants by $\frac{n(n+1)}{2}$ predicates denoted $Pred_{1,0}(\cdot), \ldots, Pred_{n,n-1}(\cdot)$. Predicates $Pred_{i,\cdot}(\cdot)$ are defined for nodes of length i. In general, the predicate $Pred_{i,j}(\cdot)$ is satisfied by nodes r with length i with $\mathsf{LLA}(r) = j$. If there is one predicate per level, we denote it as $Pred_i(\cdot)$. Hence, no predicates hold for node α. Classification of the inactive nodes and invariants are illustrated in Figure 1.

General multi-party DPF has $2n$ deterministic operators $\mathcal{F}_{1,0}, \mathcal{F}_{1,1}, \mathcal{F}_{2,0}, \mathcal{F}_{2,1}, \ldots, \mathcal{F}_{n,0}, \mathcal{F}_{n,1}$. There are $n+2$ domains $D_0, \ldots, D_{n+1}$ where D_{n+1} is equal to the output ring R, and $\mathcal{F}_{i,b} : D_{i-1} \to D_i$ for $i \in [1, n]$ and $b \in \{0, 1\}$. There is a final operator $\mathcal{G} : D_n \to D_{n+1}$. For a bitstring $r = r_1 r_2 \ldots r_i$, we use $\mathcal{F}_{i,r}$ to denote the deterministic operator that is the combination of the operators corresponding to bits of r, concretely $\mathcal{F}_{i,r_i}(\ldots \mathcal{F}_{2,r_2}(\mathcal{F}_{1,r_1}(\cdot))\ldots)$. For initial state $[\![\sigma_\varepsilon]\!]$, we denote the state $\mathcal{F}_{i,r}([\![\sigma_\varepsilon]\!])$ by $[\![\sigma_r]\!]$ and $\mathcal{F}_{i,r}(\sigma_\varepsilon)$ by σ_r. We require the following.

1. The operators $\mathcal{F}_{1,0}, \mathcal{F}_{1,1}, \mathcal{F}_{2,0}, \mathcal{F}_{2,1}, \ldots \mathcal{F}_{n,0}, \mathcal{F}_{n,1}$ and $\mathcal{G}$ can be locally evaluated on the shared state $[\![\sigma]\!]$.
2. $\mathcal{G}(\sigma_\alpha) = \beta$.
3. For r of length $i-1$ that is a prefix of α, $Pred_{i,i-1}(\mathcal{F}_{i,\overline{\alpha_i}}(\sigma_r)) = 1$.
4. If, for r of length j, there exists $i < j$, such that $Pred_{j,i}(\sigma_r) = 1$, then $Pred_{j+1,i}(\mathcal{F}_{i,0}(\sigma_r)) = 1$ and $Pred_{j+1,i}(\mathcal{F}_{i,1}(\sigma_r)) = 1$.
5. For any $i \in [0, n-1]$, if $Pred_{n,i}(\sigma_r) = 1$, then $\mathcal{G}(\sigma_r) = 0$.
6. It is infeasible to win in a decision game to distinguish different values of α and β based on the operators $\mathcal{F}_{1,0}, \mathcal{F}_{1,1}, \mathcal{F}_{2,0}, \mathcal{F}_{2,1}, \ldots \mathcal{F}_{n,0}, \mathcal{F}_{n,1}$ and $\mathcal{G}$.

Intuitively, we need Property 1 to ensure that evaluation is local. Properties 2 and 5 ensure the correct output. Property 3 makes sure that an inactive child of an active node is inactive and, similarly, Property 4 ensures that all

Algorithm 1: General Eval

Input: $\{\mathcal{F}_{i,0}\}_{i=1}^{n}, \{\mathcal{F}_{i,1}\}_{i=1}^{n}, \mathcal{G}, [\![\sigma_\varepsilon]\!], y = y_1 \dots y_k$

1 $[\![\overline{\sigma_0}]\!] \leftarrow [\![\sigma_\varepsilon]\!]$;
2 **for** $i = 1$ *to* n **do**
3 $\quad [\![\overline{\sigma_i}]\!] \leftarrow \mathcal{F}_{i,y_i}([\![\overline{\sigma_{i-1}}]\!])$;
4 $[\![b]\!] \leftarrow \mathcal{G}([\![\overline{\sigma_n}]\!])$;
5 **return** $[\![b]\!]$;

children of inactive nodes are inactive. Property 6 formulates the basic security requirement that the privacy of α is ensured during evaluation. Theorem 1 establishes that fulfilling these properties gives a DPF scheme. In the evaluation GeneralEval in Algorithm 1 we use the notation of additive secret sharing, however, in the general case, the linear sharing scheme for each level could vary.

Theorem 1. *An algorithm defined by operators* $\mathcal{F}_{1,0}, \mathcal{F}_{1,1}, \mathcal{F}_{2,0}, \mathcal{F}_{2,1}, \dots \mathcal{F}_{n,0}$, $\mathcal{F}_{n,1}$, $\mathcal{G}$, *predicates* $Pred_{1,0}(\cdot), \dots, Pred_{n,n-1}(\cdot)$ *and shared state* $[\![\sigma]\!]$ *satisfying Properties 1–5 defines a correct DPF with evaluation Algorithm 1. If it has Property 6 and uses a perfectly hiding secret sharing scheme, then it is a secure DPF.*

Proof. Define $\overline{\sigma_r} = \mathcal{F}_{i,r}(\overline{\sigma_0}) = \mathcal{F}_{i,r_i}(\dots \mathcal{F}_{2,r_2}(\mathcal{F}_{1,r_1}(\overline{\sigma_0}))\dots)$ for a bitstring r with length i. First, if r is a bitstring of length i that is a prefix of α, then $\overline{\sigma_i} = \sigma_r$. Indeed, this holds for $i = 0$ in line 1 in Algorithm 1. Then, by induction, suppose that the claim holds for $i = \ell$. Let r be a bitstring of length $\ell + 1$. Write r as $r' \| b$ where r' consists of the first ℓ bits of r and b is the last bit. Naturally r' is also a prefix of α, thus $\overline{\sigma_\ell} = \sigma_{r'}$. We have that

$$\overline{\sigma_{\ell+1}} = \mathcal{F}_{\ell+1,b}(\overline{\sigma_\ell}) = \mathcal{F}_{\ell+1,\alpha_{\ell+1}}(\overline{\sigma_\ell}) = \mathcal{F}_{m+1,\alpha_{\ell+1}}(\sigma_{r'}) = \sigma_{r'\|b} = \sigma_r.$$

Secondly, for $y = \alpha$, $\overline{\sigma_n} = \sigma_\alpha$. By Property 2, evaluating General Eval at α outputs $[\![\beta]\!]$. Thirdly, we show by induction over ℓ with $j = i + \ell$, that if for a node r of length j that is not a prefix of α, we have $\mathsf{LLA}(r) = i$, then $Pred_{j,i}(\overline{\sigma_r}) = 1$. Fix i. If $\ell = 1$, then this follows directly from Property 3. Now suppose it holds for some $\ell = k$. Let r be of length $j = i + k + 1$ where it holds that $\mathsf{LLA}(r) = i$. Note that we can write r as $r' \| b$ where r' consists of the first $i + k$ bits of r and b is the last bit. In that case, $\mathsf{LLA}(r') = i$ and r' has length $i + k$. By the induction assumption, $Pred_{i+k,i}(\overline{\sigma_{r'}}) = 1$. By Property 4, $Pred_{j,i}(\overline{\sigma_r}) = 1$.

Fourth, we note that this means that for all inactive leaves of the tree, that is, nodes r with length n such that $r \neq \alpha$, there exists i, such that $Pred_{n,i}(\overline{\sigma_r}) = 1$. Thus, by Property 5, for inputs $y \neq \alpha$, the output is 0.

According to Property 1, all parties can evaluate GeneralEval Algorithm 1 locally. Hence, if a generation algorithm satisfies Properties 1–5, then it gives a correct DPF. For security, the difference between Property 6 and the real game in the DPF security definition is that, in the real game, the parties are also given secret shares of the root state σ_ε. However, given a perfect secret sharing scheme, these secret shares produce no information for an adversary controlling the set

of parties for which the secret sharing scheme ensures privacy. Thus, given a perfectly hiding secret sharing scheme, it suffices to only consider the operators in the security proofs and Property 6 ensures security. □

4 Construction

In this section we describe our multi-party DPF scheme. First, we give a rough overview of the construction. Then we introduce some algorithms for some useful tools. We then give the key generation algorithm in two parts. In the first part of the key generation, we describe how to generate matrices with certain properties. In the second part, we explain how to build the DPF keys from these matrices. Then, we give the relatively simple evaluation algorithm. Finally, we prove the correctness of our scheme by showing that the conditions of Theorem 1 hold.

4.1 Rough outline

Our construction works over the ring $R = \mathbb{Z}_N$ where N is a product of two large primes. In the security proof we require that some values cannot be computed without factoring N, and thus we have to double the number of certain variables. However, doubling clutters the construction. Thus, we give the full construction in Appendix C.3. For the sake of simplicity and clarity, we here provide the construction where those variables are not doubled making the overall idea easier to follow. Note that it could be that the security of our scheme could also be proven for other structures than $\mathbb{Z}_N$, but we leave that for future work.

Our algorithm according to the multi-party DPF schema in Section 3.2
Our secret state $[\![\sigma]\!]$ is a secret-shared vector $\boldsymbol{\nu} \in \mathbb{Z}_N^k$. For every $i \in [1, n]$, we have that the operators $\mathcal{F}_{i,0}$ and $\mathcal{F}_{i,1}$ are left-multiplying the vector $[\![\boldsymbol{\nu}]\!]$ by specially constructed public matrices $A_{i,0}$ and $A_{i,1}$, respectively.

At the beginning, we sample a secret set of independent vectors $\boldsymbol{c_1}, \ldots, \boldsymbol{c_{\tau+n}} \in (\mathbb{Z}_N)^k$. We denote the module spanned by $\boldsymbol{c_1}, \ldots, \boldsymbol{c_{\tau+j}}$ by $\tilde{C}_j$ for $j \in [0, n]$. We also randomly sample $n + 1$ invertible matrices F_i for $i \in [0, n]$. The invariant $Pred_i(\cdot)$ for level $i \in \{1, \ldots, n\}$ is that $\boldsymbol{\nu} \in F_i\tilde{C}_i$. We also sample a set of vectors $\boldsymbol{d_0}, \boldsymbol{d_1}, \ldots, \boldsymbol{d_{3n}} \in (\mathbb{Z}_N)^k$. On the active path, we will have that $\boldsymbol{\nu} \in \boldsymbol{d_{3i}} + F_i\tilde{C}_i$

The operator $\mathcal{G}$ is multiplication by a vector $\boldsymbol{e}$ that is in the kernel of $F_n\tilde{C}_n$. We have to put this vector into the exponent in some group $\mathbb{G}$ with size N where DDH-type problems are hard. We give more detailed reasoning in Appendix D, but in short, if we do not do this, then it is possible to compute 2^n vectors corresponding to the leaves of the tree in such a way that if we choose k of those vectors corresponding to some leaves and the active leaf is among those leaves, then these matrices are full rank, but if the active leaf is not among those leaves, then the matrix has rank $k - 1$. Thus, it would be possible to compute the secret alive leaf. Putting that vector $\boldsymbol{e}$ into the exponent means that the 2^n computed vectors are in the exponent and now performing the same attack requires distinguishing whether a matrix in the exponent is full-rank or not, which is generally not possible in groups where DDH-style problems are hard.

Goals of the construction We need a construction where, at first, the parties are given some secret-shared vector $[\![\boldsymbol{z}_0]\!]$. Then, for active path on level i, we need to have the sharing of $\boldsymbol{z}_{i-1}$. Multiplying this secret-shared vector by A_{i,α_i} gives $\boldsymbol{z}_i$, but multiplying $\boldsymbol{z}_{i-1}$ by $A_{i,\overline{\alpha_i}}$ gives an element of $\tilde{C}'_i$. We also want that both A_{i,α_i} and $A_{i,\overline{\alpha_i}}$ map the submodule $\tilde{C}'_{i-1}$ into $\tilde{C}'_i$. This will guarantee that, at the end, for every inactive leaf, we have a secret-shared value of $\tilde{C}'_n$, but at the active leaf, we have $\boldsymbol{z}_n$, which is not a member of that submodule. This requires that $A_{i,\overline{\alpha_i}}$ maps both $\boldsymbol{z}_{i-1}$ and $\tilde{C}'_{i-1}$ into $\tilde{C}'_i$ but that A_{i,α_i} does not map $\boldsymbol{z}_{i-1}$ to $\tilde{C}'_i$. This is easy to obtain if $\boldsymbol{z}_{i-1}$ is a sum of some vector $\boldsymbol{d_i} \in \ker A_{i,\overline{\alpha_i}}$ and an element of $\tilde{C}'_{i-1}$ (and where also $\boldsymbol{d_i} \notin \ker A_{i,\alpha_i}$).

We could sample the submodules $\tilde{C}'_i$ randomly and sample the matrices A_{i,α_i} and $A_{i,\overline{\alpha_i}}$ accordingly so that these properties hold. However, this leads to a large number of matrices and secret submodules with complicated relations to each other. We instead choose to use submodules $\tilde{C}_i$ that are contained in each other and to build the scheme from two matrices with certain properties. We will show slightly later how $\tilde{C}'_i$ and $\tilde{C}_i$ are related. More specifically, we want to build two matrices B_0 and B_1 and that would map all the $\tilde{C}_{i-1}$ into $\tilde{C}_i$ and also satisfy some other properties that we will argue for now. We want that for $i \in [0, n-1]$, B_0 maps the $\boldsymbol{d_{3i+1}}$ into $\tilde{C}_i$, but for B_1 it holds that $B_1\boldsymbol{d_{3i+1}} \in \boldsymbol{d_{3i+2}} + \tilde{C}_i$.

We obviously cannot just have that $A_{i,\overline{\alpha_i}} = B_0$ and $A_{i,\alpha_i} = B_1$ as this would trivially leak α_i. A very easy hiding mechanism would be to use the random invertible matrices F_i, and then have that $A_{i,\overline{\alpha_i}} = F_i B_0 F_{i-1}^{-1}$ and $A_{i,\alpha_i} = F_i B_0 F_{i-1}^{-1}$ and have that $\tilde{C}'_i = F_i \tilde{C}_i$. We note that left-and-right multiplying matrices by random matrices is also a technique used in code-equivalence assumptions.

We will have $\tilde{C}'_i = F_i \tilde{C}_i$, however, we note that the F_i is repeated on two consecutive levels and for B_0 and B_1. In code equivalence problems, reusing the secret hiding matrices can make the scheme insecure, as shown in [26]. Although B_0 and B_1 do not describe equivalent codes, and thus that attack directly does not apply, we still prefer to hide them somewhat more. For that, we additionally sample matrices E_i and $\overline{E}_i$ such that they too preserve the respective secret submodules, that is, $E_i \tilde{C}_{i-1} = \tilde{C}_{i-1}$ and $\overline{E}_i \tilde{C}_i = \tilde{C}_i$, and that they map $\boldsymbol{d_{3(i-1)}}$ to $\boldsymbol{d_{3(i-1)+1}} + \tilde{C}_{i-1}$ and $\boldsymbol{d_{3(i-1)+2}}$ to $\boldsymbol{d_{3i}} + \tilde{C}_i$, respectively, thus preserving the properties required for the correctness of the scheme.

We note that we could now write this in a more concise way, with $\begin{pmatrix} A_{i,0} \\ A_{i,1} \end{pmatrix} \leftarrow \begin{pmatrix} F_i & 0 \\ 0 & F_i \end{pmatrix} \begin{pmatrix} \overline{E}_i & 0 \\ 0 & \overline{E}_i \end{pmatrix} \begin{pmatrix} B_{\pi_i(0)} \\ B_{\pi_i(1)} \end{pmatrix} E_i F_{i-1}^{-1}$. However, we note that we do not need to make the $\overline{E}_i$ same for B_0 and B_1, and that we can also somewhat mix B_0 and B_1, as far as some constraints are satisfied. Thus, we obtain $\begin{pmatrix} A_{i,0} \\ A_{i,1} \end{pmatrix} \leftarrow \begin{pmatrix} F_i & 0 \\ 0 & F_i \end{pmatrix} \begin{pmatrix} \overline{E}_{i,1+3\pi_i(0)} & \overline{E}_{i,2+\pi_i(0)} \\ \overline{E}_{i,2+\pi_i(1)} & \overline{E}_{i,1+3+\pi_i(1)} \end{pmatrix} \begin{pmatrix} B_{\pi_i(0)} \\ B_{\pi_i(1)} \end{pmatrix} E_i F_{i-1}^{-1}$.

For the last step we need that all elements of $\tilde{C}'_i = F_n \tilde{C}_n$ would be mapped to zero and that $F_n \boldsymbol{d_{3n}} \notin F_n \tilde{C}_n$ would be mapped to β. For this we can sample

a random vector $\boldsymbol{e}$ from $\tilde{C}'^{\perp}_n$ and compute the scalar product with the secret-shared vector $\boldsymbol{z}_n$. This would take care about all of the inactive leafs. For the active leaf, we note that as long as $\langle \boldsymbol{z}_n, \boldsymbol{e} \rangle$ is invertible (as will happen with an overwhelming probability), it is easy to obtain the output as β as we can rescale the original input $\boldsymbol{z}_0$ by a desired coefficient. For security reasons more discussed in Appendix D, we put the $\boldsymbol{e}$ into the exponent of the group $\mathbb{G}$. In the next section we describe this more concretely.

Sampling matrices with right properties Algorithm 2 uses MatFromVecPairs for a set of independent vectors $\{\boldsymbol{x_j}\}_{j=1}^{k'}$ and$\{\boldsymbol{y_j}\}_{j=1}^{k'}$ to obtain a matrix mapping each $\boldsymbol{x_j}$ to $\boldsymbol{y_j}$ and sampleFrom for sampling randomly from a submodule. After sampling the $\{\boldsymbol{c_j}\}_{j=1}^{\tau+n}$ and the $\{\boldsymbol{d_j}\}_{j=0}^{3n}$, we sample $\overline{\boldsymbol{c}}_{\boldsymbol{3i-1,0}}, \overline{\boldsymbol{c}}_{\boldsymbol{3i-1,1}}, \overline{\boldsymbol{c}}_{\boldsymbol{3i,2}}$, $\overline{\boldsymbol{c}}_{\boldsymbol{3i,4}}, \overline{\boldsymbol{c}}_{\boldsymbol{3i+1}} \xleftarrow{\$} \tilde{C}_i$ for $i \in [0, n]$ and matrices $B_0, B_1 \in (\mathbb{Z}_N)^{k \times k}$ such that for all $i \in [1, n]$, $B_0\tilde{C}_{i-1}, B_1\tilde{C}_{i-1} \subseteq \tilde{C}_i$, $B_0\boldsymbol{d}_{\boldsymbol{3(i-1)+1}} = \overline{\boldsymbol{c}}_{\boldsymbol{3(i-1)+2,0}}$ and $B_1\boldsymbol{d}_{\boldsymbol{3(i-1)+1}} = \boldsymbol{d}_{\boldsymbol{3(i-1)+2}} + \overline{\boldsymbol{c}}_{\boldsymbol{3(i-1)+2,1}}$. We then, sample matrices E_i from $(\mathbb{Z}_N)^{k \times k}$ such that for all $i \in [1, n]$, we have $E_i\tilde{C}_{i-1} = \tilde{C}_{i-1}$ and $E_i\boldsymbol{d}_{\boldsymbol{3(i-1)}} = \boldsymbol{d}_{\boldsymbol{3(i-1)+1}} + \overline{\boldsymbol{c}}_{\boldsymbol{3(i-1)+1}}$.

The space of possibilities for choosing such matrices E_i is rather large, as the latter condition gives only one constraint. At first glance, one may think that the first condition creates $\tau + i$ constraints, it is actually less binding, as we can choose into which elements of $\tilde{C}_{i-1}$ the basis vectors of $\tilde{C}_{i-1}$ map to. More specifically, for s constraints that require that we map some vector to some other vector, there are roughly $N^{k \times (k-s)}$ matrices that satisfy these constraints. For a constraint that requires that we map we map s vectors to some s-dimensional submodule, there are roughly $N^{k \times (k-s)+s^2}$ matrices satisfying these constraints.

For $i \in [1, n]$ sample matrices $\overline{E}_{i,1}, \overline{E}_{i,2}, \overline{E}_{i,3}, \overline{E}_{i,4}$ from $(\mathbb{Z}_N)^{k \times k}$ such that for all $i \in [1, n]$ and $c \in [1, 4]$ we have that $\overline{E}_{i,c}\tilde{C}_i = \tilde{C}_i$ and additionally, that $\overline{E}_{i,4}\boldsymbol{d}_{\boldsymbol{3(i-1)+2}} = \boldsymbol{d}_{\boldsymbol{3i}} + \overline{\boldsymbol{c}}_{\boldsymbol{3i,c}}$ and for $c = 2$, $\overline{E}_{i,2}\boldsymbol{d}_{\boldsymbol{3(i-1)+2}} = \overline{\boldsymbol{c}}_{\boldsymbol{3i,2}}$. The reason behind these last two constraints is that the middle part of A_{i,α_i} is equal to $\overline{E}_{i,3}B_0 + \overline{E}_{i,4}B_1$ and that the middle part of $A_{i,\overline{\alpha_i}}$ is equal to $\overline{E}_{i,1}B_0 + \overline{E}_{i,2}B_1$. We will want that for A_{i,α_i}, this middle part would map $\boldsymbol{d}_{\boldsymbol{3(i-1)+1}}$ to $\boldsymbol{d}_{\boldsymbol{3i}} + \tilde{C}_i$ and that for $A_{i,\overline{\alpha_i}}$, this middle part would map $\boldsymbol{d}_{\boldsymbol{3(i-1)+1}}$ to $\tilde{C}_i$.

As B_0 already maps $\boldsymbol{d}_{\boldsymbol{3(i-1)+1}}$ to $\tilde{C}_i$, we do not have to specify anything for $\overline{E}_{i,1}, \overline{E}_{i,3}$. B_1, however, maps $\boldsymbol{d}_{\boldsymbol{3(i-1)+1}}$ to $\boldsymbol{d}_{\boldsymbol{3(i-1)+2}} + \tilde{C}_i$. Thus we have to specify that $\overline{E}_{i,3}$ maps $\boldsymbol{d}_{\boldsymbol{3(i-1)+2}}$ to $\tilde{C}_i$ and $\overline{E}_{i,4}$ maps $\boldsymbol{d}_{\boldsymbol{3(i-1)+2}}$ to $\boldsymbol{d}_{\boldsymbol{3i}} + \tilde{C}_i$. Suppose that our active leaf is α with bits $\alpha_1, \ldots, \alpha_n$. If $\alpha_i = 1$, then denote $\pi_i \leftarrow \pi_{id}$, else denote $\pi_i \leftarrow \pi_{sw}$.

Set $\begin{pmatrix} A_{i,0} \\ A_{i,1} \end{pmatrix} \leftarrow \begin{pmatrix} F_i & 0 \\ 0 & F_i \end{pmatrix} \begin{pmatrix} \overline{E}_{i,1+3\pi_i(0)} & \overline{E}_{i,2+\pi_i(0)} \\ \overline{E}_{i,2+\pi_i(1)} & \overline{E}_{i,1+3+\pi_i(1)} \end{pmatrix} \begin{pmatrix} B_{\pi_i(0)} \\ B_{\pi_i(1)} \end{pmatrix} E_i F_{i-1}^{-1}$. This construction guarantees that $A_{i,\alpha_i} = F_i\overline{E}_{i,3}B_0E_iF_{i-1}^{-1} + F_i\overline{E}_{i,4}B_1E_iF_{i-1}^{-1}$ and $A_{i,\overline{\alpha_i}} = F_i\overline{E}_{i,1}B_0E_iF_{i-1}^{-1} + F_i\overline{E}_{i,2}B_1E_iF_{i-1}^{-1}$, which both makes $A_{i,0}$ and $A_{i,1}$ to map $\tilde{C}_{i-1}$ into $\tilde{C}_i$ and that makes A_{i,α_i} to map $F_{i-1}\boldsymbol{d}_{\boldsymbol{3(i-1)}}$ to $\boldsymbol{d}_{\boldsymbol{3i}} + F_i\tilde{C}_i$ and that makes $A_{i,\overline{\alpha_i}}$ to map $F_{i-1}\boldsymbol{d}_{\boldsymbol{3(i-1)}}$ to $F_i\tilde{C}_i$, thus making sure that the right invariants hold for the leaves of the tree.

Algorithm 2: SampleMatrices

Input: N, k, n, τ, u

$\boldsymbol{c_1}, \ldots, \boldsymbol{c_{\tau+n}} \xleftarrow{\$} (\mathbb{Z}_N)^k$; /* Basis elements for secret subspaces */

$\boldsymbol{d_0}, \boldsymbol{d_1}, \ldots, \boldsymbol{d_{3n}} \xleftarrow{\$} (\mathbb{Z}_N)^k$; /* Kernel vectors for B_0 */

for $i = 0, i++, i \leq n$ **do**

  $\overline{\boldsymbol{c}}_{\boldsymbol{3i-1,1}}, \overline{\boldsymbol{c}}_{\boldsymbol{3i-1,2}}, \overline{\boldsymbol{c}}_{\boldsymbol{3i-1,3}}, \overline{\boldsymbol{c}}_{\boldsymbol{3i-1,4}}, \overline{\boldsymbol{c}}_{\boldsymbol{3i}}, \overline{\boldsymbol{c}}_{\boldsymbol{3i+1,0}}, \overline{\boldsymbol{c}}_{\boldsymbol{3i+1,1}} \leftarrow$ $\mathsf{sampleFrom}(\langle \boldsymbol{c_1}, \ldots, \boldsymbol{c_{\tau+i}} \rangle, 7)$; /* Subspace vectors added to the kernel vectors */

for $i = 0, i++, i \leq \tau + n$ **do**

  $\boldsymbol{x'_{i,0}}, \boldsymbol{x'_{i,1}} \leftarrow \boldsymbol{c_i}$; /* Vectors $\boldsymbol{x'_{i,b}}$ must be mapped to something specific by B_0 and B_1, e.g the $\boldsymbol{c_i}$. */

for $i = \tau + n + 1, i++, i \leq \tau + 4n$ **do**

  $\boldsymbol{x'_{i,0}} \leftarrow \boldsymbol{d_i}$

for $i = 0, i++, i \leq n$ **do**

  $\boldsymbol{x'_{\tau+n+1+i,1}} \leftarrow \boldsymbol{d_{3(i-1)+1}}$

; /* Compute the vectors $\boldsymbol{y'_{i,b}}$ that the $\boldsymbol{c_i}$ and the $\boldsymbol{d_i}$ should be mapped to by B_0 and B_1. */

; /* First the ones that characterize what the $\boldsymbol{c_i}$ should be mapped to using Algorithm 6. */

for $i = 0, i++, i \leq \tau$ **do**

  $\boldsymbol{y'_{i,0}}, \boldsymbol{y'_{i,1}} \leftarrow \mathsf{sampleFrom}(\langle \{\boldsymbol{c_i}\}_{j=1}^{\tau+1} \rangle, 2)$

for $i = \tau + 1, i++, i \leq \tau + n$ **do**

  $\boldsymbol{y'_{i,0}}, \boldsymbol{y'_{i,1}} \leftarrow \mathsf{sampleFrom}(\langle \{\boldsymbol{c_i}\}_{j=1}^{i+1} \rangle, 2)$

; /* B_0 should map $\boldsymbol{d_{3(i-1)+1}}$ to $\overline{\boldsymbol{c}}_{\boldsymbol{3(i-1)+2,0}}$. */

for $i = \tau + n + 1, i++, i \leq \tau + 4n$ **do**

  $\boldsymbol{y'_{i,0}} \leftarrow \overline{\boldsymbol{c}}_{\boldsymbol{3(i-1)+2,0}} \in \mathbb{Z}_N^k$

; /* B_1 should map $\boldsymbol{d_{3(i-1)+1}}$ to $\boldsymbol{d_{3(i-1)+2}} + \overline{\boldsymbol{c}}_{\boldsymbol{3(i-1)+2,1}}$. */

for $i = 0, i++, i \leq n$ **do**

  $\boldsymbol{y'_{\tau+n+1+i,1}} \leftarrow \boldsymbol{d_{3(i-1)+2}} + \overline{\boldsymbol{c}}_{\boldsymbol{3(i-1)+2,1}}$

; /* Create B_0 and B_1 with Algorithm 5. */

$B_0 \leftarrow \mathsf{MatFromVecPairs}(\mathsf{N}, \mathsf{k}, \{\boldsymbol{x'_{j,0}}\}_{\mathsf{j}=1}^{\tau+4\mathsf{n}}, \{\boldsymbol{y'_{j,0}}\}_{\mathsf{j}=1}^{\tau+4\mathsf{n}}, \mathsf{u})$

$B_1 \leftarrow \mathsf{MatFromVecPairs}(\mathsf{N}, \mathsf{k}, \{\boldsymbol{x'_{j,1}}\}_{\mathsf{j}=1}^{\tau+4\mathsf{n}}, \{\boldsymbol{y'_{j,1}}\}_{\mathsf{j}=1}^{\tau+4\mathsf{n}}, \mathsf{u})$

; /* Compute the matrices $E_{i,j}$ and $\overline{E}_i$. Define the vectors that should be mapped to something by $E_{i,j}$ and $\overline{E}_i$. First the ones that describe that $\tilde{C}_i$ should be mapped to itself. */

for $i = 0, i++, i \leq n$ **do**

  for $j = 0, j++, j \leq \tau + i - 1$ **do**

    $\boldsymbol{x''_{i,j}}, \boldsymbol{x'''_{i,j,1}}, \boldsymbol{x'''_{i,j,2}}, \boldsymbol{x'''_{i,j,3}}, \boldsymbol{x'''_{i,j,4}}, \leftarrow \boldsymbol{c_j}$

  $\boldsymbol{x'''_{i,\tau+i,1}}, \boldsymbol{x'''_{i,\tau+i,2}}, \boldsymbol{x'''_{i,\tau+i,3}}, \boldsymbol{x'''_{i,\tau+i,4}} \leftarrow \boldsymbol{c_{\tau+i}}$

; /* Then the one about the kernel vector. */

for $i = 0, i++, i \leq n$ **do**

  $\boldsymbol{x''_{i,\tau+i}}, \leftarrow \boldsymbol{d_{3(i-1)}}$

  $\boldsymbol{x'''_{i,\tau+i+1,1}}, \boldsymbol{x'''_{i,\tau+i+1,2}}, \boldsymbol{x'''_{i,\tau+i+1,3}}, \boldsymbol{x'''_{i,\tau+i+1,4}} \leftarrow \boldsymbol{d_{3(i-1)+2}}$

Algorithm 2: Sampling matrices (continued)

; /* Define the vectors that must be mapped to the abovegiven vectors, first the ones that are for $E_{i,b}\tilde{C}_i = \tilde{C}_i$ */

for $i = 0, i++, i \leq n$ **do**
  for $i' = 0, i'++, i' \leq \tau + i - 1$ **do**
    $\boldsymbol{y''_{i,i'}} \leftarrow \mathsf{sampleFrom}(\langle \{\boldsymbol{c_i}\}_{j=1}^{\tau+i-1} \rangle, 1)$
    $\boldsymbol{y'''_{i,i',1}}, \boldsymbol{y'''_{i,i',2}}, \boldsymbol{y'''_{i,i',3}}, \boldsymbol{y'''_{i,i',4}} \leftarrow \mathsf{sampleFrom}(\langle \{\boldsymbol{c_i}\}_{j=1}^{\tau+i} \rangle, 4)$
    $\boldsymbol{y'''_{i,i',4}} \leftarrow \sum_{j=0}^{\tau+i} \lambda''_{i,i',j,4} \boldsymbol{c_j}$
  $\boldsymbol{y''_{i,\tau+i}} \leftarrow \mathsf{sampleFrom}(\langle \{\boldsymbol{c_i}\}_{j=1}^{\tau+i-1} \rangle, 1)$
  $\boldsymbol{y'''_{i,i',1}}, \boldsymbol{y'''_{i,i',2}}, \boldsymbol{y'''_{i,i',3}}, \boldsymbol{y'''_{i,i',4}} \leftarrow \mathsf{sampleFrom}(\langle \{\boldsymbol{c_i}\}_{j=1}^{\tau+i} \rangle, 4)$
for $i = 0, i++, i \leq n$ **do**
  for $c \in \{1, 3\}$ **do**
    $\boldsymbol{y'''_{i,\tau+i,c}} \xleftarrow{\$} \mathbb{Z}_N^k$
  $\boldsymbol{y'''_{i,\tau+i,2}} \leftarrow \overline{\boldsymbol{c}}_{\boldsymbol{3i}}$
  $\boldsymbol{y'''_{i,\tau+i,4}} \leftarrow \boldsymbol{d_{3i}} + \overline{\boldsymbol{c}}_{\boldsymbol{3i,4}}$
  $\boldsymbol{y''_{i,\tau+i,3}} \leftarrow \boldsymbol{d_{3(i-1)+1}} + \overline{\boldsymbol{c}}_{\boldsymbol{3(i-1)+1}}$
for $i = 0, i++, i \leq n$ **do**
  for $c \in [1, 4]$ **do**
    $\overline{E}_{i,c} \leftarrow \mathsf{MatFromVecPairs}(\mathsf{N}, \mathsf{k}, \{\boldsymbol{x'''_{i,j,c}}\}_{\mathsf{j}=1}^{\tau+\mathsf{i}+1}, \{\boldsymbol{y'''_{i,j,c}}\}_{\mathsf{j}=1}^{\tau+\mathsf{i}+1}, \mathsf{u})$
  $E_i \leftarrow \mathsf{MatFromVecPairs}(\mathsf{N}, \mathsf{k}, \{\boldsymbol{x''_{i,j}}\}_{\mathsf{j}=1}^{\tau+\mathsf{i}}, \{\boldsymbol{y''_{i,j}}\}_{\mathsf{j}=1}^{\tau+\mathsf{i}}, \mathsf{u})$
return $B_0, B_1, \{\overline{E}_{i,c}\}_{i=1,c=1}^{n,4}, \{E_i\}_{i=1}^{n}, \{\boldsymbol{c_i}\}_{i=0}^{\tau+n}, \{\boldsymbol{d_i}\}_{i=0}^{3n}$;

Finally, we need to sample a vector $\boldsymbol{e}$ that maps the $F_n\tilde{C}_n$ to 0 and to take care that the value $\boldsymbol{z}_n = A_{n,\alpha_n} A_{n-1,\alpha_{n-1}} \dots A_{1,\alpha_1} \boldsymbol{z}_0$ at the active leaf is mapped to β. This is easy to achieve, as long as $\langle \boldsymbol{z}_n, \boldsymbol{e} \rangle$ is invertible, as it is with an overwhelming probability. Indeed, suppose that we have reached so far with our construction, and we have that $\langle \boldsymbol{z}_n, \boldsymbol{e} \rangle = \gamma$. Now, since all our operations are linear, if we rescale $\boldsymbol{z}_0$ by $\gamma^{-1}\beta$, then after rescaling have $\langle \boldsymbol{z}_n, \boldsymbol{e} \rangle = \beta$, as desired. Note also, that we do not have to have any constraints for what the $\overline{E}_{n,c}$ map $\boldsymbol{d_{3n-1}}$ to, for $c \in \{1, 3, 4\}$. We still need that $\overline{E}_{n,2}\boldsymbol{d_{3n-1}} \in \tilde{C}_n$ though.

4.2 Key generation

Algorithm 2 defines how to sample the matrices needed in the multi-party DPF key generation construction in Algorithm 3. Here N is the size of $\mathbb{Z}_N$, k is the side length of square matrices, n is the depth of the tree, τ is the dimension of the smallest invariant submodule $\tilde{C}_0$ and v is the kernel dimension of B_0 and B_1.

Algorithm 3: Key Generation

Input: $N, k, n, \tau, \alpha = \alpha_1, \ldots, \alpha_n, \beta, u, \mathbb{G}$

$B_0, B_1, \{\overline{E}_{i,c}\}_{i=1,c=1}^{n,4}, \{E_i\}_{i=1}^{n}, \{\boldsymbol{c_i}\}_{i=0}^{\tau+n}, \{\boldsymbol{d_i}\}_{i=0}^{3n} \leftarrow$ $SampleMatrices(N, k, n, \tau, v, u)$

for $i = 0, i++, i \leq n$ **do**

  $F_i \xleftarrow{\$} ((\mathbb{Z}_N)^{k \times k})^*$; /* F_i should be invertible */

$\boldsymbol{z}_0 \leftarrow F_i \boldsymbol{d_0}$

for $i = 0, i++, i \leq n$ **do**

  if $\alpha_i = 1$ **then**

    $\pi_i \leftarrow \pi_{id}$

  else

    $\pi_i \leftarrow \pi_{sw}$

  $\begin{pmatrix} A_{i,0} \\ A_{i,1} \end{pmatrix} \leftarrow \begin{pmatrix} F_i & 0 \\ 0 & F_i \end{pmatrix} \begin{pmatrix} \overline{E}_{i,1+3\pi_i(0)} & \overline{E}_{i,2+\pi_i(0)} \\ \overline{E}_{i,2+\pi_i(1)} & \overline{E}_{i,1+3+\pi_i(1)} \end{pmatrix} \begin{pmatrix} B_{\pi_i(0)} \\ B_{\pi_i(1)} \end{pmatrix} E_i F_{i-1}^{-1}$

Compute $\boldsymbol{e}$ such that $\langle \boldsymbol{e}, F_n \boldsymbol{c_i} \rangle = 0$ for all $i \in [1, \tau + n]$;

Compute $\gamma \leftarrow \langle \boldsymbol{e}, F_n \boldsymbol{d_{3n}} \rangle$;

If γ is not invertible, start again, else $\boldsymbol{z}_0 \leftarrow \boldsymbol{z}_0 \cdot \beta \cdot \gamma^{-1}$;

Compute a secret-sharing $[\![\boldsymbol{z}_0]\!]$ of $\boldsymbol{z}_0$;

Compute $[\boldsymbol{e}]$;

return $\{A_{i,0}\}_{i=1}^{n}, \{A_{i,1}\}_{i=1}^{n}, [\boldsymbol{e}], [\![\boldsymbol{z}_0]\!]$;

Algorithm 4: Eval

Input: $\{A_{i,0}\}_{i=1}^{n}, \{A_{i,1}\}_{i=1}^{n}, [\boldsymbol{e}], [\![\boldsymbol{z}_0]\!], , y = y_1 \ldots y_n$

$[\![\overline{\boldsymbol{z}}_0]\!] \leftarrow [\![\boldsymbol{z}_0]\!]$;

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

  $[\![\overline{\boldsymbol{z}}_i]\!] \leftarrow A_{i,y_i} \cdot [\![\overline{\boldsymbol{z}}_{i-1}]\!]$;

$\langle\!|b|\!\rangle \leftarrow \langle [\boldsymbol{e}], [\![\overline{\boldsymbol{z}}_n]\!] \rangle$;

return $\langle\!|b|\!\rangle$;

4.3 Evaluation

The evaluation operation Eval is given in Algorithm 4 using the secret shared $\boldsymbol{z}_0$ and the public parts of the state. For each level in Eval, the parties have a secret-shared vector of k elements of $\mathbb{Z}_N$, denoted by $\overline{\boldsymbol{z}}_i$. The parties get their share of $[\![\overline{\boldsymbol{z}}_0]\!]$ as part of their key. In the context of the schema, the vector $[\![\overline{\boldsymbol{z}}_0]\!]$ is $[\![\sigma_0]\!]$ and the multiplication $A_{i,y_i} \cdot [\![\overline{\boldsymbol{z}}_{i-1}]\!]$ is the operator to get the next level. Finally, $\left\langle [\boldsymbol{e}], [\![\overline{\boldsymbol{z}}_n{}^{(1)}]\!] \right\rangle$ is the operator to get the desired output so that multiplying the locally computed outputs of all parties give the reconstructed output.

4.4 Correctness proof

We prove helper lemmata and then show the properties required by the schema in Section 3.2 to ensure correctness according to Theorem 1.

Lemma 1. *For all* $i \in [1,n]$, $b \in \{0,1\}$ $B_b\tilde{C}_{i-1} \subseteq \tilde{C}_i$, $E_i\tilde{C}_{i-1} \subseteq \tilde{C}_{i-1}$ *and for* $c \in \{1,2,3,4\}$ $\overline{E}_{i,c}\tilde{C}_i \subseteq \tilde{C}_i$.

Proof. This follows directly from the construction of those matrices. □

Lemma 2. *For all* $i \in [1,n]$, $b \in \{0,1\}$ $A_{i,b}F_{i-1}\tilde{C}_{i-1} \subseteq F_i\tilde{C}_i$.

Proof. Suppose that $\boldsymbol{x'} \in F_{i-1}\tilde{C}_{i-1}$. Denote by $\boldsymbol{x} = F_{i-1}^{-1}\boldsymbol{x'}$, that is, $\boldsymbol{x'} = F_{i-1}\boldsymbol{x}$, hence $\boldsymbol{x} \in \tilde{C}_{i-1}$. We can write

$$\begin{pmatrix} A_{i,0} \\ A_{i,1} \end{pmatrix} = \begin{pmatrix} F_i & 0 \\ 0 & F_i \end{pmatrix} \begin{pmatrix} \overline{E}_{i,1+3\pi_i(0)} & \overline{E}_{i,2+\pi_i(0)} \\ \overline{E}_{i,2+\pi_i(1)} & \overline{E}_{i,1+3+\pi_i(1)} \end{pmatrix} \begin{pmatrix} B_{\pi_i(0)} \\ B_{\pi_i(1)} \end{pmatrix} E_iF_{i-1}^{-1} =$$

$$= \begin{pmatrix} F_i\overline{E}_{i,1+3\pi_i(0)} & F_i\overline{E}_{i,2+\pi_i(0)} \\ F_i\overline{E}_{i,2+\pi_i(1)} & F_i\overline{E}_{i,1+3\pi_i(1)} \end{pmatrix} \begin{pmatrix} B_{\pi_i(0)}E_iF_{i-1}^{-1} \\ B_{\pi_i(1)}E_iF_{i-1}^{-1} \end{pmatrix} =$$

$$= \begin{pmatrix} F_i\overline{E}_{i,1+3\pi_i(0)}B_{\pi_i(0)}E_iF_{i-1}^{-1} + F_i\overline{E}_{i,2+\pi_i(0)}B_{\pi_i(1)}E_iF_{i-1}^{-1} \\ F_i\overline{E}_{i,2+\pi_i(1)}B_{\pi_i(0)}E_iF_{i-1}^{-1} + F_i\overline{E}_{i,1+3\pi_i(1)}B_{\pi_i(1)}E_iF_{i-1}^{-1} \end{pmatrix}.$$

If we want to consider what $A_{i,0}$ maps $F_{i-1}\boldsymbol{x}$ to, we can analyze what $F_i\overline{E}_{i,1+3\pi_i(0)}B_{\pi_i(0)}E_iF_{i-1}^{-1} + F_i\overline{E}_{i,2+\pi_i(0)}B_{\pi_i(1)}E_iF_{i-1}^{-1}$ maps $F_{i-1}\boldsymbol{x}$ to. We see that $A_{i,0}$ is a sum of two terms, and we show how the first term acts on $F_{i-1}\boldsymbol{x}$. The other case is analogous, as is the case for $A_{i,1}$. Concerning the first term, we see that $F_i\overline{E}_{i,1+3\pi_i(0)}B_{\pi_i(0)}E_iF_{i-1}^{-1} \cdot F_{i-1}\boldsymbol{x} = F_i\overline{E}_{i,1+3\pi_i(0)}B_{\pi_i(0)}E_i\boldsymbol{x}$. By Lemma 1we know that since $\boldsymbol{x} \in \tilde{C}_{i-1}$, then also $E_i \cdot \boldsymbol{x} \in \tilde{C}_{i-1}$. By the same Lemma we know that since $E_i\boldsymbol{x} \in \tilde{C}_{i-1}$, then $B_{\pi_i(0)}E_i\boldsymbol{x} \in \tilde{C}_i$. By the same Lemma we know that since $B_{\pi_i(0)}E_i\boldsymbol{x} \in \tilde{C}_i$, also $\overline{E}_{i,1+3\pi_i(0)}B_{\pi_i(0)}E_i\boldsymbol{x} \in \tilde{C}_i$. Thus naturally $F_i\overline{E}_{i,1+3\pi_i(0)}B_{\pi_i(0)}E_i\boldsymbol{x} \in F_i\tilde{C}_i$. This can be shown analogously for the other summands and for $A_{i,1}$. Hence the claim holds. □

Lemma 3. *For all* $i \in [1,n]$, $A_{i,\overline{\alpha_i}}F_{i-1}(\boldsymbol{d_{3(i-1)}} + \tilde{C}_{i-1}) \subseteq F_i\tilde{C}_i$.

Proof. If $\alpha_i = 1$, then $\pi_i = \pi_{id}$ and thus $A_{i,0} = F_i\overline{E}_{i,1}B_0E_iF_{i-1}^{-1} + F_i\overline{E}_{i,2}B_1E_iF_{i-1}^{-1}$ and $A_{i,1} = F_i\overline{E}_{i,3}B_0E_iF_{i-1}^{-1} + F_i\overline{E}_{i,4}B_1E_iF_{i-1}^{-1}$.

If $\alpha_i = 0$, then we have that $\pi_i = \pi_{sw}$, and thus $A_{i,0} = F_i\overline{E}_{i,4}B_1E_iF_{i-1}^{-1} + F_i\overline{E}_{i,3}B_0E_iF_{i-1}^{-1}$ and $A_{i,1} = F_i\overline{E}_{i,2}B_1E_iF_{i-1}^{-1} + F_i\overline{E}_{i,1}B_0E_iF_{i-1}^{-1}$. We can sum that up as $A_{i,\alpha_i} = F_i\overline{E}_{i,3}B_0E_iF_{i-1}^{-1} + F_i\overline{E}_{i,4}B_1E_iF_{i-1}^{-1}$ and $A_{i,\overline{\alpha_i}} = F_i\overline{E}_{i,1}B_0E_iF_{i-1}^{-1} + F_i\overline{E}_{i,2}B_1E_iF_{i-1}^{-1}$. Now we can state

$$\begin{aligned} &A_{i,\overline{\alpha_i}}F_{i-1}(\boldsymbol{d_{3(i-1)}} + \tilde{C}_{i-1}) = \\ &(F_i\overline{E}_{i,1}B_0E_iF_{i-1}^{-1} + F_i\overline{E}_{i,2}B_1E_iF_{i-1}^{-1})F_{i-1}(\boldsymbol{d_{3(i-1)}} + \tilde{C}_{i-1}) = \\ &F_i\overline{E}_{i,1}B_0E_i(\boldsymbol{d_{3(i-1)}} + \tilde{C}_{i-1}) + F_i\overline{E}_{i,2}B_1E_i(\boldsymbol{d_{3(i-1)}} + \tilde{C}_{i-1}). \end{aligned}$$

As $E_i\boldsymbol{d_{3(i-1)}} = \boldsymbol{d_{3(i-1)+1}} + \overline{\boldsymbol{c}}_{\boldsymbol{3(i-1)+1}}$, we can say that $E_i(\boldsymbol{d_{3(i-1)}} + \tilde{C}_{i-1}) \in \boldsymbol{d_{3(i-1)+1}} + \tilde{C}_{i-1})$. Similarly, because $B_0\boldsymbol{d_{3(i-1)+1}} \in \tilde{C}_i$, and since B_0 maps $\tilde{C}_i$ into $\tilde{C}_i$, we have $B_0E_i(\boldsymbol{d_{3(i-1)}} + \tilde{C}_{i-1}) \in \tilde{C}_i$. Now $\overline{E}_{i,1}$ maps $\tilde{C}_i$ into $\tilde{C}_i$

and, therefore, $\overline{E}_{i,1}B_1E_i(\boldsymbol{d_{3(i-1)}}+\tilde{C}_{i-1}) \in \tilde{C}_i$. Naturally, $F_i\overline{E}_{i,1}B_0E_i(\boldsymbol{d_{3(i-1)}}+\tilde{C}_{i-1}) \in F_i\tilde{C}_i$.

For $F_i\overline{E}_{i,2}B_1E_i(\boldsymbol{d_{3(i-1)}}+\tilde{C}_{i-1})$, we analogously get that $E_i(\boldsymbol{d_{3(i-1)}}+\tilde{C}_{i-1}) \in \boldsymbol{d_{3(i-1)+1}}+\tilde{C}_{i-1}$, because $E_i(\boldsymbol{d_{3(i-1)}} \in \boldsymbol{d_{3(i-1)+1}}+\tilde{C}_{i-1}$. Analogously we get that $B_1E_i(\boldsymbol{d_{3(i-1)}}+\tilde{C}_{i-1}) \in \boldsymbol{d_{3(i-1)+2}}+\tilde{C}_i)$. Then, because $\boldsymbol{d_{3(i-1)+2}} \in \ker\overline{E}_{1,2}+\tilde{C}_i$, we get that $\overline{E}_{i,2}B_1E_i(\boldsymbol{d_{3(i-1)}}+\tilde{C}_{i-1}) \in \tilde{C}_i$. Finally, it naturally follows that $F_i\overline{E}_{i,2}B_1E_i(\boldsymbol{d_{3(i-1)}}+\tilde{C}_{i-1}) \in F_i\tilde{C}_i$. Hence the claim holds. □

Lemma 4. *For all $i \in [1,n]$, $A_{i,\alpha_i}F_{i-1}(\boldsymbol{d_{3(i-1)}}+\tilde{C}_{i-1}) \subseteq F_i(\boldsymbol{d_{3i}}+\tilde{C}_i)$.*

Proof. From the proof of the Lemma 3, we have that $A_{i,\alpha_i} = F_i\overline{E}_{i,3}B_0E_iF_{i-1}^{-1} + F_i\overline{E}_{i,4}B_1E_iF_{i-1}^{-1}$ Thus, $A_{i,\alpha_i}F_{i-1}(\boldsymbol{d_{3(i-1)}}+\tilde{C}_{i-1}) = F_i\overline{E}_{i,3}B_0E_i(\boldsymbol{d_{3(i-1)}}+\tilde{C}_{i-1})+ F_i\overline{E}_{i,4}B_1E_i(\boldsymbol{d_{3(i-1)}}+\tilde{C}_{i-1})$.

Consider $F_i\overline{E}_{i,3}B_0E_i(\boldsymbol{d_{3(i-1)}}+\tilde{C}_{i-1})$. As $E_i(\boldsymbol{d_{3(i-1)}}+\tilde{C}_{i-1}) = \boldsymbol{d_{3(i-1)+1}}+\overline{\boldsymbol{c}}_{\boldsymbol{3(i-1)+1}}$, then $E_i(\boldsymbol{d_{3(i-1)}}+\tilde{C}_{i-1}) \in \overline{\boldsymbol{c}}_{\boldsymbol{3(i-1)+1}}+\tilde{C}_{i-1}$. Since $\overline{\boldsymbol{c}}_{\boldsymbol{3(i-1)+1}} \in \tilde{C}_{i-1}$, then $B_0E_i(\boldsymbol{d_{3(i-1)}}+\tilde{C}_{i-1}) \in \tilde{C}_i$. Analogously to the proof of Lemma 3, we conclude that $F_i\overline{E}_{i,3}B_0E_i(\boldsymbol{d_{3(i-1)}}+\tilde{C}_{i-1}) \in F_i\tilde{C}_i$.

On the other hand, concerning $F_i\overline{E}_{i,4}B_1E_i(\boldsymbol{d_{3(i-1)}}+\tilde{C}_{i-1})$, we have that $E_i(\boldsymbol{d_{3(i-1)}}+\tilde{C}_{i-1}) \in \boldsymbol{d_{3(i-1)+1}}+\tilde{C}_{i-1}$, because $E_i\boldsymbol{d_{3(i-1)}} = \boldsymbol{d_{3(i-1)+1}}+\overline{\boldsymbol{c}}_{\boldsymbol{3(i+1)+1}}$. Next, it follows $B_1E_i(\boldsymbol{d_{3(i-1)}}+\tilde{C}_{i-1}) \in (\boldsymbol{d_{3(i-1)+2}}+\tilde{C}_i)$, because $B_1\boldsymbol{d_{3(i-1)+1}} = \boldsymbol{d_{3(i-1)+2}}+\overline{\boldsymbol{c}}_{\boldsymbol{3(i+1)+2,1}}$. Next, it follows that $\overline{E}_{i,4}B_1E_i(\boldsymbol{d_{3(i-1)}}+\tilde{C}_{i-1}) \in (\boldsymbol{d_{3i}}+\tilde{C}_i)$. Finally we naturally have that $F_i\overline{E}_{i,4}B_1E_i(\boldsymbol{d_{3(i-1)}}+\tilde{C}_{i-1}) \in F_i(\boldsymbol{d_{3(i-1)}}+\tilde{C}_i)$. Hence the result follows. □

Lemma 5. Eval *in Algorithm 4 can be computed locally, hence, Property 1 holds.*

Proof. The main body of Eval is locally multiplying linearly secret-shared vectors by public matrices. Computing the output b requires computing a scalar product, where $\boldsymbol{e} \in \mathbb{G}$. First, computing $[\![\overline{\boldsymbol{z}}_n]\!] = [\![\overline{\boldsymbol{z}}_n]\!]$ is clearly local. Finally, taking group elements to secret-shared powers is a local operation. Concretely, computing $\langle[\boldsymbol{e}], [\![\overline{\boldsymbol{z}}_n]\!]\rangle$ means each party j locally computes $\langle\!\langle b\rangle\!\rangle_j = \prod_i [d_i][\![\overline{\boldsymbol{z}}_{n,i}]\!]_j$ for all i. Hence, $b = \prod_j \langle\!\langle b\rangle\!\rangle_j = \prod_j\prod_i [d_i][\![\overline{\boldsymbol{z}}_{n,i}]\!]_j = \prod_i [d_i]\overline{\boldsymbol{z}}_{n,i} = [\sum d_i\overline{\boldsymbol{z}}_{n,i}]$. □

Lemma 6. *For any $i \in [1,n]$, $A_{i,\alpha_i}A_{i-1,\alpha_{i-1}}\dots A_{1,\alpha_1}\boldsymbol{z}_0 \in \boldsymbol{d_{3i}}+\tilde{C}_i$.*

Proof. For $i = 0$, this holds because $\boldsymbol{d_0} \in \boldsymbol{d_0}+\tilde{C}_0$. Using induction, assume that the statement holds for $i-1$. Thus, $A_{i-1,\alpha_{i-1}}\dots A_{1,\alpha_1}\boldsymbol{z}_0 \in \boldsymbol{d_{3(i-1)}}+\tilde{C}_{i-1}$. When we multiply this by A_{i,α_i} from the left, then, by Lemma 4, we get that the result must belong to $\boldsymbol{d_{3i}}+\tilde{C}_i$. Thus, by induction, the principle holds. □

Lemma 7. *In our DPF construction, Property 3 holds.*

Proof. The condition $Pred_i(\mathcal{F}_{i,\overline{\alpha_i}}(\sigma_r)) = 1$ translated into the specifics of our algorithm means that $\overline{\boldsymbol{z}}_i$ is in $\tilde{C}_i$. By Lemma 6, if r of length $i-1$ is a prefix of α, then we have that $\overline{\boldsymbol{z}}_{i-1} = A_{i,\alpha_i}A_{i-1,\alpha_{i-1}}\dots A_{1,\alpha_1}\boldsymbol{z}_0 \in F_{i-1}\boldsymbol{d_{3(i-1)}}+\tilde{C}_i$. By Lemma 3, $A_{i,\overline{\alpha_i}}F_{i-1}(\boldsymbol{d_{3(i-1)}}+\tilde{C}_{i-1}) \subseteq F_i\tilde{C}_i$. Thus, the claim holds. □

Game 1: FSS, input $(\alpha^{[1]}, \beta^{[1]})$

Input : $\mathcal{A}$, $\mathsf{prm} = (N, n, k, \tau, u, \mathbb{G})$

1 $(\alpha^{[1]}, \beta^{[1]}), (\alpha^{[2]}, \beta^{[2]}) \leftarrow \mathcal{A}(\mathsf{prm})$;

2 $\{1, \ldots, m\} \supsetneq T \leftarrow \mathcal{A}(\mathsf{prm})$;

3 $\{A_{i,0}\}_{i=1}^{n}, \{A_{i,1}\}_{i=1}^{n}, [\boldsymbol{e}], [\![\boldsymbol{z}_0{}^{(1)}]\!], [\![\boldsymbol{z}_0{}^{(2)}]\!] \leftarrow \mathsf{Gen}(\mathsf{prm}, \alpha^{[1]}, \beta^{[1]})$;

4 **return** $\mathcal{A}(\{A_{i,0}\}_{i=1}^{n}, \{A_{i,1}\}_{i=1}^{n}, [\boldsymbol{e}],$

5 $\{[\![\boldsymbol{z}_0{}^{(1)}]\!]_j\}_{j\in T}, \{[\![\boldsymbol{z}_0{}^{(2)}]\!]_j\}_{j\in T}, \mathsf{prm})$;

Game 2: FSS, input $(\alpha^{[2]}, \beta^{[2]})$

Input : $\mathcal{A}$, $\mathsf{prm} = (N, n, k, \tau, u, \mathbb{G})$

1 $(\alpha^{[1]}, \beta^{[1]}), (\alpha^{[2]}, \beta^{[2]}) \leftarrow \mathcal{A}(\mathsf{prm})$;

2 $\{1, \ldots, m\} \supsetneq T \leftarrow \mathcal{A}(\mathsf{prm})$;

3 $\{A_{i,0}\}_{i=1}^{n}, \{A_{i,1}\}_{i=1}^{n}, [\boldsymbol{e}], [\![\boldsymbol{z}_0{}^{(1)}]\!], [\![\boldsymbol{z}_0{}^{(2)}]\!] \leftarrow \mathsf{Gen}(\mathsf{prm}, \alpha^{[2]}, \beta^{[2]})$;

4 **return** $\mathcal{A}(\{A_{i,0}\}_{i=1}^{n}, \{A_{i,1}\}_{i=1}^{n}, [\boldsymbol{e}],$

5 $\{[\![\boldsymbol{z}_0{}^{(1)}]\!]_j\}_{j\in T}, \{[\![\boldsymbol{z}_0{}^{(2)}]\!]_j\}_{j\in T}, \mathsf{prm})$;

Lemma 8. *In our DPF construction, Properties 4 and 5 holds.*

Proof. Property 4 follows from Lemma 2. Property 5 states that if $\overline{\boldsymbol{z}}_n$ is in $F_n\tilde{C}_n$, then $\langle [\boldsymbol{e}], [\![\overline{\boldsymbol{z}}_n]\!] \rangle = [0]$ which holds by the construction of $\boldsymbol{e}$ in line 11. □

Lemma 9. *In our DPF construction, Property 2 holds.*

Proof. By Lemma 6 for prefix of length n, we have that for $y = \alpha$, $\overline{\boldsymbol{z}}_n = \boldsymbol{d_{3n}} + \tilde{C}'_n$. On line 11, $\boldsymbol{e}$ is such that $\left\langle \boldsymbol{e}, \boldsymbol{d_{3n}} + \tilde{C}'_n \right\rangle = \langle \boldsymbol{e}, \boldsymbol{d_{3n}} \rangle = \gamma$. On line 13 in Algorithm 3, we multiply $\boldsymbol{z}_0$ by $\gamma^{-1}\beta$. Since $\overline{\boldsymbol{z}}_n$ is obtained from $\overline{\boldsymbol{z}}_0$ using linear operations, the new value of $\langle \boldsymbol{e}, \overline{\boldsymbol{z}}_n \rangle = \gamma^{-1}\beta\gamma = \beta$. Hence, the claim holds. □

Theorem 2. *Our DPF construction is correct.*

Proof. Follows directly from Theorem 1 and Lemmata 5, 7, 8, and 9. □

The correctness proof of the full algorithms with doubled variables given in Appendix C.3 is analogous.

5 Security proof

The security proof in this section is given for the full version of the Algorithms 9 and 10 given in Appendix C that double the variables in Algorithms 3 and 4.

Theorem 3. *Let N be a product of two large primes. Suppose that $(\mathbb{Z}_n, k, \tau, n)$-Multi-Switching-Decisional Linear Double Code Equivalence With Common Submodules Assumption (Assumption 6) holds and that $(\mathbb{G}, \mathcal{F}, 2n + 2\tau)$-Submodule Discrete-Log Vector Assumption (Assumption 1) holds for the function family $\mathcal{F}$ being the $\{A_{i,0}\}_{i=1}^{n}$, $\{A_{i,1}\}_{i=1}^{n}$ as described in Algorithm 9 where the submodule $\tilde{B}$ is $\tilde{C}_n^{(1)} + \tilde{C}_n^{(2)}$. Then the construction given by Algorithms 9 and 10 is a secure DPF, provided that factoring N is hard.*

Sketch. In the first Game 1, the input is $(\alpha^{(1)}, \beta^{(1)})$ and the key is produced the normal way. Consider an adversary who corrupts a strict subset $T \subset \{1, \ldots, m\}$ of the parties, thus learning $\{A_{i,0}\}_{i=1}^{n}, \{A_{i,1}\}_{i=1}^{n}, [\boldsymbol{e}]$ and $[\![\boldsymbol{z}_0]\!]_j$ for $j \in T$. We

Game 3: FSS, $\boldsymbol{z}_0, \boldsymbol{z}_1$ random

Input : $\mathcal{A}$, $\mathsf{prm} = (N, n, k, \tau, u, \mathbb{G})$

$(\alpha^{[1]}, \beta^{[1]}), (\alpha^{[2]}, \beta^{[2]}) \leftarrow \mathcal{A}(\mathsf{prm})$;

$\{1, \ldots, m\} \supsetneq T \leftarrow \mathcal{A}(\mathsf{prm})$;

$\{A_{i,0}\}_{i=1}^n, \{A_{i,1}\}_{i=1}^n, [\boldsymbol{e}], [\![\boldsymbol{z}_0{}^{(1)}]\!], [\![\boldsymbol{z}_0{}^{(2)}]\!] \leftarrow \mathsf{Gen}(\mathsf{prm}, \alpha^{[1]}, \beta^{[1]})$;

for $j \in T$ **do**

  $[\![\boldsymbol{z}_0{}^{(1)}]\!]_j, [\![\boldsymbol{z}_0{}^{(2)}]\!]_j \xleftarrow{\$} (\mathbb{Z}_N)^k$;

return $\mathcal{A}(\{A_{i,0}\}_{i=1}^n, \{A_{i,1}\}_{i=1}^n, [\boldsymbol{e}],$

$\{[\![\boldsymbol{z}_0{}^{(1)}]\!]_j\}_{j\in T}, \{[\![\boldsymbol{z}_0{}^{(2)}]\!]_j\}_{j\in T}, \mathsf{prm})$;

Game 4: FSS, $\boldsymbol{e}$ replaced by random

Input : $\mathcal{A}$, $\mathsf{prm} = (N, n, k, \tau, u, \mathbb{G})$

$(\alpha^{[1]}, \beta^{[1]}), (\alpha^{[2]}, \beta^{[2]}) \leftarrow \mathcal{A}(\mathsf{prm})$;

$\{1, \ldots, m\} \supsetneq T \leftarrow \mathcal{A}(\mathsf{prm})$;

$\{A_{i,0}\}_{i=1}^n, \{A_{i,1}\}_{i=1}^n, [\boldsymbol{e}], [\![\boldsymbol{z}_0{}^{(1)}]\!], [\![\boldsymbol{z}_0{}^{(2)}]\!] \leftarrow \mathsf{Gen}(\mathsf{prm}, \alpha^{[1]}, \beta^{[1]})$;

for $j \in T$ **do**

  $[\![\boldsymbol{z}_0{}^{(1)}]\!]_j, [\![\boldsymbol{z}_0{}^{(2)}]\!]_j \xleftarrow{\$} (\mathbb{Z}_N)^k$;

return

$\mathcal{A}(\{A_{i,0}\}_{i=1}^n, \{A_{i,1}\}_{i=1}^n, [\boldsymbol{r}],$

$\{[\![\boldsymbol{z}_0{}^{(1)}]\!]_j\}_{j\in T}, \{[\![\boldsymbol{z}_0{}^{(2)}]\!]_j\}_{j\in T}, \mathsf{prm})$;

highlight the changes between the games with red font. As the first step, in Game 3, we replace the $[\![\boldsymbol{z}_0]\!]_j$ by randomly sampled element. We can do this because these values are secret-shared and thus information-theoretically hidden.

We now note that in Game 3, the vectors $\boldsymbol{c_i'}^{(b)}$ were chosen randomly, for $i \in [1, 2n{+}2\tau]$. These are basis vectors for $\tilde{C}_n^{(1)} {+} \tilde{C}_n^{(2)}$. We now use our Submodule Discrete-Log Vector Assumption, which states that either we can replace $[\boldsymbol{e}]$ by a random vector or an efficient extractor can produce an element of $\tilde{C}_n^{(1)} + \tilde{C}_n^{(2)}$. By Theorem 5 we have that in case such extractor $\mathtt{Extr}$ exists, then $\mathcal{A}$ together with $\mathtt{Extr}$ have factored N. Since we assumed that N is hard to factor, then we can replace $[\boldsymbol{e}]$ by a random vector $[\boldsymbol{r}]$, thus obtaining Game 4

We now note that since now nothing in the output depends on $\beta^{[1]}$, we can replace it with $\beta^{[2]}$, thus obtaining Game 5. Now we want to use the $(\mathbb{Z}_n, k, \tau, n)$-Multi-Switching-Decisional Linear Double Code Equivalence With Common Submodules Assumption. Let the challenger in that assumption sample B_0,B_1, $\overline{E}_{i,c}$ for $i \in [1, n]$ and $c \in [1, n]$, E_i and F_i and $H_i$for $i \in [1, n]$. we then give the challenger the $\alpha^{[1]}$ and $\alpha^{[2]}$ that we got from the adversary. We then get back $\{A_{i,0}\}_{i=1}^n$,$\{A_{i,1}\}_{i=1}^n$ and $\{H_i F_i^{-1}\}_{i=1}^n$.

We now set $A_{i,0} \leftarrow H_i F_i^{-1} A_{i,0}$ and $A_{i,1} \leftarrow H_i F_i^{-1} A_{i,1}$. We now note that we have the exact same distribution for as is obtained in Algorithm 9 for $\{A_{i,0}\}_{i=1}^n, \{A_{i,1}\}_{i=1}^n$. Thus, we can now replace $\alpha^{(1)}$ by $\alpha^{(2)}$ and if the adversary can distinguish between those, it can also work as a distinguisher in the game for Assumption , which means that there must be an extractor who can output an element from either $\tilde{C}_n^{(1)} + \tilde{C}_n^{(2)}$, $\boldsymbol{d_{3(i-1)}}^{(b)} + \tilde{C}_n^{(1)} + \tilde{C}_n^{(2)}$, or $\boldsymbol{d_{3(i-1)+1}}^{(b)} + \tilde{C}_n^{(1)} + \tilde{C}_n^{(2)}$. By Theorem 5 this would imply factoring of N which we assumed to be infeasible. Hence we can get to Game 6. Now we can, analogously to before, get to Game 2. Hence the claim is proven. □

Game 5: FSS, using $\beta^{[2]}$

Input : $\mathcal{A}$, $\mathsf{prm} = (N, n, k, \tau, u, \mathbb{G})$

$(\alpha^{[1]}, \beta^{[1]}), (\alpha^{[2]}, \beta^{[2]}) \leftarrow \mathcal{A}(\mathsf{prm})$;

$\{1, \ldots, m\} \supsetneq T \leftarrow \mathcal{A}(\mathsf{prm})$;

$\{A_{i,0}\}_{i=1}^{n}, \{A_{i,1}\}_{i=1}^{n}, [\boldsymbol{e}], [\![\boldsymbol{z}_0{}^{(1)}]\!], [\![\boldsymbol{z}_0{}^{(2)}]\!] \leftarrow \mathsf{Gen}(\mathsf{prm}, \alpha^{[1]}, \beta^{[2]})$;

for $j \in T$ **do**

  $[\![\boldsymbol{z}_0{}^{(1)}]\!]_j, [\![\boldsymbol{z}_0{}^{(2)}]\!]_j \xleftarrow{\$} (\mathbb{Z}_N)^k$;

return

$\mathcal{A}(\{A_{i,0}\}_{i=1}^{n}, \{A_{i,1}\}_{i=1}^{n}, [\boldsymbol{r}], \{[\![\boldsymbol{z}_0{}^{(1)}]\!]_j\}_{j\in T}, \{[\![\boldsymbol{z}_0{}^{(2)}]\!]_j\}_{j\in T}, \mathsf{prm})$;

Game 6: FSS, using $\alpha^{[2]}$

Input : $\mathcal{A}$, $\mathsf{prm} = (N, n, k, \tau, u, \mathbb{G})$

$(\alpha^{[1]}, \beta^{[1]}), (\alpha^{[2]}, \beta^{[2]}) \leftarrow \mathcal{A}(\mathsf{prm})$;

$\{1, \ldots, m\} \supsetneq T \leftarrow \mathcal{A}(\mathsf{prm})$;

$\{A_{i,0}\}_{i=1}^{n}, \{A_{i,1}\}_{i=1}^{n}, [\boldsymbol{e}], [\![\boldsymbol{z}_0{}^{(1)}]\!], [\![\boldsymbol{z}_0{}^{(2)}]\!] \leftarrow \mathsf{Gen}(\mathsf{prm}, \alpha^{[2]}, \beta^{[2]})$;

for $j \in T$ **do**

  $[\![\boldsymbol{z}_0{}^{(1)}]\!]_j, [\![\boldsymbol{z}_0{}^{(2)}]\!]_j \xleftarrow{\$} (\mathbb{Z}_N)^k$;

return

$\mathcal{A}(\{A_{i,0}\}_{i=1}^{n}, \{A_{i,1}\}_{i=1}^{n}, [\boldsymbol{r}], \{[\![\boldsymbol{z}_0{}^{(1)}]\!]_j\}_{j\in T}, \{[\![\boldsymbol{z}_0{}^{(2)}]\!]_j\}_{j\in T}, \mathsf{prm})$;

6 Discussion of assumptions

6.1 Code equivalence assumptions

In our actual proof we use only one of the code-based assumptions introduced in Section 2.3.[3] There has been a long line of assumptions stating that for a given code with a generator matrix C, multiplying it from the right by a random invertible matrix, and left by some other matrix, such as permutation matrix, monomial matrix, or a noninvertible matrix, hides the structure of the code and makes it indistinguishable from random [40]. We know also that it is widely believed that although this problem is easy for some types of codes if we right-multiply by a permutation matrix instead of an invertible matrix, for invertible matrices almost all codes are believed to be secure against this type of attack, provided that the underlying field has size greater than 5 [34].

We can restate the linear code equivalence as trying to distinguish $Q \begin{pmatrix} B_0 \\ B_1 \end{pmatrix} F$ from random, where Q is a random monomial matrix, F is a random invertible matrix and B_0, B_1 given by the adversary form a generator matrix for a code.

Let us compare this to the matrix $\begin{pmatrix} \overline{E}_{1+3\pi_i(0)} & \overline{E}_{2+\pi_i(0)} \\ \overline{E}_{2+\pi_i(1)} & \overline{E}_{1+3+\pi_i(1)} \end{pmatrix} \begin{pmatrix} B_{\pi_i(0)} \\ B_{\pi_i(1)} \end{pmatrix} E_i F_i^{-1}$ in Assumption 3 (Switching-Decisional Linear Code Equivalence With Common Submodules). We note that in both cases we left-multiply by a random invertible matrix. By choosing the matrix $\begin{pmatrix} \overline{E}_{1+3\pi_i(0)} & \overline{E}_{2+\pi_i(0)} \\ \overline{E}_{2+\pi_i(1)} & \overline{E}_{1+3+\pi_i(1)} \end{pmatrix}$ we have far larger entropy than choosing a monomial matrix. We also do not require that this product looks random, it should merely hide the order of B_0 and B_1. The matrix $\begin{pmatrix} \overline{E}_{1+3\pi_i(0)} & \overline{E}_{2+\pi_i(0)} \\ \overline{E}_{2+\pi_i(1)} & \overline{E}_{1+3+\pi_i(1)} \end{pmatrix}$ is close to being random, with the added caveat that it

[3] Note that we switch here the usual directions of left and right that are usually used in these kinds of discussions. This is done so because usually generator matrices are multiplied from the left but we multiply our matrices from the right, thus to get equivalent assumptions, we needed to make these cosmetic changes.

must satisfy some constraints given by the code generator matrix $\begin{pmatrix} B_{\pi_i(0)} \\ B_{\pi_i(1)} \end{pmatrix}$. For this reason, we added the caveat to our assumption that, in case there is a distinguishing adversary, it must be able to produce an example of an element for which this constraint applies. Hence, we believe that Assumption 3 is justified.

Concerning Assumption 4 (Multi-Switching-Decisional Linear Code Equivalence With Common Submodules), we notice that we essentially sampled n instances of Assumption 3 with the same B_0 and B_1 and the $\tilde{C}_i$ contained in each other, and that we are also given the additional $H_i F_i^{-1}$. Let us first consider the issue that we have multiple instances of Assumption 3 problems. All of them are hidden with independent instances of the $\overline{E}_{i,c}, E_i$ and F_i. We also do not give $\begin{pmatrix} B_0 \\ B_1 \end{pmatrix}$ to the adversary this time. We thus do not believe that we can consider this version of the game just as a repeated version of the Assumption 3 game.

We discuss in Appendix E why we think that given two matrices that both map the same $\tau + i$-dimensional submodule into the same $\tau + i + 1$-dimensional submodule, it is hard to compute such submodules even if we do not work over $\mathbb{Z}_{pq}$ where the factorization of pq is not known.

Now we need to discuss what happens if we additionally give the adversary $H_i F_i^{-1}$. If the H_i were chosen without any constraints, then $H_i F_i^{-1}$ would information-theoretically hide F_i^{-1}. In our case, however, the H_i is subject to one or two constraints. Similarly to before, we assume that if the adversary manages to distinguish the sampling of H_i from random, then they must have figured out the vectors that are subject to the constraints, and thus, the extractor can output them. If the extractor cannot output them, then the adversary does not know what they are, and cannot distinguish between our case, where H_i was chosen subject to those constraints, and the general case, with no constraints.

Since in Assumption 4, we reuse some of the hiding matrices, one might worry whether that might leak some information. Indeed, we note that there have been proposed assumptions of code equivalences where the hiding monomial matrix in linear equivalence problems is reused and for which Budroni et al showed in [9] that the respective ILCE and 2-LCE assumptions do not hold, by constructing an attack against them, also giving an attack that works against matrix codes. However, their attack requires multiple pairs of codes where in each pair one code is obtained from the other by (right)-multiplying by the same matrix. In our assumption, there are some repeating elements. However, we argue that we do not have equivalent codes in our construction, since we have a large number of confounding matrices with high entropy.

Let us now discuss the possible repeating matrices, starting with the central matrices B_0 and B_1. These are in the centers of the matrix products other unknown matrices, not left/right multiplied, and secondly, the matrices $F_i \overline{E}_{i,c}$ and $E_i F_{i-1}$ do not form equivalent codes, for example, they have constraints of different ranks and are square matrices. Secondly, one could argue that the matrices F_i and F_{i-1}^{-1} do repeat in the constructions of $A_{i,0}$ and $A_{i,1}$ and also that F_i is reused in the construction of $A_{i+1,0}$ and $A_{i+1,1}$ in its inverse form.

However, these are both multiplied by the $\overline{E}_{i,c}$, E_i and H_i which provide a great deal of entropy. Again, one does not obtain equivalent codes.

Thirdly, one could argue that we can describe the submodule $\tilde{C}_i$ by some generator matrix T_i such that $\tilde{C}_i = \{T_i \boldsymbol{x} | \boldsymbol{x} \in \mathbb{Z}_N^{\tau+i}\}$. Could we then not argue that $A_{i,0}F_{i-1}T_i$ and $A_{i,1}F_{i-1}T_i$ are equivalent code and that the T_i repeats over different levels? First, note that F_{i-1} changes the T_i totally for each level. Secondly, note that even disregarding F_{i-1}, with an overwhelming probability, $A_{i,0}F_{i-1}T_i$ and $A_{i,1}F_{i-1}T_i$ are different codes. The central components B_0 and B_1 map $\tilde{C}_{i-1}$ into $\tilde{C}_i$, but of course they cannot map a submodule of dimension $\tau + i - 1$ *onto* a submodule of dimension $\tau + i$. Hence they both map the $\tilde{C}_{i-1}$ onto different submodules of $\tilde{C}_i$ of rank $\tau + i - 1$ and we do not get equivalent codes.

Fourthly, the attack given in [9] allows one to obtain the secret matrix, however, we show in Appendix C that one must break factoring in order to learn elements of secret submodules $\tilde{C}_i$ or the secret kernel vectors $\boldsymbol{d_i}$. Thus, it seems unlikely that this method of attack is successful.

Could the attack of Budroni et al be generalized to common *subcodes* that are hidden with repeated hiding matrices? This is unlikely. In the technique of Budroni et al we have that the repeated hiding matrix Q is stated as a vector and then, from tensoring together various publicly known matrices, one obtains a large matrix for which Q is orthogonal to every row, and thus Q is in the kernel of that matrix, which allows us to recover it. For subcodes, we would need to multiply Q by some projection matrix first. Thus we would not have a large matrix where every row is orthogonal with some one vector but the vector would vary for all rows, thus defeating the premise of the attack.

Our final assumptions double the number of variables. However, the arguments made for singular variables can also be generalized for the doubled case.

6.2 Submodule Discrete-Log Vector Assumption

Let $\boldsymbol{e} = (\boldsymbol{e}_1, \ldots \boldsymbol{e}_k)$. We note that if the adversary can produce any element $\boldsymbol{b}$ of $\tilde{B}$, then they can test whether $\langle [\boldsymbol{e}], \boldsymbol{b} \rangle = [0]$ and the extractor is successful. The meaningful things that the adversary can do with elements in the group exponent is to compute scalar products in the form of $\langle [\boldsymbol{c}], \boldsymbol{a} \rangle$ with arbitrary vectors $\boldsymbol{a}$ and test for equality of $[a] \stackrel{?}{=} [b]$. We can also consider the multiplication of single elements like $[a] \in \mathbb{G}$ and $b \in \mathbb{Z}_N$ as special instances of scalar products where the other elements in the vector are zeroes.

We first note that since $\boldsymbol{e}$ is chosen at random from a module of size $\mathbb{Z}_N^{k-\rho}$, then guessing any particular element $\boldsymbol{e}_i$ is hard even if one knows all of $\tilde{B}$. In addition, guessing any linear relation between $< k - \rho$ coefficients is similarly hard, as even for a fixed $\tilde{B}$, the number of choices for such relations is exponential. Thus any $k - \rho - 1$ elements of $\boldsymbol{e}$, are distributed uniformly at random.

Could we obtain another vector $\boldsymbol{c} \notin \tilde{B}$ such that $\langle [\boldsymbol{e}], \boldsymbol{a} \rangle = [0]$? This is very unlikely, even if one knows $\tilde{B}$. If we can do that, then we can reduce the dimension of freedom for the possibilities of $\boldsymbol{e}$ by one. Essentially, such a vector $\boldsymbol{c}$ does not

depend on $\tilde{B}$ and depends only on $\boldsymbol{e}$. Thus producing such a vector means that we learn some nontrivial information about $\boldsymbol{e}$ which the DL assumption forbids.

Suppose now that the adversary has some partial information about $\tilde{B}$ but not enough to produce any element $\boldsymbol{b}$ of $\tilde{B}$. For example, consider the case when the adversary learns the first $k-1$ coordinates of ρ basis vectors of B. Could the adversary do anything useful with those? If they computed the scalar product with the first $k-1$ coordinates of $[\boldsymbol{e}]$, then the result would be randomly distributed. If they managed to guess the last coefficient of any basis vector, then they could check whether it is correct, but this would require N guesses with N being exponential. More specifically, if we had that $\boldsymbol{b} = (b_1, \ldots, b_{k-1}, x)$ and $[\boldsymbol{e}] = [\boldsymbol{e}_1], \ldots, [\boldsymbol{e}_k]$, then, in order to find an x such that $\langle [\boldsymbol{e}], \boldsymbol{b} \rangle = [0]$, one would have that $x[\boldsymbol{e}_k] = [\sum_{j=1}^{-k-1} b_j \boldsymbol{e}_j]$. Thus, finding x would be equivalent to finding the discrete logarithm of $[\sum_{j=1}^{-k-1} b_j \boldsymbol{e}_j]$ in the basis of $[\boldsymbol{e}_k]$ which is hard.

We can also transform $[\boldsymbol{e}]$ by some matrix multiplications. Note, however, that in the exponent, we still can only test whether some scalar products are zero. Thus, if we for some matrix M can compute $[\boldsymbol{e}M]$, then this is only useful if one can also obtain some vector $\boldsymbol{x}$ such that $\langle [\boldsymbol{e}M], \boldsymbol{x} \rangle = [0]$. This, however is equivalent to $\langle [\boldsymbol{e}], M\boldsymbol{x} \rangle = [0]$. We noted before that it is infeasible to obtain a vector that is orthogonal to $\boldsymbol{e}$ but not in B, thus it must be that $M\boldsymbol{x} \in B$, and hence, the extractor can output an element of B.

Could knowing some element of $\boldsymbol{b'} \in \tilde{B}^{\perp}$ help the adversary? We note that with an overwhelming probability, $\langle [\boldsymbol{e}], \boldsymbol{b'} \rangle$ is nonzero and thus this test is not useful. We could compute $[\boldsymbol{b'}]$ and add it to $[\boldsymbol{e}]$ to obtain $[\boldsymbol{e} + \boldsymbol{b'}]$. However, this is just another random element of $\tilde{B}^{\perp}$, from the same distribution as $[\boldsymbol{e}]$, and thus it is not particularly helpful. What if we could obtain many elements of $\tilde{B}^{\perp}$? If we can obtain as many independent elements in $\mathbb{Z}_N$ as is the rank of $B^{\perp}$ we can simply compute $\tilde{B}$ and thus the extractor is successful. In general, if we obtain $\mathsf{rank}(B^{\perp}) - t$ such independent elements, and for some vector $\boldsymbol{b} \in B$ we know at least $\rho + t$ coordinates then we can find out $\boldsymbol{b}$ without using $\boldsymbol{e}$. (recall that for elements of $\tilde{B}$, we can choose the first ρ coordinates arbitrarily)

Now, suppose that we obtain $\mathsf{rank}(\tilde{B}^{\perp}) - t$ such independent elements and for some basis vector $\boldsymbol{b}$ of $\tilde{B}$ we know $\rho + t - 1$ coordinates. Could we do anything more with this? In essence, we obtain a system of linear equations for the missing coefficients of $\boldsymbol{b}$ where the coefficients in one row are in the exponent and the others are in the plain and the solving of which still requires computing discrete logarithms. Thus, even if the adversary learns the maximal amount about B without still learning it, distinguishing $\boldsymbol{e}$ or part of it from random still requires solving a discrete logarithm problem. As future work, we hope to obtain a more thorough analysis of this assumption and reduce it to the Generic Group Model.

7 Related work

Function secret sharing (FSS) was proposed in [6] and the notion of distributed point functions (DPF) appeared already in [23]. While FSS is often defined as a

multi-party primitive, DPF was originally defined for the two-party case. However, for both FSS and DPF schemes, the existing constructions are mostly for the two-party case. The overall idea is that the setup phase generates keys for each party. Afterwards, the parties can use this key to locally compute a secret share of the function output for any desired input. Since the initial proposals, efficient DPF schemes have been proposed [7,25]. In addition, DPF schemes have been extended to the multi-point function case [30,5]. These approaches follow a common pattern introduced in Sec. 3.1. Since the original proposals, the multi-party case for interesting FSS functionalities, including point functions, has been an open question with some approaches towards it but no well established practical schemes. Note that, as already established in [6], trivial schemes, such as secret sharing the truth table of the function, always exist for any number of parties. For point functions, a more efficient way to share and compress the truth table is also addressed in [6] resulting in a scheme for t parties with up to $t-1$ corrupted parties and key size $O(s \cdot 2^{t/2} \cdot |D|^{1/2})$ for PRG seed length s and input domain D. The main open question proposed there is reducing the key size with respect to D without introducing stronger assumptions than the existence of one-way functions. Further improvements to this approach are given in [29].

Concurrently with our work, [24] also considered the PRG-based scheme for multi-party DPF. They extend the variant of [6] where the input α is transformed to two indices used to address the correct field in a table. The parties get PRG seeds that they only differ in the row of α and correction words that ensure that the right column gets corrected to β. They extend this approach to multi-party case by interpreting it as a combinatorial design on bipartite graphs and then proposing a randomized construction for multi-party. They achieve a key size that is polynomial in the number of parties t and sublinear in the domain size $|D|$, concretely $O(|D|^{1/2+\varepsilon}t^3)$ for some arbitrary constant $\varepsilon > 0$. This construction relies on existence of one-way functions. This paper also discusses an impossibility result showing that a DPF with deterministic combinatorial design such as the variant proposed in [6] must have key sizes exponential in the number of parties. Our construction extends a different variant of two-party DPF and is not subject to this impossibility result.

Other concurrent works extending the grid-based idea of [6] are and DDH-based approach in [17] and the honest majority DPF in [18]. [18] achieves the key size $O(\sqrt{|D|\binom{t}{m+1}}\log q)$ for m corrupted parties and $\beta \in \mathbb{F}_q$. Replicated secret sharing for honest majority multi-party DPF was used in [12] to achieve key size $O(\sqrt{|D|})$ for information-theoretic DPFs. Similar key sizes are achieved by schemes with non-linear reconstruction algorithms [14,29].

Multi-party DPF from standard public-key assumptions are also developed to achieve homomorphic secret sharing (HSS) [15]. Initially, [15] worked for a fixed number of parties and uses nesting to combine DPF and HSS for smaller number of parties to achieve DPF and HSS for larger number of parties, getting an 8-party DPF. This work has been extended to any number of parties by [1] and [16]. Two-party succinct HSS is used by all pairs of parties in [1] to get multi-party DPF and a batch of DPFs. Concretely, [16] shows how to nest more

schemes by showing how to reduce the complexity of the nested schemes to get DPF. Both of these give constructions with key size polynomial in the domain size and number of parties. Each nesting operation in [16] increases the size of the domain of the initial 2-party DPF, for k nesting operations giving a scheme for 2^{k+1} parties with domain size δ^d, the initial domain for the two-party needs to be $\delta^{(c,d)\uparrow k}$ where $(c, d) \uparrow k$ is the tower of exponents, for example, they need a two-party DPF with domain size $\delta^{c^{c^d}}$ for 8-party DPF. The computational complexity of their scheme also has a power tower relation to the number of parties thus making it infeasible for large number of parties.

Threshold fully homomorphic encryption has been used to achieve FSS for threshold structures [3]. More specifically, they propose a universal thresholdizer that can be constructed from threshold fully homomorphic encryption and used to create threshold variants of various cryptographic schemes. The threshold fully homomorphic encryption is based on secret sharing schemes. The thresholdizer can be used to build a FSS scheme where the reconstruction algorithm for the outputs depends on the depth of the function. Additive function sharing spooky encryption was used to achieve FSS schemes for all functions in [21].

There exist information-theoretic DPF constructions for the multi-party case. The branch for information-theoretic FSS was started by [32] that considered point-functions where the reconstruction algorithm depends on the private information retrieval protocol. For distributed point functions with additive reconstruction, [8] started the research with constructions for 3 and 4 servers where at most one of them could be corrupted. These constructions were based on the information theoretic n-server private information retrieval. The main limitation of their work is that the key sizes are exponential in the size of the output group. This was extended to the eight server case, tolerating t corrupted parties and $d(t + 1)$-party case for any integer d in [31]. In addition, [31] reduces the key size, and allows for Abelian groups as the function output. A multi-point function extension of these approaches together with further reducing the size of the key is considered in [28]. For l-point function, the construction in [28] applies to $d(t + l - 1) + 1$ parties. An extended setting where a user has access to multiple of the servers holding the keys is considered in [26].

8 Conclusion

We generalized the tree-based two-party DPF of [6] and proposed a set of properties that have to hold for a scheme generalizing this idea to the multi-party case. We prove that if these properties are fulfilled, the resulting scheme is a distributed point function. Subsequently, we presented a concrete construction that satisfies these properties. This construction yields the first scheme for DPF for an arbitrary number of parties with a key size polylogarithmic in the domain size where the computation and key sizes are independent of the number of parties. More specifically, the key size is $O(\log^3 |D|)$ where $|D|$ is the size of the domain. We demonstrate that our scheme satisfies all the properties necessary to ensure it is a correct distributed point function.

We discussed the security of eigenvectors of matrices over $\mathbb{Z}_{pq}$ and introduced security assumptions that we believe are plausible. We then showed that our scheme is secure under these assumptions, factoring, and the Generic Group Model. It is further work to either study these assumptions in more detail or to lift the proposed construction to one using more common assumptions.

Acknowledgments

This work was supported by the Estonian Research Council grant (PSG1228).

References

1. Abram, D., Roy, L., Scholl, P.: Succinct homomorphic secret sharing. In: Joye, M., Leander, G. (eds.) Advances in Cryptology - EUROCRYPT 2024 - 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zurich, Switzerland, May 26-30, 2024, Proceedings, Part VI. Lecture Notes in Computer Science, vol. 14656, pp. 301–330. Springer (2024). https://doi.org/10.1007/978-3-031-58751-1_11, https://doi.org/10.1007/978-3-031-58751-1_11
2. Boneh, D., Boyle, E., Corrigan-Gibbs, H., Gilboa, N., Ishai, Y.: Lightweight techniques for private heavy hitters. In: 42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021. pp. 762–776. IEEE (2021). https://doi.org/10.1109/SP40001.2021.00048, https://doi.org/10.1109/SP40001.2021.00048
3. Boneh, D., Gennaro, R., Goldfeder, S., Jain, A., Kim, S., Rasmussen, P.M.R., Sahai, A.: Threshold cryptosystems from threshold fully homomorphic encryption. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part I. LNCS, vol. 10991, pp. 565–596. Springer, Cham (Aug 2018). https://doi.org/10.1007/978-3-319-96884-1_19
4. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y.: Compressing vector OLE. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. pp. 896–912 (2018)
5. Boyle, E., Gilboa, N., Hamilis, M., Ishai, Y., Tu, Y.: Improved Constructions for Distributed Multi-Point Functions . In: 2025 IEEE Symposium on Security and Privacy (SP). pp. 44–44. IEEE Computer Society, Los Alamitos, CA, USA (May 2025). https://doi.org/10.1109/SP61157.2025.00044, https://doi.ieeecomputersociety.org/10.1109/SP61157.2025.00044
6. Boyle, E., Gilboa, N., Ishai, Y.: Function secret sharing. In: Annual international conference on the theory and applications of cryptographic techniques. pp. 337–367. Springer (2015)
7. Boyle, E., Gilboa, N., Ishai, Y.: Function secret sharing: Improvements and extensions. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 1292–1303 (2016)
8. Boyle, E., Gilboa, N., Ishai, Y., Kolobov, V.I.: Information-theoretic distributed point functions. In: Dachman-Soled, D. (ed.) ITC 2022. LIPIcs, vol. 230, pp. 17:1–17:14. Schloss Dagstuhl (Jul 2022). https://doi.org/10.4230/LIPIcs.ITC.2022.17

9. Budroni, A., Chi-Domínguez, J.J., D'Alconzo, G., Scala, A.J.D., Kulkarni, M.: Don't use it twice! Solving relaxed linear equivalence problems. In: Chung, K.M., Sasaki, Y. (eds.) ASIACRYPT 2024, Part VIII. LNCS, vol. 15491, pp. 35–65. Springer, Singapore (Dec 2024). https://doi.org/10.1007/978-981-96-0944-4_2
10. Budroni, A., Chi-Domínguez, J.J., D'Alconzo, G., Di Scala, A.J., Kulkarni, M.: Don't use it twice! solving relaxed linear equivalence problems. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 35–65. Springer (2024)
11. Bunn, P., Katz, J., Kushilevitz, E., Ostrovsky, R.: Efficient 3-party distributed ORAM. In: Galdi, C., Kolesnikov, V. (eds.) Security and Cryptography for Networks - 12th International Conference, SCN 2020, Amalfi, Italy, September 14-16, 2020, Proceedings. Lecture Notes in Computer Science, vol. 12238, pp. 215–232. Springer (2020). https://doi.org/10.1007/978-3-030-57990-6_11, https://doi.org/10.1007/978-3-030-57990-6_11
12. Bunn, P., Kushilevitz, E., Ostrovsky, R.: Cnf-fss and its applications. In: Hanaoka, G., Shikata, J., Watanabe, Y. (eds.) Public-Key Cryptography – PKC 2022. pp. 283–314. Springer International Publishing, Cham (2022). https://doi.org/https://doi.org/10.1007/978-3-030-97121-2_11
13. Carrell, J.B.: Fundamentals of linear algebra. The University of British Columbia (2005)
14. Corrigan-Gibbs, H., Boneh, D., Mazières, D.: Riposte: An anonymous messaging system handling millions of users. In: 2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015. pp. 321–338. IEEE Computer Society (2015). https://doi.org/10.1109/SP.2015.27, https://doi.org/10.1109/SP.2015.27
15. Couteau, G., Kumar, N.: 10-party sublinear secure computation from standard assumptions. In: Reyzin, L., Stebila, D. (eds.) Advances in Cryptology - CRYPTO 2024 - 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2024, Proceedings, Part IX. Lecture Notes in Computer Science, vol. 14928, pp. 39–73. Springer (2024). https://doi.org/10.1007/978-3-031-68400-5_2, https://doi.org/10.1007/978-3-031-68400-5_2
16. Couteau, G., Kumar, N., Ye, X.: Multiparty homomorphic secret sharing and more from LPN and MQ. IACR Cryptol. ePrint Arch. p. 966 (2025), https://eprint.iacr.org/2025/966
17. Damie, M., Hahn, F., Peter, A., Ramon, J.: DDH-based schemes for multiparty function secret sharing. Cryptology ePrint Archive, Paper 2025/1805 (2025), https://eprint.iacr.org/2025/1805
18. Damie, M., Hahn, F., Peter, A., Ramon, J.: Eliminating exponential key growth in prg-based distributed point functions (2025), https://arxiv.org/abs/2509.22022
19. Dauterman, E., Feng, E., Luo, E., Popa, R.A., Stoica, I.: DORY: an encrypted search system with distributed trust. In: 14th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2020, Virtual Event, November 4-6, 2020. pp. 1101–1119. USENIX Association (2020), https://www.usenix.org/conference/osdi20/presentation/dauterman-dory
20. Dauterman, E., Rathee, M., Popa, R.A., Stoica, I.: Waldo: A private time-series database from function secret sharing. In: 43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, May 22-26, 2022. pp. 2450–2468. IEEE (2022). https://doi.org/10.1109/SP46214.2022.9833611, https://doi.org/10.1109/SP46214.2022.9833611

21. Dodis, Y., Halevi, S., Rothblum, R.D., Wichs, D.: Spooky encryption and its applications. In: Robshaw, M., Katz, J. (eds.) Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part III. Lecture Notes in Computer Science, vol. 9816, pp. 93–122. Springer (2016). https://doi.org/10.1007/978-3-662-53015-3_4, https://doi.org/10.1007/978-3-662-53015-3_4
22. Falk, B.H., Lu, S., Ostrovsky, R.: DURASIFT: A robust, decentralized, encrypted database supporting private searches with complex policy controls. In: Cavallaro, L., Kinder, J., Domingo-Ferrer, J. (eds.) Proceedings of the 18th ACM Workshop on Privacy in the Electronic Society, WPES@CCS 2019, London, UK, November 11, 2019. pp. 26–36. ACM (2019). https://doi.org/10.1145/3338498.3358651, https://doi.org/10.1145/3338498.3358651
23. Gilboa, N., Ishai, Y.: Distributed point functions and their applications. In: Nguyen, P.Q., Oswald, E. (eds.) Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8441, pp. 640–658. Springer (2014). https://doi.org/10.1007/978-3-642-55220-5_35, https://doi.org/10.1007/978-3-642-55220-5_35
24. Goel, A., Wang, M., Wang, Z.: Multiparty distributed point functions. In: Kalai, Y.T., Kamara, S.F. (eds.) Advances in Cryptology - CRYPTO 2025 - 45th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2025, Proceedings, Part IV. Lecture Notes in Computer Science, vol. 16003, pp. 140–173. Springer (2025). https://doi.org/10.1007/978-3-032-01884-7_5, https://doi.org/10.1007/978-3-032-01884-7_5
25. Guo, X., Yang, K., Wang, X., Zhang, W., Xie, X., Zhang, J., Liu, Z.: Half-tree: Halving the cost of tree expansion in COT and DPF. In: Hazay, C., Stam, M. (eds.) EUROCRYPT 2023, Part I. LNCS, vol. 14004, pp. 330–362. Springer, Cham (Apr 2023). https://doi.org/10.1007/978-3-031-30545-0_12
26. Khalesi, A., Akhbari, B.: The distributed multi-user point function (2025), https://arxiv.org/abs/2501.10251
27. Koyama, K., Maurer, U.M., Okamoto, T., Vanstone, S.A.: New public-key schemes based on elliptic curves over the ring z_n. In: Feigenbaum, J. (ed.) Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings. Lecture Notes in Computer Science, vol. 576, pp. 252–266. Springer (1991). https://doi.org/10.1007/3-540-46766-1_20, https://doi.org/10.1007/3-540-46766-1_20
28. Kruglik, S., Dau, S.H., Kiah, H.M., Wang, H., Zhang, L.F.: Verifiable information-theoretic function secret sharing. Cryptology ePrint Archive, Report 2024/453 (2024), https://eprint.iacr.org/2024/453
29. Kumar, C., Patranabis, S., Mukhopadhyay, D.: Compact key function secret sharing with non-linear decoder. IACR Commun. Cryptol. **1**(2), 8 (2024). https://doi.org/10.62056/A3C3C3W9P, https://doi.org/10.62056/a3c3c3w9p
30. Külaots, E., Krips, T., Eerikson, H., Pullonen-Raudvere, P.: SLAMP-FSS: Two-party multi-point function secret sharing from simple linear algebra. Cryptology ePrint Archive, Paper 2024/1394 (2024), https://eprint.iacr.org/2024/1394
31. Li, J., Ke, P., Zhang, L.F.: Efficient information-theoretic distributed point functions with general output groups. Designs, Codes and Cryptography pp. 1–30 (2025)

32. Li, W.M., Zhang, L.F.: Towards efficient information-theoretic function secret sharing. IEEE Access **8**, 28512–28523 (2020). https://doi.org/10.1109/ACCESS.2020.2971722, https://doi.org/10.1109/ACCESS.2020.2971722
33. Newman, Z., Servan-Schreiber, S., Devadas, S.: Spectrum: High-bandwidth anonymous broadcast. In: Phanishayee, A., Sekar, V. (eds.) 19th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2022, Renton, WA, USA, April 4-6, 2022. pp. 229–248. USENIX Association (2022), https://www.usenix.org/conference/nsdi22/presentation/newman
34. Persichetti, E., Santini, P.: A new formulation of the linear equivalence problem and shorter less signatures. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 351–378. Springer (2023)
35. Rabin, M.O.: Digitalized signatures and public-key functions as intractable as factorization (1979)
36. Rivest, R.L., Shamir, A., Adleman, L.M.: A method for obtaining digital signatures and public-key cryptosystems. Communications of the Association for Computing Machinery **21**(2), 120–126 (Feb 1978). https://doi.org/10.1145/359340.359342
37. Shamir, A.: On the generation of multivariate polynomials which are hard to factor. In: Kosaraju, S.R., Johnson, D.S., Aggarwal, A. (eds.) Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA. pp. 796–804. ACM (1993). https://doi.org/10.1145/167088.167291, https://doi.org/10.1145/167088.167291
38. Shoup, V.: Lower bounds for discrete logarithms and related problems. In: Fumy, W. (ed.) Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11-15, 1997, Proceeding. Lecture Notes in Computer Science, vol. 1233, pp. 256–266. Springer (1997). https://doi.org/10.1007/3-540-69053-0_18, https://doi.org/10.1007/3-540-69053-0_18
39. Verbel, J., Baena, J., Cabarcas, D., Perlner, R., Smith-Tone, D.: On the complexity of "superdetermined" minrank instances. In: International Conference on Post-Quantum Cryptography. pp. 167–186. Springer (2019)
40. Weger, V., Gassner, N., Rosenthal, J.: A survey on code-based cryptography. arXiv preprint arXiv:2201.07119 (2022)
41. Williams, H.C.: Some public key crypto-functions as intractable as factorization. In: Blakley, G.R., Chaum, D. (eds.) Advances in Cryptology, Proceedings of CRYPTO '84, Santa Barbara, California, USA, August 19-22, 1984, Proceedings. Lecture Notes in Computer Science, vol. 196, pp. 66–70. Springer (1984). https://doi.org/10.1007/3-540-39568-7_7, https://doi.org/10.1007/3-540-39568-7_7

A Assumptions with duplicated variables

For space concerns, we put two of our assumptions here. We note that these assumptions are really similar to the code based-assumptions that we introduced in Subsection 2.3. The difference is essentially that we duplicate a number of secret variables.

Assumption 5 ((R, k, τ, n, i, u)-Switching-Decisional Linear Double Code Equivalence With Common Submodules).

Let R be a ring. Let $\boldsymbol{c_1}^{(b)}, \ldots, \boldsymbol{c_{\tau+n}}^{(b)} \xleftarrow{\$} R^k$ for $b \in \{1, 2\}$. Let $B_0, B_1 \in R^{k\times k}$ be randomly sampled under the condition that $B_c\langle \boldsymbol{c_1}^{(b)}, \ldots, \boldsymbol{c_{\tau+i}}^{(b)}\rangle \subseteq$

$\langle \boldsymbol{c_1}^{(b)}, \ldots, \boldsymbol{c_{\tau+i+1}}^{(b)} \rangle$ *for any* $i \in [0, n-1]$ *and for* $b \in \{1,2\}$. *Let* $\boldsymbol{d}^{(b)} \in R^k$ *be an arbitrary element such that* $B_0 \boldsymbol{d}^{(b)} \in \langle \boldsymbol{c_1}^{(b)}, \ldots, \boldsymbol{c_{\tau+i}}^{(b)} \rangle$ *and* $B_1 \boldsymbol{d}^{(b)} \notin \langle \boldsymbol{c_1}^{(b)}, \ldots, \boldsymbol{c_{\tau+i}}^{(b)} \rangle$ *for* $b \in \{1,2\}$.

Choose $\overline{E}_1, \overline{E}_2, \overline{E}_3, \overline{E}_4$ *randomly under the constraint that their codimension size is* u, *that for all* $c \in [1,4]$, *for* $b \in \{1,2\}$, $\overline{E}_c \langle \boldsymbol{c_1}^{(b)}, \ldots, \boldsymbol{c_{\tau+i+1}}^{(b)} \rangle \subseteq \langle \boldsymbol{c_1}^{(b)}, \ldots, \boldsymbol{c_{\tau+i+1}}^{(b)} \rangle$ *and that* $\overline{E}_2 B_1 \boldsymbol{d}^{(b)} \in \langle \boldsymbol{c_1}^{(b)}, \ldots, \boldsymbol{c_{\tau+i}}^{(b)} \rangle$. *Choose* E *randomly under the constraint that* $E \langle \boldsymbol{c_1}^{(b)}, \ldots, \boldsymbol{c_{\tau+i}}^{(b)} \rangle \subseteq \langle \boldsymbol{c_1}^{(b)}, \ldots, \boldsymbol{c_{\tau+i}}^{(b)} \rangle$ *and that* $\boldsymbol{d}^{(b)} \in \operatorname{Im} E$ *for* $b \in \{1,2\}$. *Choose* $F \xleftarrow{\$} GL_k(N)$ *at random.*

Choose at random a bit α_i. *If* $\alpha_i = 1$, *set* $\pi_i = \pi_{id}$, *else set* $\pi_i = \pi_{sw}$.

Now compute

$$\begin{pmatrix} A_{i,0} \\ A_{i,1} \end{pmatrix} \leftarrow \begin{pmatrix} \overline{E}_{1+3\pi_i(0)} & \overline{E}_{2+\pi_i(0)} \\ \overline{E}_{2+\pi_i(1)} & \overline{E}_{1+3+\pi_i(1)} \end{pmatrix} \begin{pmatrix} B_{\pi_i(0)} \\ B_{\pi_i(1)} \end{pmatrix} E_i F_i^{-1}.$$

Then the (R, k, τ, n, j, u)*-Switching-Decisional Linear Double Code Equivalence With Common Submodules Assumption states that if an adversary who is given* B_0, B_1, $A_{i,0}$ *and* $A_{i,1}$ *can guess* α_i *better than random chance, then there is an extractor* `Extr` *who can output either an element of* $\langle \boldsymbol{c_1}^{(1)}, \ldots, \boldsymbol{c_{\tau+i}}^{(1)}, \boldsymbol{c_1}^{(2)}, \ldots, \boldsymbol{c_{\tau+i}}^{(2)} \rangle$, *some element of* $\boldsymbol{d}^{(b)} + \langle \boldsymbol{c_1}^{(1)}, \ldots, \boldsymbol{c_{\tau+i}}^{(1)}, \boldsymbol{c_1}^{(2)}, \ldots, \boldsymbol{c_{\tau+i}}^{(2)} \rangle$ *or such an element* $\boldsymbol{d'}^{(b)}$ *that* $E\boldsymbol{d'}^{(b)} \in \boldsymbol{d}^{(b)} + \langle \boldsymbol{c_1}^{(1)}, \ldots, \boldsymbol{c_{\tau+i}}^{(1)}, \boldsymbol{c_1}^{(2)}, \ldots, \boldsymbol{c_{\tau+i}}^{(2)} \rangle$ *or an element from* $\langle \boldsymbol{c_1}^{(1)}, \ldots, \boldsymbol{c_{\tau+i}}^{(1)}, \boldsymbol{c_1}^{(2)}, \ldots, \boldsymbol{c_{\tau+i}}^{(2)} \rangle^{\perp}$.

Assumption 6 ((R, k, τ, n)**-Multi-Switching-Decisional Linear Double Code Equivalence With Common Submodules).**

Let R *be a ring. Let* $\boldsymbol{c_1}^{(b)}, \ldots, \boldsymbol{c_{\tau+n}}^{(b)} \xleftarrow{\$} R^k$. *Let* $B_0, B_1 \in R^{k \times k}$ *be such that* $B_b \langle \boldsymbol{c_1}^{(b)}, \ldots, \boldsymbol{c_{\tau+i}}^{(b)} \rangle \subseteq \langle \boldsymbol{c_1}^{(b)}, \ldots, \boldsymbol{c_{\tau+i+1}}^{(b)} \rangle$ *for any* $i \in [0, n-1]$. *For* $i \in [1, n]$, *let* $\boldsymbol{d_{3(i-1)+1}} \in R^k$ *be an arbitrary element such that* $B_0 \boldsymbol{d_{3(i-1)+1}}^{(b)} \in \langle \boldsymbol{c_1}^{(b)}, \ldots, \boldsymbol{c_{\tau+i}}^{(b)} \rangle$ *and* $B_1 \boldsymbol{d_{3(i-1)+1}}^{(b)} \notin \langle \boldsymbol{c_1}^{(b)}, \ldots, \boldsymbol{c_{\tau+i}}^{(b)} \rangle$.

For $i \in [1, n]$, *choose* $\overline{E}_{i,1}, \overline{E}_{i,2}, \overline{E}_{i,3}, \overline{E}_{i,4}$ *randomly under the constraint that their codimension size is* u, *that for all* $c \in [1,4]$, $\overline{E}_{i,c} \langle \boldsymbol{c_1}^{(b)}, \ldots, \boldsymbol{c_{\tau+i+1}}^{(b)} \rangle \subseteq \langle \boldsymbol{c_1}^{(b)}, \ldots, \boldsymbol{c_{\tau+i+1}}^{(b)} \rangle$ *and that* $\overline{E}_{i,2} B_1 \boldsymbol{d}^{(b)} \in \langle \boldsymbol{c_1}^{(b)}, \ldots, \boldsymbol{c_{\tau+i}}^{(b)} \rangle$. *For* $i \in [1, n]$, *choose* E_i *randomly under the constraint that* $E_i \langle \boldsymbol{c_1}^{(b)}, \ldots, \boldsymbol{c_{\tau+i}}^{(b)} \rangle \subseteq \langle \boldsymbol{c_1}^{(b)}, \ldots, \boldsymbol{c_{\tau+i}}^{(b)} \rangle$ *and that* $\boldsymbol{d_{3(i-1)+1}}^{(b)} \in \operatorname{Im} E_i$. *For* $i \in [1, n]$, *choose* $F_i \xleftarrow{\$} GL_k(N)$ *at random.*

For $i \in [1, n]$, *denote by* $\boldsymbol{d_{3(i-1)}}^{(b)}$ *some such vector that* $E_i \boldsymbol{d_{3(i-1)}}^{(b)} \in \boldsymbol{d_{3(i-1)+1}}^{(b)} + \langle \boldsymbol{c_1}^{(b)}, \ldots, \boldsymbol{c_{\tau+i}}^{(b)} \rangle$. *Now choose for* $i \in [1, n]$ *a* $H_i \xleftarrow{\$} GL_k(n)$ *under the constraint that*

$H_i F_i^{-1} \overline{E}_{i,4} B_1 \boldsymbol{d_{3(i-1)+1}}^{(b)} = F_{i+1}^{-1} \boldsymbol{d_{3i}}^{(b)}$.

Let the adversary $\mathcal{A}$ *choose two* n*-bit bitstrings* $\alpha^{[1]}$ *and* $\alpha^{[2]}$. *Choose at random* $\overline{b} \leftarrow \{1,2\}$. *Let* $\alpha = \alpha^{[b]}$ *and let* $\alpha = \alpha_1, \ldots, \alpha_n$.

Now, for every $i \in [1, n]$, *if* $\alpha_i = 1$, *set* $\pi_i = \pi_{id}$, *else set* $\pi_i = \pi_{sw}$.

Now compute

$$\begin{pmatrix} A_{i,0} \\ A_{i,1} \end{pmatrix} \leftarrow \begin{pmatrix} \overline{E}_{i,1+3\pi_i(0)} & \overline{E}_{i,2+\pi_i(0)} \\ \overline{E}_{i,2+\pi_i(1)} & \overline{E}_{i,1+3+\pi_i(1)} \end{pmatrix} \begin{pmatrix} B_{\pi_i(0)} \\ B_{\pi_i(1)} \end{pmatrix} E_i F_i^{-1}.$$

The adversary is given $\{A_{i,0}\}_{i=1}^{n}$,$\{A_{i,1}\}_{i=1}^{n}$ *and* $\{H_iF_i^{-1}\}_{i=1}^{n}$. *The* (R, k, τ, n)*-Multi-Switching-Decisional Linear Code Equivalence With Common Submodules-Assumption states that if the adversary can guess* $\bar{b}$ *better than random, then there exists an extractor* `Extr`*, who, given* $\{A_{i,0}\}_{i=1}^{n}$,$\{A_{i,1}\}_{i=1}^{n}$ *and* $\{H_iF_i^{-1}\}_{i=1}^{n}$ *can produce either an element of* $\langle \boldsymbol{c_1}^{(1)}, \ldots, \boldsymbol{c_{\tau+i}}^{(1)}, \boldsymbol{c_1}^{(2)}, \ldots, \boldsymbol{c_{\tau+i}}^{(2)} \rangle$*, some* $\boldsymbol{d_{3(i-1)}} + \langle \boldsymbol{c_1}^{(1)}, \ldots, \boldsymbol{c_{\tau+i}}^{(1)}, \boldsymbol{c_1}^{(2)}, \ldots, \boldsymbol{c_{\tau+i}}^{(2)} \rangle$ *or some* $\boldsymbol{d_{3(i-1)+1}} + \langle \boldsymbol{c_1}^{(1)}, \ldots, \boldsymbol{c_{\tau+i}}^{(1)}, \boldsymbol{c_1}^{(2)}, \ldots, \boldsymbol{c_{\tau+i}}^{(2)} \rangle$*, or an element from* $\langle \boldsymbol{c_1}^{(1)}, \ldots, \boldsymbol{c_{\tau+i}}^{(1)}, \boldsymbol{c_1}^{(2)}, \ldots, \boldsymbol{c_{\tau+i}}^{(2)} \rangle^{\perp}$.

B Tools

In this subsection we list a few useful constructions for obtaining matrices with desired properties.

B.1 For a set of independent vectors $\{\boldsymbol{x_j}\}_{\boldsymbol{j=1}}^{\boldsymbol{k'}}$, and a set of vectors $\{\boldsymbol{y_j}\}_{\boldsymbol{j=1}}^{\boldsymbol{k'}}$obtaining a matrix that maps each $\boldsymbol{x_j}$ to $\boldsymbol{y_j}$

We note that a $k \times k$ matrix M can be uniquely described by a set of independent vectors $\{\boldsymbol{x_j}\}_{j=1}^{k}$, and a set of vectors $\{\boldsymbol{y_j}\}_{j=1}^{k}$ such that $M\boldsymbol{x_j} = \boldsymbol{y_j}$ for all $j \in \{1, \ldots, k\}$. Note that the $\{\boldsymbol{y_j}\}_{j=1}^{k}$ do not necessarily have to be independent. We note that if we denote the desired matrix by M, and we denote by X the matrix where the column vectors are the $\{\boldsymbol{x_j}\}_{j=1}^{k}$, and we denote by Y the matrix where the column vectors are the $\{\boldsymbol{y_j}\}_{j=1}^{k}$, then the property that we want M to satisfy is $MX = Y$. Now we can compute $M = YX^{-1}$.

Note that if $k' < k$, then we can choose the other vectors of X and Y arbitrarily. These other vectors can also be used to give the matrix some other properties we might desire. If we want to give X a kernel of dimension v, then we can add the vectors $\boldsymbol{x_{k'+1}}, \ldots, \boldsymbol{x_{k'+v}}$ with the corresponding $\boldsymbol{y_{k'+1}} = \cdots = \boldsymbol{y_{k'+v}} = \boldsymbol{0}$. If we want that the matrix would have a hull vector, that is, such a vector $\boldsymbol{z}$ such that $\boldsymbol{z}$ belongs to the code generated by the matrix and also to the kernel of the matrix, we can simply add the pairs $(\boldsymbol{x_{k'+v+1}}, \boldsymbol{z})$ and $(\boldsymbol{z}, \boldsymbol{0})$ to the constraints where $\boldsymbol{x_{k'+v+1}}$ is arbitrary, that is, add the constraints that the matrix should map $\boldsymbol{x_{k'+v+1}}$ to $\boldsymbol{z}$ and $\boldsymbol{z}$ to zero. Thus, we can obtain a hull of arbitrarily large dimension, if we want to. We omit the parameter for constructing the hull from our current algorithm, as it is likely unnecessary, as a nontrivial hull is usually only relevant to Permutation Equivalence Code problems [34], and in our case our security is more related to Linear Code Equivalence problem where this is not relevant.

The algorithm is given in Algorithm 5. For ease of presentation, we assume that the $\{\boldsymbol{x_j}\}_{j=1}^{k'}$ are linearly independent. Note that we check whether the matrix X is invertible and restart the algorithm if it is not, but this step can be omitted in practice, as the probability of X being noninvertible is negligibly small. This algorithm also allows us to obtain a matrix with a kernel size at least v.

Algorithm 5: MatFromVecPairs

Input : $N, k, \{\boldsymbol{x_j}\}_{j=1}^{k'}, \{\boldsymbol{y_j}\}_{j=1}^{k'}, v$

$\boldsymbol{x_{k'+1}}, \ldots, \boldsymbol{x_k} \xleftarrow{\$} \mathbb{Z}_N^k$
if $v > 0$ **then**
  $\boldsymbol{y_{k'+1}}, \ldots, \boldsymbol{y_{k'+v}} \leftarrow \boldsymbol{0} \in \mathbb{Z}_N^k$
  $\boldsymbol{y_{k'+v+1}}, \ldots, \boldsymbol{y_k} \xleftarrow{\$} \mathbb{Z}_N^k$
else
  $\boldsymbol{y_{k'+1}}, \ldots, \boldsymbol{y_k} \xleftarrow{\$} \mathbb{Z}_N^k$
$X \leftarrow \mathsf{ToMat}(\boldsymbol{x_1}, \ldots, \boldsymbol{x_k})$
if X *noninvertible* **then**
  start again
$Y \leftarrow \mathsf{ToMat}(\boldsymbol{y_1}, \ldots, \boldsymbol{y_k})$
$M \leftarrow YX^{-1}$
return M;

Algorithm 6: sampleFrom

Input : $\{\boldsymbol{x_j}\}_{j=1}^{k'}, t$

for $i = 1, i++, i \leq t$ **do**
  for $j = 1, j++, j \leq k'$ **do**
    $\lambda_{i,j} \xleftarrow{\$} \mathbb{Z}_N$
  $\boldsymbol{y_i} \leftarrow \sum_{j=1}^{k'} \lambda_{i,j} \boldsymbol{x_j}$;
$X \leftarrow \mathsf{ToMat}(\boldsymbol{x_1}, \ldots, \boldsymbol{x_k})$
return $\{\boldsymbol{y_i}\}_{i=1}^{t}$;

B.2 Sample random vectors from a submodule

Algorithm 6 lets us sample t vectors from a submodule given by a set of basis vectors $\{\boldsymbol{x_i}_{j=1}^{k'}\}$.

B.3 Obtaining a matrix with desired eigenvalues and kernel

We will see that a lot about the behaviour of a square matrix can be described using its eigenvectors and eigenvalues. We thus will describe how to obtain a matrix with desired eigenvector-eigenvalue pairs. We note that we can consider kernel vectors as a type of eigenvectors with the corresponding eigenvalue 0. Suppose we wish to obtain a $k \times k$ matrix with rank $r \geq k$ and we wish that the matrix would have linearly independent eigenvectors $\boldsymbol{q_1}, \ldots, \boldsymbol{q_v}$ with corresponding nonzero eigenvalues $\lambda_1, \ldots, \lambda_v$ where $v \leq r$ and kernel vectors $\boldsymbol{s_1}, \ldots, \boldsymbol{s_u}$ with $u \leq n - r$. If $v < r$ and/or $u < n - r$, we will choose the other eigenvectors or kernel vectors randomly. We will assume that the given eigenvectors and kernel vectors are linearly independent.

We will use the fact that if Q is an invertible matrix with column vectors $\overline{\boldsymbol{q_1}}, \ldots, \overline{\boldsymbol{q_n}}$, and $D = \mathsf{diag}(\mu_1, \ldots, \mu_k)$, then the matrix $A = QDQ^{-1}$ has the

eigenpairs $(\overline{\boldsymbol{q_1}}, \mu_1), \ldots, (\overline{\boldsymbol{q_k}}, \mu_k)$. This allows us to easily construct the desired matrix in Algorithm 7

Algorithm 7: matFromEigenVecs
Input : $\{(\boldsymbol{q_j}, \mu_j)\}_{j=1}^{n}$,
$D \leftarrow \mathsf{diag}(\mu_1, \ldots, \mu_n)$;
$Q \leftarrow \mathsf{ToMat}(\boldsymbol{q_1}, \ldots, \boldsymbol{q_n})$;
$A \leftarrow QDQ^{-1}$;
return A;

C Finding elements of $\tilde{C}_i$,$\tilde{C}_i^{\perp}$ or any d_j implies factoring

In this section we give the full algorithm, some extra preliminaries and describe how to show that finding certain values in our full algorithm is as hard as factoring N.

C.1 Hardness of Factoring

Assumption 7 (Factoring). *For integers $N = pq$ where p and q are large prime numbers, it is computationally infeasible to find p and q given N.*

In the following, we are operating over the ring $\mathbb{Z}_{pq}$. Note that, if factoring is hard, many operations, that are easy over finite fields, are hard over $\mathbb{Z}_{pq}$. For example, taking the square root, that is the core of Rabin's cryptosystem [35] relies. The security of a version of RSA [36] relies on the hardness of taking cube roots. There are cryptosystems that are based on constructing elliptic curves over $\mathbb{Z}_{pq}$, where breaking the security is as hard as factoring [27]. We note that, in general, finding roots of general polynomials is hard.

We note that there are a number of works, for example, [41] or [27] which consider various operations that are hard over the ring $\mathbb{Z}_{pq}$ where p and q are large primes, the most famous of which is the RSA cryptosystem [36]. However, it is not the only one. Rabin's cryptosystem [35] is based on the fact that computing square roots in $\mathbb{Z}_{pq}$ is as hard as factoring, similarly, the work of [27] has a cryptosystem where an elliptic curve is defined over $\mathbb{Z}_{pq}$ and it is shown that being able to solve some equations over that ring is equivalent to factoring. Shamir in [37] shows, among other things, that finding roots of general polynomials is as hard as factoring in $\mathbb{Z}_{pq}$.

More precisely, the following definition and result are given in [37].

Definition 2 (Randomly solvable [37]). *A system of multivariate rational equations of fixed degree d in the form $R_i(a_1, \ldots, a_K) = v_i \pmod N$ for $i = 1, \ldots, l$ is said to be randomly solvable, if the v_i values are computed by substituting uniformly distributed random values for the variables $a_1, \ldots, a_K$ in the R_i functions.*

Theorem 4 ([37]). *If a randomly solvable system of equations is invariant under some invertible rational transformation of the variables which is not the identity, and the system is known to have at most a polynomial number of solutions, then finding any one of them is at least as difficult as the factorization of the modulus N.*

The following is a short intuition about why this result holds. If we have some two different solutions x_1 and x_2 to some system of equations over $\mathbb{Z}_N$, where $N = pq$ and the factorization is unknown, then over $\mathbb{Z}_p$ we have the solutions $x_{1,p}$ and $x_{2,p}$ and over $\mathbb{Z}_q$ we have the solutions $x_{1,q}$ and $x_{2,q}$ with $x_1 = x_{1,p} \pmod p$, $x_2 = x_{2,p} \pmod p$, $x_1 = x_{1,q} \pmod q$ and $x_2 = x_{2,q} \pmod q$. If $x_{1,p} = x_{2,p}$ then $x_1 = x_2 \pmod p$ but $x_1 \neq x_2 \pmod N$ and thus $x_1 - x_2$ is a multiple of p and thus $gcd(N, x_1 - x_2) = p$ and we can factor N. Thus let us assume that $x_{1,p} \neq x_{2,p}$. Analogously now assume that $x_{1,q} \neq x_{2,q}$.

Hence, the system must have two other solutions y_1, y_2 such that $y_1 = x_{1,p} \pmod p$ and $y_1 = x_{2,q} \pmod q$ and $y_2 = x_{2,p} \pmod p$ and $y_2 = x_{1,q} \pmod q$. The pair (x_1, x_2) is information-theoretically indistinguishable from (y_1, y_2) since their difference is just the decision of how we choose to enumerate $x_{1,q}, x_{2,q}$.

Thus, if we have an algorithm that outputs the solutions to the system of equations, we obtain both x_1, x_2, y_1, y_2. However, $x_1 - y_1$ is a multiple of q but not a multiple of N and thus $gcd(x_1 - y_1, N) = q$, thus allowing for factorization.

C.2 General discussion of strategy for the proof to work

In order for this kind of proof to work, we need to make sure that there is only a polynomial number of solutions, that we can establish some kind of transformation of variables and that, given one part of a solution, we can extract all the other parts of the solution from it.

For the transformation, we will simply create two groups of variables, that we denote with variable$^{(1)}$ and variable$^{(2)}$ that will behave exactly the same, and the transformation would be simply switching them.

Concerning the polynomial number of solutions, we will give some extra information to the solver in this proof that they do not have in the real world. Obviously giving more information does not make solving a problem harder. We can also strategically give extra information here so that we could extract one solution from parts of a solution.

In order for the symmetry to work, this fixed information needs to be the same for both variable$^{(1)}$ and variable$^{(2)}$. This also means that these relations have to hold for both cases.

The large picture of our approach is the following. We give, in addition to the $A_{i,0}$ and $A_{i,1}$ also the $\overline{E}_{i,c}$, B_0,B_1,E_i, F_i, α and β. We also will need to give some additional information.

We will describe how we can, in general, learn all the secrets from just one part of the secret. Due to this we have to alter our construction slightly.

The main challenge here is that the secret submodules $\tilde{C}_i^{(b)}$ are exponentially large, and thus describing an algorithm that can recover any $\boldsymbol{d_i}$ from any element of the secret submodules is somewhat involved.

We first start from the point where we assume that we have got some element $\boldsymbol{c_0''}$ of $\tilde{C}_n^{(1)} + \tilde{C}_n^{(2)}$ such that $\boldsymbol{r} \notin \tilde{C}_n^{(1)}, \tilde{C}_n^{(2)}$. With an overwhelming probability, the only proper submodules $\tilde{C}$ of $\tilde{C}_n^{(1)} + \tilde{C}_n^{(2)}$ with the property that $\overline{E}_{n,1}\tilde{C} = \overline{E}_{n,3}\overline{E}_{n,3}$ are $\tilde{C}_n^{(1)}$ and $\tilde{C}_n^{(2)}$.

Naturally, they also both preserve $\tilde{C}_n^{(1)} + \tilde{C}_n^{(2)}$.

This means that if $\boldsymbol{c_0''}$ is not in a module generated by some subset of size $\tau' < 2n + 2\tau$ of eigenvectors of $\overline{E}_{n,1}$, then we can, by recursively computing $\boldsymbol{c_i''} \leftarrow \overline{E}_{n,1}\boldsymbol{c_{i-1}''}$, obtain $2n + 2\tau$ basis vectors for $\tilde{C}_n^{(1)} + \tilde{C}_n^{(2)}$. If $\boldsymbol{r}$ however is in a module generated by some strict subset of eigenvectors of $\overline{E}_{n,1}$, then we get a submodule of order τ' like this, which is easy to detect. In that case, however, with an overwhelming probability, $\boldsymbol{r}$ is not in a module generated by some subset of size $\tau' < 2n + 2\tau$ of eigenvectors of $\overline{E}_{n,3}$, and in that case we generate the $\boldsymbol{c_i''}$ by using $\overline{E}_{n,3}$. We then take the reduced row echelon form of $\boldsymbol{c_1''}, \ldots, \boldsymbol{c_{2n+2\tau}''}$ which we denote by R. Note that now we have a format that does not depend on the original $\boldsymbol{r}$. However, now we have the issue that if we want to compute the original $\boldsymbol{c_i}^{(1)}$ from this matrix R, then the naive construction of giving some transformation matrix as auxiliary information that would transform our RREF to the $\boldsymbol{c_i}^{(1)}, \boldsymbol{c_i}^{(2)}$ has the issue that it is not symmetrical towards the variable$^{(1)}$ and variable$^{(2)}$.

We overcome this issue, by instead giving a transformation matrix V that allows us to obtain from the R a square matrix U that preserves a submodule of dimension 2 and two vectors h_1 and h_2 from that submodule, and an additional hint $\overline{\lambda}$. We will show that one can thus recover two eigenvectors of U that belong to that preserved submodule. Now we denote the two eigenvectors by $\boldsymbol{c_1'}^{(1)}$ and $\boldsymbol{c_1'}^{(2)}$. We construct the U, h_1,h_2 and $\overline{\lambda}$ in such a way that $\boldsymbol{c_1'}^{(b)} \in \tilde{C}_0^{(b)}$ for $b \in \{1, 2\}$. However, this construction is symmetrical towards $\boldsymbol{c_1'}^{(1)}$ and $\boldsymbol{c_1'}^{(2)}$, thus allowing the general technique to work.

Then, we define the canonical basis of $\tilde{C}_0^{(b)}$ for $b \in \{1, 2\}$. For this, let us define two helper matrices B_2 and E_1'. The matrix B_2 has the property that $B_2\tilde{C}_{i-1}^{(b)} \in \tilde{C}_i^{(b)}$ for all $i \in [1, n]$ and $b \in \{1, 2\}$ and E_1' is a randomly chosen matrix such that $E_1'\tilde{C}_0^{(b)} = \tilde{C}_0^{(b)}$. More specifically, we sample at random variables ${\lambda_{i,j,2}}^{(b)}$ and ${\lambda'''_{i',j}}^{(b)}$ such that $B_2\boldsymbol{c_i'}^{(b)} = \sum_{j=1}^{\tau+1} {\lambda_{i,j,2}}^{(b)}\boldsymbol{c_j'}^{(b)}$ for $i \in [1, \tau]$, $b \in \{1, 2\}$, and $B_2\boldsymbol{c_i'}^{(b)} = \sum_{j=1}^{\tau+i+1} {\lambda_{i,j,2}}^{(b)}\boldsymbol{c_j'}^{(b)}$ for $i \in [\tau + 1, \tau + n]$, $b \in \{1, 2\}$, and $E_1'\boldsymbol{c_{i'}'}^{(b)} = \sum_{j=1}^{\tau+i} {\lambda'''_{i',j}}^{(b)}\boldsymbol{c_j'}^{(b)}$ for $i \in [1, n]$, $i' \in [1, \tau]$, $b \in \{1, 2\}$.

Now we can define the canonical basis of $\tilde{C}_0^{(b)}$ for $b \in \{1, 2\}$ We do this by computing $\boldsymbol{c_i'}^{(b)} \leftarrow E_1'\boldsymbol{c_{i-1}'}^{(b)}$ for $i \in [2, \tau - 1]$. It is easy to see that we get τ vectors that are in $\tilde{C}_0^{(b)}$ that way. With an overwhelming probability, these are all linearly independent and thus form a basis for $\tilde{C}_0^{(b)}$. After that we compute

$\boldsymbol{c'_i}^{(b)} \leftarrow B_1 \boldsymbol{c'_{i-1}}^{(b)}$ for $i \in [\tau, \tau+n]$. It is easy to see that $\{\boldsymbol{c'_j}^{(b)}\}_{j=1}^{\tau+i}$ is a basis for $\tilde{C}_i^{(b)}$, and we call that the *canonical basis*.

(We note that although we could use $\overline{E}_1$ and B_0, instead, for example, we need to construct the matrix in such a way that they behave in a certain way for $\boldsymbol{d_j}^{(b)}$ and we will construct the $\boldsymbol{d_j}^{(b)}$ in a way dependent from $\boldsymbol{c'_i}$, and thus, to avoid a circular dependency, we use different matrices here)

Concerning $\boldsymbol{d_0}^{(b)}$, we construct them in the following way. We fix some vector $\boldsymbol{u_0} \notin \tilde{C}_n^{(1)} + \tilde{C}_n^{(2)}$ and some coefficients μ_j and let $\boldsymbol{d_0}^{(b)} \leftarrow \boldsymbol{u_0} + \sum_{j=1}^{\tau} \mu_j \boldsymbol{c_\tau}^{(b)}$. We then choose $\boldsymbol{d_j}^{(b)}$ for $j \in [1, 3n]$ and $b \in \{0, 1\}$ arbitrarily.

To obtain the other $\boldsymbol{d_j}^{(b)}$ from $\boldsymbol{d_0}^{(b)}$ we note that we can obtain $\boldsymbol{d_j}^{(b)} + \overline{\boldsymbol{c}}_{\boldsymbol{j}}{}^{(b)}$ from $\boldsymbol{d_{j-1}}^{(b)}$. Thus, to be able to recursively compute the $\boldsymbol{d_j}^{(b)}$, we need to know the $\overline{\boldsymbol{c}}_{\boldsymbol{j}}{}^{(b)}$. The $\overline{\boldsymbol{c}}_{\boldsymbol{j}}{}^{(b)}$ are sampled as some linear combination of $\boldsymbol{c'_i}^{(b)}$, so let us fix this combination and make it the same for both $b = 1, 2$.

If at the beginning we are given just an element $\boldsymbol{r}$ of $\tilde{C}_n^{(b)}$ instead of an element of $\tilde{C}_n^{(1)} + \tilde{C}_n^{(2)}$, then we can obtain $\boldsymbol{c'_1}^{(b)}$ more simply, just by giving some public vector such that multiplying the RREF-basis by that vector gives us $\boldsymbol{c'_1}^{(b)}$. We note that this vector must be the same for $b = 1, 2$. We also note that it is easy to distinguish whether $\boldsymbol{r} \in \tilde{C}_n^{(b)}$ or $\boldsymbol{r} \in \tilde{C}_n^{(1)} + \tilde{C}_n^{(2)}$ by repeatedly left multiplying it by both $\overline{E}_{n,1}$ and $\overline{E}_3$ and to see whether we obtain a set of $\tau + n$ of independent vectors or a set of $2\tau + 2n$ independent vectors.

We can thus obtain the canonical basis for the $\tilde{C}_n^{(b)}$ and from it all the $\boldsymbol{d_j}^{(b)}$ as described above.

However, how do we find an element of $\boldsymbol{c'_1}^{(1)}$ if we are given $\boldsymbol{c'_1}^{(2)}$? The current construction does not allow for that. For that, let us construct an extra square matrix $\overline{H}$ where two of the eigenvectors are $\boldsymbol{c'_1}^{(1)}$ and $\boldsymbol{c'_1}^{(2)}$ with corresponding eigenvalues $\overline{a_1}$ and $\overline{a_2}$, and the other eigenpairs are $(\overline{\boldsymbol{q_j}}, \overline{a_j})$ for $j \in [3, k]$. Now, we also give out $\overline{H}$ and $\overline{\boldsymbol{q_j}}$ for $j \in [3, k]$. Given an eigenvector, it is easy to find the corresponding eigenvalue by simple linear algebra. However, if now one additionally learns one of the $\boldsymbol{c'_1}^{(1)}$ and $\boldsymbol{c'_1}^{(2)}$, then one also learns the corresponding eigenvalue of that eigenvector and thus one then knows $k-1$ of the k eigenvalues. However, the trace of the matrix is equal to the sum of the eigenvalues, and hence it is trivial to compute the last missing eigenvalue, and hence the missing one from $\boldsymbol{c'_1}^{(1)}$ and $\boldsymbol{c'_1}^{(2)}$. From there on, one can learn all the other missing information.

If we are given some $\boldsymbol{d_i}^{(b)}$ for some $i \in [0, 3n]$, then it is easy to see that multiplying it by either B_0, $B_0 E_j$ or $\overline{E}_{j,2}$, depending on what $i \pmod 3$ is, gives us an element of $\tilde{C}_j^{(b)}$. Thus we can proceed as in the previous point. If we are given some linear combination of $\boldsymbol{d_i}^{(1)}$ and $\boldsymbol{d_i}^{(2)}$, we likewise obtain a general element of $\tilde{C}_j^{(1)} + \tilde{C}_j^{(2)}$ and proceed as above.

We note that this construction means that we have to somewhat change the order in which we choose our variables. Thus we first choose the $\boldsymbol{c_i}^{(b)}$ for $i \in [1, \tau+n]$. We thus now have fixed our submodules $\tilde{C}_i^{(b)}$ for $i \in [0, n]$. Choose

$\overline{E}_{n,1}$ and $\overline{E}_{n,3}$ in such a way that $\overline{E}_{n,c}\tilde{C}_n^{(b)} = \tilde{C}_n^{(b)}$ for $c \in \{1,3\}$. Choose E_1' in such a way that $E_1'\tilde{C}_0^{(b)} = \tilde{C}_0^{(b)}$. Choose B_2 in such a way that $B_2\tilde{C}_{i-1}^{(b)} \subseteq \tilde{C}_i^{(b)}$ for all $i \in [1,n]$.

We can now write down the RREF of $\tilde{C}_n^{(1)}$ and the RREF of $\tilde{C}_n^{(2)}$, which we denote by $R^{(1)}$ and $R^{(2)}$, respectively. Fix now a vector $\boldsymbol{v}$. We now let $\boldsymbol{c_1'}^{(b)} \leftarrow \boldsymbol{v}R^{(b)}$ for $b \in \{0,1\}$.

Now, we can write down the RREF of $\tilde{C}_n^{(1)} + \tilde{C}_n^{(2)}$, let us denote that by R. Sample now at random ξ_1 and ξ_2. We now set $h_1 \leftarrow \frac{\xi_2 \boldsymbol{c_1'}^{(1)} - \xi_1 \boldsymbol{c_1'}^{(2)}}{\xi_2 - \xi_1}$ and $h_2 \leftarrow \frac{\boldsymbol{c_1'}^{(1)} - \boldsymbol{c_1'}^{(2)}}{\xi_2 - \xi_1}$. (Note that this is equivalent to $\boldsymbol{c_1'}^{(1)} = h_1 + \xi_1 h_2$ and $\boldsymbol{c_1'}^{(2)} = h_1 + \xi_2 h_2$).

We want that $Uh_1 = h_1 + \sigma_{2,1}h_2$ and $Uh_2 = h_1 + \sigma_{2,2}h_2$.

Choose ζ_1 randomly. Then denote $\overline{\lambda} \leftarrow \zeta_1\xi_1$. Let $\zeta_2 \leftarrow \frac{\overline{\lambda}}{\xi_2}$. We thus have that $\overline{\lambda} = \zeta_1\xi_1 = \zeta_2\xi_2$. Now construct the matrix U in such a way that it has the eigenpairs $(\boldsymbol{c_1'}^{(1)}, \zeta_1)$ and $(\boldsymbol{c_1'}^{(2)}, \zeta_2)$.

One can check that $Uh_1 = h_1 + (\frac{1}{\xi_1\xi_2})h_2$ and $Uh_2 = h_1 + (1 - \frac{1}{\xi_1} - \frac{1}{\xi_2})h_2$.

Now sample a matrix V such that $R \cdot V = \begin{pmatrix} U & h_1 & h_2 \end{pmatrix}$

Now compute $\boldsymbol{c_i'}^{(b)} \leftarrow E_1'\boldsymbol{c_{i-1}'}^{(b)}$ for $i \in [2, \tau-1]$ and $\boldsymbol{c_i'}^{(b)} \leftarrow B_1\boldsymbol{c_{i-1}'}^{(b)}$ for $i \in [\tau, \tau+n]$.

To tie $\boldsymbol{c_1'}^{(1)}$ and $\boldsymbol{c_1'}^{(2)}$ together, we sample vectors $\overline{\boldsymbol{q_i}}$ for $i \in [3,n]$ and eigenvalues $\overline{a_i}$ for $i \in [1,n]$. We compute $\overline{H}$ to be the matrix that has the eigenpairs $(\boldsymbol{c_1'}^{(1)}, \overline{a_1}), (\boldsymbol{c_1'}^{(2)}, \overline{a_2}), (\overline{\boldsymbol{q_3}}, \overline{a_3}), \ldots, (\overline{\boldsymbol{q_n}}, \overline{a_n})$.

We now need to sample the $\overline{\boldsymbol{c}}_{\boldsymbol{j}}^{(b)}$. For that , we first, for $i \in [0,n]$, sample $\{\xi'_{3i-1,1,j}\}_{j=1,c=1}^{\tau+i,4}$, $\{\xi'_{3i,j}\}_{j=1}^{\tau+i}$,$\{\xi'_{3i+1,j}\}_{j=1}^{\tau+i} \xleftarrow{\$} \mathbb{Z}_N$. We note that these are the same for $b \in \{1,2\}$.

Now we can compute $\overline{\boldsymbol{c}}_{\boldsymbol{3i-1,c}}^{(b)} \leftarrow \sum_{j=1}^{\tau+i} \xi'_{3i-1,c,j}\boldsymbol{c_j'}^{(b)}$ for $b \in \{1,2\}$ and $c \in [1,4]$, $\overline{\boldsymbol{c}}_{\boldsymbol{3i}}^{(b)} \leftarrow \sum_{j=1}^{\tau+i} \xi'_{3i,j}\boldsymbol{c_j'}^{(b)}$ for $b \in \{1,2\}$, and $\overline{\boldsymbol{c}}_{\boldsymbol{3i+1}}^{(b)} \leftarrow \sum_{j=1}^{\tau+i} \xi'_{3i+1,j}\boldsymbol{c_j'}^{(b)}$ for $b \in \{1,2\}$ Now we can finally get to the $\boldsymbol{d_j}$. Choose at random $\boldsymbol{u_0}$ and some coefficients μ_j and let $\boldsymbol{d_0}^{(b)} \leftarrow \boldsymbol{u_0} + \sum_{j=1}^{\tau} \mu_j \boldsymbol{c_\tau}^{(b)}$. Then sample at random $\boldsymbol{d_j}^{(b)}$ for $j \in [1,3n]$.

Now sample B_0 in such a way that $B_0\tilde{C}_{i-1}^{(b)} \in \tilde{C}_i^{(b)}$ and $B_0\boldsymbol{d_{3(i-1)+1}}^{(b)} = \overline{\boldsymbol{c}}_{\boldsymbol{3(i-1)+2,0}}^{(b)}$ for $b \in \{1,2\}$ and $i \in [1,n]$.

Sample B_1 such that $B_1\tilde{C}_{i-1}^{(b)} \in \tilde{C}_i^{(b)}$ and $B_1\boldsymbol{d_{3(i-1)+1}}^{(b)} = \boldsymbol{d_{3(i-1)+2}}^{(b)} + \overline{\boldsymbol{c}}_{\boldsymbol{3(i-1)+2,1}}^{(b)}$ for $b \in \{1,2\}$ and $i \in [1,n]$.

Sample E_i such that $E_i\tilde{C}_{i-1}^{(b)} \in \tilde{C}_{i-1}^{(b)}$ and $E_i\boldsymbol{d_{3(i-1)}}^{(b)} = \boldsymbol{d_{3(i-1)+1}}^{(b)} + \overline{\boldsymbol{c}}_{\boldsymbol{3(i-1)+1}}^{(b)}$ for $i \in [1,n]$.

For $i \in [1,n]$, sample $\overline{E}_{i,c}$ such that $\overline{E}_{i,c}\tilde{C}_i^{(b)} \in \tilde{C}_i^{(b)}$ for $c \in \{1,3\}$.

For $i \in [1,n]$, sample $\overline{E}_{i,2}$ such that $\overline{E}_{i,2}\tilde{C}_i^{(b)} \in \tilde{C}_i^{(b)}$ and $\overline{E}_{i,2}\boldsymbol{d_{3(i-1)+2}}^{(b)} = \boldsymbol{d_{3i}}^{(b)} + \overline{\boldsymbol{c}}_{\boldsymbol{3i}}^{(b)}$. Sample $\overline{E}_{i,4}$ such that $\overline{E}_{n,4}\tilde{C}_n^{(b)} \in \tilde{C}_n^{(b)}$ and $\overline{E}_{n,4}\boldsymbol{d_{3(n-1)+2}}^{(b)} = \boldsymbol{d_{3n}}^{(b)} + \overline{\boldsymbol{c}}_{\boldsymbol{3n,4}}^{(b)}$ for $i \in [1,n]$.

Concerning the orthogonal modules to $\tilde{C}_i^{(1)} + \tilde{C}_i^{(2)}$, the largest ortogonal module is the one corresponding to $\tilde{C}_1^{(1)} + \tilde{C}_1^{(2)}$. We note that if we know some $\tilde{C}_i^{(1)} + \tilde{C}_i^{(2)}$, then we also trivially know $(\tilde{C}_i^{(1)} + \tilde{C}_i^{(2)})^\perp$ and vice versa.

Thus, if we have computed the $\tilde{C}_1^{(1)}$ and $\tilde{C}_2^{(2)}$, it is easy to compute all the relevant orthogonal modules.

What about the other way around? Suppose that we have some element $\boldsymbol{r}$ of $(\tilde{C}_1^{(1)} + \tilde{C}_1^{(2)})^\perp$? As above, we can construct some two random matrices O_0 and O_1 such that $O_0(\tilde{C}_1^{(1)} + \tilde{C}_1^{(2)})^\perp \in (\tilde{C}_1^{(1)} + \tilde{C}_1^{(2)})^\perp$ for at least one of them, $(\boldsymbol{r}, O_b\boldsymbol{r}, O_bO_b\boldsymbol{r} \dots O_b^{k-2\tau}\boldsymbol{r})$ spans $(\tilde{C}_1^{(1)} + \tilde{C}_1^{(2)})^\perp$. (More specifically, we describe it as $O_b\ddot{\boldsymbol{o}}_i = \sum_{j=1}^{k-2n-2\tau} \omega_{i,j,b}\ddot{\boldsymbol{o}}_j$ for $i \in [1, k-2n-2\tau]$.) From that we can obtain $\tilde{C}_1^{(1)} + \tilde{C}_1^{(2)}$ and proceed as previously.

What if we have an element of, say $(\tilde{C}_1^{(1)})^\perp$ which does not belong to $(\tilde{C}_1^{(2)})^\perp$?

For that, let us use the same technique as we used above for letting two vectors be the two only unknown eigenvectors in some known matrix.

More specifically, choose some $\boldsymbol{o}^{(1)} \in (\tilde{C}_1^{(1)})^\perp$ and $\boldsymbol{o}^{(2)} \in (\tilde{C}_1^{(2)})^\perp$. At random, sample vectors $\{\overline{\boldsymbol{o_j}}\}_{j=3}^k$ and ring elements $\{\ddot{a}_j\}_{j=1}^k$. Then let us construct a matrix O' with eigenpairs $(\boldsymbol{o}^{(1)}, \ddot{a}_1), (\boldsymbol{o}^{(2)}, \ddot{a}_2), (\overline{\boldsymbol{o_3}}, \ddot{a}_3), \dots, (\overline{\boldsymbol{o_k}}, \ddot{a}_k)$ and let us make O' and $\{\overline{\boldsymbol{o_j}}\}_{j=3}^k$ and ring elements $\{\ddot{a}_j\}_{j=3}^k$ public.

This allows us to construct all the unknown elements from one element of either $(\tilde{C}_1^{(1)})^\perp$ or $(\tilde{C}_1^{(2)})^\perp$.

Denote the basis vectors of $\mathsf{RREF}((\tilde{C}_1^{(1)} + \tilde{C}_1^{(2)})^\perp)$ by $\{\ddot{\boldsymbol{o}}_i\}_{i=1}^{k-2n-2\tau}$.

C.3 General solution

We give the analogue for the *SampleMatrices* algorithm in Algorithm 8.

Now we give the algorithm for constructing the DPF key from the matrices.

We now give the evaluation algorithm, which is very similar to the Evaluation algorithm in the main body of the paper.

Theorem 5. *Suppose that factoring $N = pq$ is hard. Then, if an adversary is given the values $\{A_{i,0}\}_{i=1}^n, \{A_{i,1}\}_{i=1}^n$ as from Algorithm 9, then the adversary can not compute any member of the submodules $\tilde{C}_i^{(1)} + \tilde{C}_i^{(2)}$ nor any of the $\boldsymbol{d_{3(i-1)}}^{(b)} + \tilde{C}_{i-1}^{(b)}$ or $\boldsymbol{d_{3(i-1)+1}}^{(b)} + \tilde{C}_{i-1}^{(b)}$ for $i \in [1, n]$ nor any element of $(\tilde{C}_i^{(1)})^\perp$, $(\tilde{C}_i^{(2)})^\perp$ nor $(\tilde{C}_i^{(1)} + \tilde{C}_i^{(2)})^\perp$.*

Proof. We prove this in the order that we describe a system of randomly solvable equations concerning our system. We then show that learning any part of the solution gives us the full solution. Knowing that, we show that there is only a polynomial number of solutions to some parts of the solution, which then implies that there is only a polynomial number of solutions total. We then describe a simple change in variables that leaves our system invariant. Hence we can use Theorem 4 and obtain that it is infeasible to obtain a solution to the system.

We then argue that since knowing any of the values given in the statement of the theorem would allow us to find a solution, and computing a solution implies

factoring N, then it is infeasible to find any of the values given in the statement of the theorem.

To apply Theorem 4, let us specify what are the unknown variables in that theorem. For visual clarity, we denote the unknown variables with *red*.

In addition to the publicly known $A_{i,0}, A_{i,1}$, we will also fix B_0, B_1, $\overline{E}_i$ for $i \in [1,n]$, $E_{i,c}$ for $i \in [1,n]$ and $c \in [1,4]$, the vector $\boldsymbol{u_0}$, the α and β. We will also fix $\boldsymbol{u_0}$, $\{\mu_j\}_{j=1}^{\tau}$, $\boldsymbol{v}$, $\overline{\lambda}$, V, B_2, E'_1,$\{\xi'_{3i-1,1,j}\}_{j=1,c=1}^{\tau+i,4}$, $\{\xi'_{3i,j}\}_{j=1}^{\tau+i}$, $\{\xi'_{3i+1,j}\}_{j=1}^{\tau+i}$, $\overline{H}$, $\{\overline{\boldsymbol{q_i}}\}_{i=3}^{n}$, $\{\overline{a_i}\}_{i=3}^{n}$, O', $\{\overline{\boldsymbol{o_j}}\}_{j=3}^{k}$, $\{\ddot{a}_j\}_{j=3}^{k}$, O_0,O_1.

The equations that hold are the following:

1. $A_{i,\alpha_i} = F_i\overline{E}_iB_0E_{i,2}F_{i-1}^{-1} + F_i\overline{E}_iB_1E_{i,4}F_{i-1}^{-1}$ for $i \in [1,n]$
2. $A_{i,\overline{\alpha_i}} = F_i\overline{E}_iB_0E_{i,1}F_{i-1}^{-1} + F_i\overline{E}_iB_1E_{i,3}F_{i-1}^{-1}$ for $i \in [1,n]$
3. $B_0\boldsymbol{d_{3(i-1)+1}}^{(b)} = \overline{\boldsymbol{c}}_{\boldsymbol{3(i-1)+2,0}}^{(b)}$ for $i \in [1,n]$ and $b \in \{1,2\}$.
4. $B_1\boldsymbol{d_{3(i-1)+1}}^{(b)} = \boldsymbol{d_{3(i-1)+2}}^{(b)} + \overline{\boldsymbol{c}}_{\boldsymbol{3(i-1)+2,1}}^{(b)}$ for $i \in [1,n]$ and $b \in \{1,2\}$.
5. $E_i\boldsymbol{d_{3(i-1)}}^{(b)} = \boldsymbol{d_{3(i-1)+1}}^{(b)} + \overline{\boldsymbol{c}}_{\boldsymbol{3(i-1)+1}}^{(b)}$ for $i \in [1,n]$ and $b \in \{1,2\}$.
6. $\overline{E}_{i,4}\boldsymbol{d_{3(i-1)+2}}^{(b)} = \boldsymbol{d_{3i}}^{(b)} + \overline{\boldsymbol{c}}_{\boldsymbol{3i,4}}^{(b)}$ for $i \in [1,n]$ and $b \in \{1,2\}$.
7. $E_i\boldsymbol{c'_{i'}}^{(b)} = \sum_{j=1}^{\tau+i-1} \lambda'_{i,i',j}{}^{(b)}\boldsymbol{c'_j}^{(b)}$ for $i \in [1,n]$, $i' \in [1,\tau+i]$, $b \in \{1,2\}$.
8. $\overline{E}_{i,c}\boldsymbol{c'_{i'}}^{(b)} = \sum_{j=1}^{\tau+i} \lambda''_{i,i',j,c}{}^{(b)}\boldsymbol{c'_j}^{(b)}$ for $i \in [1,n]$, $i' \in [1,\tau+i+1]$, $b \in \{1,2\}$.
9. $E'_1\boldsymbol{c'_{i'}}^{(b)} = \sum_{j=1}^{\tau+i} \lambda'''_{i',j}{}^{(b)}\boldsymbol{c'_j}^{(b)}$ for $i \in [1,n]$, $i' \in [1,\tau]$, $b \in \{1,2\}$.
10. $B_0\boldsymbol{c'_i}^{(b)} = \sum_{j=1}^{\tau+1} \lambda_{i,j,0}{}^{(b)}\boldsymbol{c'_j}^{(b)}$ for $i \in [1,\tau]$, $b \in \{1,2\}$.
11. $B_1\boldsymbol{c'_i}^{(b)} = \sum_{j=1}^{\tau+1} \lambda_{i,j,1}{}^{(b)}\boldsymbol{c'_j}^{(b)}$ for $i \in [1,\tau]$, $b \in \{1,2\}$.
12. $B_2\boldsymbol{c'_i}^{(b)} = \sum_{j=1}^{\tau+1} \lambda_{i,j,2}{}^{(b)}\boldsymbol{c'_j}^{(b)}$ for $i \in [1,\tau]$, $b \in \{1,2\}$.
13. $B_0\boldsymbol{c'_i}^{(b)} = \sum_{j=1}^{\tau+i+1} \lambda_{i,j,0}{}^{(b)}\boldsymbol{c'_j}^{(b)}$ for $i \in [\tau+1,\tau+n]$, $b \in \{1,2\}$.
14. $B_1\boldsymbol{c'_i}^{(b)} = \sum_{j=1}^{\tau+i+1} \lambda_{i,j,1}{}^{(b)}\boldsymbol{c'_j}^{(b)}$ for $i \in [\tau+1,\tau+n]$, $b \in \{1,2\}$.
15. $B_2\boldsymbol{c'_i}^{(b)} = \sum_{j=1}^{\tau+i+1} \lambda_{i,j,2}{}^{(b)}\boldsymbol{c'_j}^{(b)}$ for $i \in [\tau+1,\tau+n]$, $b \in \{1,2\}$.
16. $\boldsymbol{d_0}^{(b)} = \boldsymbol{u_0} + \sum_{j=1}^{\tau} \mu_j\boldsymbol{c_\tau}^{(b)}$ for $b \in \{1,2\}$.
17. $R^{(b)}$ is in RREF form for $b \in \{1,2\}$.
18. $R = \mathsf{RREF}\begin{pmatrix} R^{(1)} \\ R^{(2)} \end{pmatrix}$
19. $\boldsymbol{c'_1}^{(b)} = \boldsymbol{v}R^{(b)}$ for $b \in \{0,1\}$.
20. $\boldsymbol{c'_1}^{(b)} = h_1 + \xi_b h_2$ for $b \in \{1,2\}$.
21. $\overline{H}\boldsymbol{c'_1}^{(b)} = \overline{a_b}\boldsymbol{c'_1}^{(b)}$ for $b \in \{1,2\}$
22. $\overline{H}\overline{\boldsymbol{q_i}} = \overline{a_i\boldsymbol{q_i}}$ for $i \in [3,n]$
23. $\overline{\lambda} = \zeta_b\xi_b$ for $b \in \{1,2\}$.
24. $U\boldsymbol{c'_1}^{(b)} = \zeta_b\boldsymbol{c'_1}^{(b)}$.
25. $Uh_1 = h_1 + \sigma_{1,2}h_2$.
26. $Uh_2 = h_1 + \sigma_{2,2}h_2$.
27. $R \cdot V = \begin{pmatrix} U & h_1 & h_2 \end{pmatrix}$
28. $\boldsymbol{c'_i}^{(b)} = E'_1\boldsymbol{c'_{i-1}}^{(b)}$ for $i \in [2,\tau-1]$ for $b \in \{1,2\}$.
29. $\boldsymbol{c'_i}^{(b)} \leftarrow B_1\boldsymbol{c'_{i-1}}^{(b)}$ for $i \in [\tau,\tau+n]$ for $b \in \{1,2\}$.

30. $\overline{\boldsymbol{c}}_{\boldsymbol{3(i-1)+1}}{}^{(b)} = \sum_{j=1}^{\tau+i} \xi'_{3(i-1)+1,j} \boldsymbol{c'_j}{}^{(b)}$ for $b \in \{1,2\}$
31. $\overline{\boldsymbol{c}}_{\boldsymbol{3(i-1)+2,c}}{}^{(b)} = \sum_{j=1}^{\tau+i} \xi'_{3(i-1)+2,c,j} \boldsymbol{c'_j}{}^{(b)}$ for $b \in \{1,2\}$ for $c \in \{0,1\}$
32. $\overline{\boldsymbol{c}}_{\boldsymbol{3(i-1)+2,c}}{}^{(b)} = \sum_{j=1}^{\tau+i} \xi'_{3(i-1)+2,c,j} \boldsymbol{c'_j}{}^{(b)}$ for $b \in \{1,2\}$,for $c \in \{2,4\}$
33. $O' \boldsymbol{o}^{(b)} = \ddot{a}_b \boldsymbol{o}^{(b)}$ for $b \in \{0,1\}$
34. $O' \overline{\boldsymbol{o_j}} = \ddot{a}_j \overline{\boldsymbol{o_j}}$ for $j \in [3,n]$
35. $\{\ddot{\boldsymbol{o}}_i, \}_{i=1}^{k-2n-2\tau}$ is in RREF form.
36. $\langle \ddot{\boldsymbol{o}}_i, \boldsymbol{c'_j}{}^{(b)} \rangle = 0$ for $b \in \{0,1\}$ for $i \in [1, k-2n-2\tau]$, for $j \in [1,\tau]$.
37. $O_0 \ddot{\boldsymbol{o}}_i = \sum_{j=1}^{k-2n-2\tau} \omega_{i,j,0} \ddot{\boldsymbol{o}}_j$ for $i \in [1, k-2n-2\tau]$.
38. $O_1 \ddot{\boldsymbol{o}}_i = \sum_{j=1}^{k-2n-2\tau} \omega_{i,j,1} \ddot{\boldsymbol{o}}_j$ for $i \in [1, k-2n-2\tau]$.

We will now show how, given one part of a solution, given in the statement of the theorem, allows us to compute the whole solution.

We gave the rough construction for this kind of construction in the beginning of this section. Let us now repeat this in slightly more detail.

There are three cases for us to consider.

Let us start with the case when the adversary has produced some element $\boldsymbol{c''_0} \in \tilde{C}_n^{(1)} + \tilde{C}_n^{(2)}$ with $\boldsymbol{r} \notin \tilde{C}_n^{(1)}, \tilde{C}_n^{(2)}$. We have that by recursively computing either by $\boldsymbol{c''_i} \leftarrow \overline{E}_{n,1} \boldsymbol{c''_0}$ or $\boldsymbol{c''_i} \leftarrow \overline{E}_{n,3} \boldsymbol{c''_0}$ and obtain $2n + 2\tau$ basis vectors for $\tilde{C}_n^{(1)} + \tilde{C}_n^{(2)}$.

Using that we can compute the R by finding the RREF form of $\tilde{C}_n^{(1)} + \tilde{C}_n^{(2)}$. From this we can obtain $\left(U\ h_1\ h_2\right) \leftarrow R \cdot V$.

Then we can compute $\boldsymbol{c'_1}{}^{(b)}$ in the following way.

Given h_1, h_2 and U and since we know that $Uh_1 = h_1 + \sigma_{1,2} h_2$ and $Uh_2 = h_1 + \sigma_{2,2} h_2$, one can easily compute $\sigma_{1,2}, \sigma_{2,2} \in \mathbb{Z}_N$.

We know that $\boldsymbol{c'_1}{}^{(1)} = h_1 + \xi_1 h_2$ and $\boldsymbol{c'_1}{}^{(2)} = h_1 + \xi_2 h_2$.

We thus have that

$$\zeta_1 h_1 + \zeta_1 \xi_1 h_2 = \zeta_1 \boldsymbol{c'_1}{}^{(1)} = U \boldsymbol{c'_1}{}^{(1)} = U(h_1 + \xi_1 h_2) = h_1(\sigma_{1,1} + \xi_1 \sigma_{2,1}) + h_2(\sigma_{1,2} + \xi_1 \sigma_{2,2}),$$

Since h_1 and h_2 are linearly independent, it follows that $\zeta_1 = \sigma_{1,1} + \xi_1 \sigma_{2,1}$ and $\zeta_1 \xi_1 = \sigma_{1,2} + \xi_1 \sigma_{2,2}$

Analogously

$$\zeta_2 h_1 + \zeta_2 \xi_1 h_2 = h_1(\sigma_{1,1} + \xi_2 \sigma_{2,1}) + h_2(\sigma_{1,2} + \xi_2 \sigma_{2,2}),$$

and analogously it follows that $\zeta_2 = \sigma_{1,1} + \xi_2 \sigma_{2,1}$ and $\zeta_2 \xi_2 = \sigma_{1,2} + \xi_2 \sigma_{2,2}$.

Using that $\overline{\lambda} = \zeta_1 \xi_1 = \zeta_2 \xi_2$, we can thus solve the linear system of equation for ζ_b and ξ_1 and via that compute the $\boldsymbol{c'_1}{}^{(b)}$.

Now we compute $\boldsymbol{c'_i}{}^{(b)} \leftarrow E'_1 \boldsymbol{c'_{i-1}}{}^{(b)}$ for $i \in [2, \tau-1]$ and $\boldsymbol{c'_i}{}^{(b)} \leftarrow B_1 \boldsymbol{c'_{i-1}}{}^{(b)}$ for $i \in [\tau, \tau+n]$ for $b \in \{1,2\}$.

We can now compute $\boldsymbol{d_0}{}^{(b)} \leftarrow \boldsymbol{u_0} + \sum_{j=1}^{\tau} \mu_j \boldsymbol{c_\tau}{}^{(b)}$ for $b \in \{1,2\}$.

We now compute the $\overline{\boldsymbol{c}}_{\boldsymbol{j}}{}^{(b)}$ by $\overline{\boldsymbol{c}}_{\boldsymbol{3i-1,c}}{}^{(b)} \leftarrow \sum_{j=1}^{\tau+i} \xi'_{3i-1,c,j} \boldsymbol{c'_j}{}^{(b)}$ for $c \in [1,4]$, $\overline{\boldsymbol{c}}_{\boldsymbol{3i}}{}^{(b)} \leftarrow \sum_{j=1}^{\tau+i} \xi'_{3i,j} \boldsymbol{c'_j}{}^{(b)}$ for $b \in \{1,2\}$, and $\overline{\boldsymbol{c}}_{\boldsymbol{3i+1,c}}{}^{(b)} \leftarrow \sum_{j=1}^{\tau+i} \xi'_{3i+1,j,c} \boldsymbol{c'_j}{}^{(b)}$ for $b \in \{1,2\}$, for $c \in \{0,1\}$.

Now we can iteratively compute $\boldsymbol{d_{3(i-1)+1}}^{(b)} = E_i \boldsymbol{d_{3(i-1)}}^{(b)} - \overline{\boldsymbol{c}}_{\boldsymbol{3(i-1)+1}}{}^{(b)}$, $\boldsymbol{d_{3(i-1)+2}}^{(b)} = B_1 \boldsymbol{d_{3(i-1)+1}}^{(b)} - \overline{\boldsymbol{c}}_{\boldsymbol{3(i-1)+2,1}}{}^{(b)}$ and $\boldsymbol{d_{3i}} = \overline{E}_{i,4} \boldsymbol{d_{3(i-1)+2}}^{(b)} - \overline{\boldsymbol{c}}_{\boldsymbol{3i,4}}{}^{(b)}$. We have thus obtained all the variables.

Now consider the alternate scenario where the adversary obtains $\boldsymbol{r} \in \tilde{C}_n^{(b)}$. We then compute in a similar fashion as in the previous case a basis for $\tilde{C}_n^{(b)}$, and then compute the RREF of it, obtaining $R^{(b)}$. From this we can compute $\boldsymbol{c'_1}^{(b)} \leftarrow \boldsymbol{v}^T R^{(b)}$.

Now, we can compute for the matrix $\overline{H}$ the eigenvalue $\overline{a_b}$ corresponding to $\boldsymbol{c'_1}^{(b)}$. Since we know the trace of $\overline{H}$, and the other eigenvalues $\overline{a_j}$ for $j \in [3, n]$ and since the sum of all eigenvalues is the trace of $\overline{H}$, we can compute $\overline{a_{3-b}}$, and from that we can compute $\boldsymbol{c'_1}^{(3-b)}$. Hence we then know both $\boldsymbol{c'_1}^{(1)}$ and $\boldsymbol{c'_1}^{(2)}$. From these we can compute all the other variables, as in the previous case.

Now consider the case where the adversary knows some $\boldsymbol{d_j}^{(b)}$ for $j \in [0, 3n-1]$. For each of those vectors there exists a known matrix that maps that vector to some element of $\tilde{C}_i$ and from there on we can proceed as above.

Thus we conclude that it is infeasible for an adversary to learn any of the hidden variables noted above. We note that this holds in the setting where the adversary knows more information than she knows in the real setting, as in the real setting, the adversary does not know the Q'_i,Q''_i, their respective eigenvalues, the E_i, the values of α and β. Thus we consider it infeasible that an adversary would be able to learn any of the respective eigenvectors or eigenvalues.

Considering the number of solutions, we note that the matrix $\overline{H}$ has at most n^2 eigenvectors, and thus there are at most a polynomial number of choices for $\boldsymbol{c'_1}^{(1)}$ and $\boldsymbol{c'_1}^{(2)}$, and when those are fixed, all the other parts of the solution become fixed as well, and thus we have only a polynomial number of solutions.

Concerning the orthogonal module $(\tilde{C}_i^{(1)} + \tilde{C}_i^{(2)})^\perp$, we note that one can learn $(\tilde{C}_i^{(1)} + \tilde{C}_i^{(2)})$ from $(\tilde{C}_i^{(1)} + \tilde{C}_i^{(2)})^\perp$ and vice versa.

If one knows one element of either $(\tilde{C}_i^{(1)} + \tilde{C}_i^{(2)})^\perp$, $(\tilde{C}_i^{(1)})^\perp$, $(\tilde{C}_i^{(2)})^\perp$, one can construct from these the whole $(\tilde{C}_i^{(1)} + \tilde{C}_i^{(2)})^\perp$ by the techniques discussed above.

The last component for applying Theorem 4 is a rational transformation. We will use the following rational transformation: that for all $j \in [0, 3n]$ we exchange $\boldsymbol{d_j}^{(1)} \rightleftarrows \boldsymbol{d_j}^{(2)}$; for all $i \in [1, n]$, $c \in \{2, 4\}$ we exchange $\overline{\boldsymbol{c}}_{\boldsymbol{3i,c}}{}^{(1)} \rightleftarrows \overline{\boldsymbol{c}}_{\boldsymbol{3i,c}}{}^{(2)}$; for all $i \in [1, n]$ we exchange $\overline{\boldsymbol{c}}_{\boldsymbol{3i+1}}{}^{(1)} \rightleftarrows \overline{\boldsymbol{c}}_{\boldsymbol{3i+1}}{}^{(2)}$; for all $i \in [1, n]$ for all $c \in \{0, 1\}$ we exchange $\overline{\boldsymbol{c}}_{\boldsymbol{3i+2,c}}{}^{(1)} \rightleftarrows \overline{\boldsymbol{c}}_{\boldsymbol{3i+2,c}}{}^{(2)}$; $R^{(1)} \rightleftarrows R^{(2)}$; for all $i \in [1, \tau + n]$ we exchange $\boldsymbol{c'_j}^{(1)} \rightleftarrows \boldsymbol{c'_j}^{(2)}$; $\overline{a_1} \rightleftarrows \overline{a_2}$; $\zeta_1 \rightleftarrows \zeta_2$; $\xi_1 \rightleftarrows \xi_2$; and the following variables are unchanged under the transformation: R, U, h_1, h_2.

Thus we can apply the Theorem 4, and thus we have that the theorem holds. □

D How to break the scheme if e is not in the exponent

To get the correct output, we define a vector $\boldsymbol{e}$ such that $\left\langle \boldsymbol{e}, F_n \tilde{C}_n \right\rangle = 0$ and $\langle \boldsymbol{e}, \boldsymbol{z}_n \rangle = \beta$ where $\boldsymbol{z}_n$ is the secret-shared state at the node α. This ensures that

all the inactive leaves map to zero in the end. However, this introduces a security issue. Namely, for any $\alpha^* \in \{0,1\}^n$, define $\overline{A_{\alpha^*}} := A_{n,\alpha_n^*} \cdot A_{n-1,\alpha_{n-1}^*} \dots A_{1,\alpha_1^*}$, where α_i^* is the ith bit of α^*. If $\alpha_* \neq \alpha$, then by our construction $\overline{A_{\alpha^*}}\boldsymbol{z}_0 \in F_n\tilde{C}_n$ and hence $\boldsymbol{e}^\perp\overline{A_{\alpha^*}}\boldsymbol{z}_0 = 0$. For the active leaf α, we have that $\boldsymbol{e}^\perp\overline{A_{\alpha^*}}\boldsymbol{z}_0 = \beta$.

Hence, if $\boldsymbol{e}$ was publicly available, we could, for every $\alpha^* \in \{0,1\}^n$, compute the vector $\boldsymbol{e}^\perp\overline{A_{\alpha^*}}$. We will now describe a test, that, for given k leaves, allows us to test whether the active leaf is among them by using just the publicly available information.

Namely, suppose that we have a set of k leaves $\alpha_1, \dots, \alpha_k$, where $\alpha \neq \alpha_j$, $j \in [1,k]$. We now compute the square matrix with row vectors $\boldsymbol{e}^\perp\overline{A_{\alpha_i}}$. If we multiply this matrix by $\boldsymbol{z}_0$, then the result would be 0, since $\boldsymbol{e}^\perp\overline{A_{\alpha_j}}\boldsymbol{z}_0 = 0$, for $j \in [1,k]$. Thus, since there is a nonzero vector in the kernel of this matrix, it would not be full-rank. However, if α was one of the α_j, then the matrix would not map $\boldsymbol{z}_0$ to a zero vector and thus the matrix would with an overwhelming probability be full-rank. If $\boldsymbol{e}$ is given in $\mathbb{Z}_N$, then it is easy to detect whether a given matrix is full-rank or not. This means that if $\boldsymbol{e}$ was given in $\mathbb{Z}_N$, we could find out which point is α. However, if it is given in the exponent, in $\mathbb{G}$, then one would have to decide whether a matrix given in the exponent is full-rank, which is typical example of a DDH-style hard problem, and per our Submodule Discrete-Log Vector Heuristic, it is hard to compute.

E On the difficulty of computing invariant submodules

We will now discuss the hardness of computing shared invariant submodules.

For start consider the case is when we have that for some two τ-dimensional submodules $\tilde{C}$ and $\tilde{D}$ and some two $k \times k$ matrices E_1 and E_2 it holds that $E_1\tilde{C} = \tilde{D}$ and $E_2\tilde{C} = \tilde{D}$. In the special case that $\tau = 1$, this means that for some nonzero $\boldsymbol{x}$ and $\boldsymbol{y}$ and $\alpha \in R$ it holds that $E_1\boldsymbol{x} = \boldsymbol{y}$ and $E_2\boldsymbol{x} = \alpha\boldsymbol{y}$. We can restate this as $(E_1 - \alpha E_2)\boldsymbol{x} = 0$, and thus the matrix $(E_1 - \alpha E_2)$ has rank no more than $k-1$. This is an example of a MinRank problem with two matrices. In the Minrank problem, some matrices $E_1, \dots, E_t$ are given and one is required to find coefficients $\alpha_2, \dots, \alpha_t$ such that $E_1 + \sum_{j=2}^{t} \alpha_j E_t$ has a rank smaller than some target.

MinRank problems are generally solved in various ways, but the only technique that does not at least linearly with the ring size is Kipnis-Shamir modelling, where one computes a Gröbner basis for the terms of a multivariate equations that one can transform the problem into. If there are only two matrices, however, it seems that the problem is likely to have a polynomial solution. More specifically, in [39], it is estimated that the complexity of solving random instances of Kipnis-Shamir where the number of matrices is m, the matrix dimension is k and the desired rank is r, is given as

$$O(r\kappa)^{(d_{KS}+2)\omega}$$

where $2 < \omega \leq 3$ is the linear algebra constant,

$$d_{KS} = \min\{d | \binom{r}{d} k > \binom{r}{d+1} m, 1 \leq d \leq r-1\}$$

and κ is such that $\max\{\frac{m}{k-r}, d_{KS}+1\} \leq \kappa \leq k-r$.In our case, if $r = k-1$, we get that $\binom{r}{d}k > \binom{r}{d+1}m$ is equivalent to $\binom{k-1}{d}k > \binom{k-1}{d+1}m$. Now $\frac{\binom{k-1}{d}}{\binom{k-1}{d+1}} = \frac{d+1}{k-1-d}$, hence by dividing both sides by $\binom{k-1}{d+1}$ we obtain $\frac{d+1}{k-1-d}k > m$ or $\frac{k}{k-1-d}(d+1) > m$. Obviously $\frac{k}{k-1-d} > 1$. Thus d is no bigger than $m-1$. If k is significantly larger than m, then $\frac{k}{k-1-d} \approx 1$ and then we can simply take that $d = m-1$. We can take $\kappa = m$. We thus get that for m matrices the rough estimate $O(km)^{(m+1)\omega}$. If we have only $m = 2$, then this is just $O(2k)^{3\omega}$, which is polynomial.

However, if we increase τ, then we can obtain MinRank instances with more matrices.

More specifically, consider, as it is the case for us, that for some two matrices it holds that $E_0\tilde{C_{i-1}} = \tilde{C}_i$ and $E_0\tilde{C_{i-1}} = \tilde{C}_i$ for $i \in [1, n]$.

Then we know that for some $\boldsymbol{c} \in \tilde{C}_0$ and any bitstring $a = a_1 \dots a_t$ we can compute $E_a = E_{a_1}E_{a_2}\dots E_{a_t}$. Now, choose t such that $2^t > \dim \tilde{C}_t$. In that case the vectors $\{E_a\boldsymbol{c}\}_{a \text{ is a } t \text{ bit bitstring}}$ must all lie in $\tilde{C}_t$ and thus be linearly dependent. Thus we have that some $\sum_{a \text{ is a } t \text{ bit bitstring}} \alpha_a E_a \boldsymbol{c} = 0$. Hence we have that the matrix $\sum_{a \text{ is a } t \text{ bit bitstring}} \alpha_a E_a$ must not be full rank. Now, figuring out this matrix and the respective coefficients and $\boldsymbol{c}$ requires solving a MinRank instance with at least $\dim \tilde{C}_t + 1$ matrices. We can approximate the complexity of computing a Gröbner basis for such a matrix as $O((n \dim \tilde{C}_t)^{\dim \tilde{C}_t \omega}$.

We note again that even if a Gröbner basis is computed, over $\mathbb{Z}_N$, it is not enough still, as one must also then find a root to a high-degree polynomial, which is hard over $\mathbb{Z}_N$, as implied by Theorem 5 . The goal behind this Appendix is to give insight into future work about whether $\mathbb{Z}_N$ could be replaced by a finite field.

Algorithm 8: SampleMatricesFull

Input : N, k, n, τ, v, u

$\boldsymbol{c_i}^{(b)} \leftarrow (\mathbb{Z}_N)^k$ for $b \in \{1,2\}$ and $i \in [1, \tau+n]$;

Denote $\tilde{C}_i^{(b)} := \langle \boldsymbol{c_1}^{(b)}, \ldots, \boldsymbol{c_{i\tau+i}}^{(b)} \rangle$ for $i \in [1, \tau+n]$;

Sample E_1' such that $E_1'\tilde{C}_0^{(b)} \in \tilde{C}_0^{(b)}$;

Sample B_2 such that $B_2\tilde{C}_0^{(b)} \in \tilde{C}_1^{(b)}$;

Compute $R^{(b)} \leftarrow \mathsf{RREF}(\tilde{C}_n^{(b)})$ for $b \in \{1,2\}$;

$\boldsymbol{v} \xleftarrow{\$} (\mathbb{Z}_N)^{\tau+n}$;

$\boldsymbol{c_1'}^{(b)} \leftarrow \boldsymbol{v}^T R^{(b)}$ for $b \in \{1,2\}$;

$R \leftarrow \mathsf{RREF}(\tilde{C}_n^{(1)} + \tilde{C}_n^{(2)})$;

$\xi_1, \xi_2, \zeta_1 \xleftarrow{\$} \mathbb{Z}_N$;

$h_1 \leftarrow \frac{\xi_2 \boldsymbol{c_1'}^{(1)} - \xi_1 \boldsymbol{c_1'}^{(2)}}{\xi_2 - \xi_1}$;

$h_2 \leftarrow \frac{\boldsymbol{c_1'}^{(1)} - \boldsymbol{c_1'}^{(2)}}{\xi_2 - \xi_1}$;

$\overline{\lambda} \leftarrow \zeta_1 \xi_1$;

$\zeta_2 \leftarrow \frac{\overline{\lambda}}{\xi_2}$;

$U \leftarrow MatFromEigenVecs((\boldsymbol{c_1'}^{(1)}, \zeta_1)(\boldsymbol{c_1'}^{(2)}, \zeta_2))$;

Sample the matrix V such that $R \cdot V = \begin{pmatrix} U & h_1 & h_2 \end{pmatrix}$;

for $i = 1, i++, i \leq \tau - 1$ **do**

  $\boldsymbol{c_i'}^{(b)} \leftarrow E_1' \boldsymbol{c_{i-1}'}^{(b)}$;

for $i = \tau, i++, i \leq n$ **do**

  $\boldsymbol{c_i'}^{(b)} \leftarrow B_1 \boldsymbol{c_{i-1}'}^{(b)}$;

$\overline{\boldsymbol{q_i}} \xleftarrow{\$} (\mathbb{Z}_N)^k$ for $i \in [3, n]$;

$\overline{a_i} \xleftarrow{\$} \mathbb{Z}_N$ for $i \in [1, n]$;

$\overline{H} \leftarrow MatFromEigenVecs((\boldsymbol{c_1'}^{(1)}, \overline{a_1}), (\boldsymbol{c_1'}^{(2)}, \overline{a_2}), (\overline{\boldsymbol{q_3}}, \overline{a_3}), \ldots, (\overline{\boldsymbol{q_n}}, \overline{a_n}))$;

for $i = 0, i++, i \leq n$ **do**

  $\{\xi'_{3(i-1)+1,j}\}_{j=1}^{\tau+i}, \{\xi'_{3(i-1)+2,j,c}\}_{j=1,c=0}^{\tau+i,1}, \{\xi'_{3i,c,j}\}_{j=1,c\in\{2,4\}}^{\tau+i} \xleftarrow{\$} \mathbb{Z}_N$;

  $\overline{\boldsymbol{c}}_{\boldsymbol{3(i-1)+1}}{}^{(b)} \leftarrow \sum_{j=1}^{\tau+i} \xi'_{3(i-1)+1,j} \boldsymbol{c_j'}^{(b)}$ for $b \in \{1,2\}$;

  $\overline{\boldsymbol{c}}_{\boldsymbol{3(i-1)+2,c}}{}^{(b)} \leftarrow \sum_{j=1}^{\tau+i} \xi'_{3(i-1)+2,j,c} \boldsymbol{c_j'}^{(b)}$ for $b \in \{1,2\}$ for $c \in \{1,2\}$;

  $\overline{\boldsymbol{c}}_{\boldsymbol{3i,c}}{}^{(b)} \leftarrow \sum_{j=1}^{\tau+i} \xi'_{3i,c,j} \boldsymbol{c_j'}^{(b)}$ for $b \in \{1,2\}$ for $c \in \{2,4\}$;

$\boldsymbol{u_0} \xleftarrow{\$} (\mathbb{Z}_N)^k$;

$\{\mu_j\}_{j=1}^{\tau} \xleftarrow{\$} \mathbb{Z}_N$;

$\boldsymbol{d_0}^{(b)} \leftarrow \boldsymbol{u_0} + \sum_{j=1}^{\tau} \mu_j \boldsymbol{c_j'}^{(b)}$; `/* For` $b = 1,2$ `we share` $\boldsymbol{u_0}$ `and` $\{\mu_j\}_{j=1}^{\tau}$ `*/`

$\boldsymbol{d_1}^{(b)}, \ldots, \boldsymbol{d_{3n}}^{(b)} \xleftarrow{\$} (\mathbb{Z}_N)^k$ for $b \in \{1,2\}$;

Sample B_0 such that $B_0\tilde{C}_{i-1}^{(b)} \in \tilde{C}_i^{(b)}$ and $B_0 \boldsymbol{d_{3(i-1)+1}}^{(b)} = \overline{\boldsymbol{c}}_{\boldsymbol{3(i-1)+2,0}}$ for $b \in \{1,2\}$ and $i \in [1, n]$;

Sample B_1 such that $B_1\tilde{C}_{i-1}^{(b)} \in \tilde{C}_i^{(b)}$ and $B_1 \boldsymbol{d_{3(i-1)+1}}^{(b)} = \boldsymbol{d_{3(i-1)+2}}^{(b)} + \overline{\boldsymbol{c}}_{\boldsymbol{3(i-1)+2,1}}{}^{(b)}$ for $b \in \{1,2\}$ and $i \in [1, n]$;

Sample E_i such that $E_i\tilde{C}_{i-1}^{(b)} \in \tilde{C}_{i-1}^{(b)}$ and $E_i \boldsymbol{d_{3(i-1)}}^{(b)} = \boldsymbol{d_{3(i-1)+1}}^{(b)} + \overline{\boldsymbol{c}}_{\boldsymbol{3(i-1)+1}}{}^{(b)}$ for $i \in [1, n]$;

Sample $\overline{E}_{i,c}$ such that $\overline{E}_{i,c}\tilde{C}_i^{(b)} \in \tilde{C}_i^{(b)}$ for $c \in \{1,3\}$ for $i \in [1, n]$;

Sample $\overline{E}_{i,2}$ such that $\overline{E}_{i,2}\tilde{C}_i^{(b)} \in \tilde{C}_i^{(b)}$ and $\overline{E}_{i,2} \boldsymbol{d_{3(i-1)+2}}^{(b)} = \overline{\boldsymbol{c}}_{\boldsymbol{3i,2}}{}^{(b)}$ for $i \in [1, n]$;

Sample $\overline{E}_{i,4}$ such that $\overline{E}_{i,4}\tilde{C}_i^{(b)} \in \tilde{C}_i^{(b)}$ and $\overline{E}_{i,4} \boldsymbol{d_{3(i-1)+2}}^{(b)} = \boldsymbol{d_{3i}}^{(b)} + \overline{\boldsymbol{c}}_{\boldsymbol{3i,4}}{}^{(b)}$ for $i \in [1, n]$;

return $B_0, B_1, \{\overline{E}_{i,c}\}_{i=1,c=1}^{n,4}, \{E_i\}_{i=1}^{n}, \{\boldsymbol{c_i'}^{(b)}\}_{i=0,b=1}^{\tau+n,2}, \{\boldsymbol{d_i}^{(b)}\}_{i=0,b=1}^{3n,2}$;

Algorithm 9: Key Generation, Full

Input : $N, k, n, \tau, \alpha = \alpha_1, \ldots, \alpha_n, \beta, v, u$

$B_0, B_1, \{\overline{E}_{i,c}\}_{i=1,c=1}^{n,4}, \{E_i\}_{i=1}^{n}, \{\boldsymbol{c_i'}^{(b)}\}_{i=0,b=1}^{\tau+n,2}, , \{\boldsymbol{d_i}^{(b)}\}_{i=0,b=1}^{3n,2} \leftarrow$ $SampleMatricesFull(N, k, n, \tau, v, u)$

for $i = 0, i++, i \leq n$ **do**

  $F_i \xleftarrow{\$} ((\mathbb{Z}_N)^{k \times k})^*$; `/* F_i should be invertible */`

$\boldsymbol{z}_0{}^{(b)} \leftarrow F_i \boldsymbol{d_0}^{(b)}$ for $b \in \{1, 2\}$;

for $i = 0, i++, i \leq n$ **do**

  if $\alpha_i = 1$ **then**

    $\pi_i \leftarrow \pi_{id}$

  else

    $\pi_i \leftarrow \pi_{sw}$

  $\begin{pmatrix} A_{i,0} \\ A_{i,1} \end{pmatrix} \leftarrow \begin{pmatrix} F_i & 0 \\ 0 & F_i \end{pmatrix} \begin{pmatrix} \overline{E}_{i,1+3\pi_i(0)} & \overline{E}_{i,2+\pi_i(0)} \\ \overline{E}_{i,2+\pi_i(1)} & \overline{E}_{i,1+3+\pi_i(1)} \end{pmatrix} \begin{pmatrix} B_{\pi_i(0)} \\ B_{\pi_i(1)} \end{pmatrix} E_i F_{i-1}^{-1}$

Compute $\boldsymbol{e}$ such that $\left\langle \boldsymbol{e}, \boldsymbol{c_i}^{(b)} \right\rangle = 0$ for all $i \in [1, \tau + n]$ and $b \in \{1, 2\}$;

Compute $\gamma \leftarrow \left\langle \boldsymbol{e}, F_n(\boldsymbol{d_{3n}}^{(1)} + \boldsymbol{d_{3n}}^{(2)}) \right\rangle$;

If γ is not invertible, start again, else $\boldsymbol{z}_0{}^{(b)} \leftarrow \boldsymbol{z}_0{}^{(b)} \cdot \beta \cdot \gamma^{-1}$ for $b \in \{1, 2\}$;

Compute a secret-sharing $[\![\boldsymbol{z}_0{}^{(b)}]\!]$ of $\boldsymbol{z}_0{}^{(b)}$ for $b \in \{1, 2\}$;

Compute $[\boldsymbol{e}]$;

return $\{A_{i,0}\}_{i=1}^{n}, \{A_{i,1}\}_{i=1}^{n}, [\boldsymbol{e}], [\![\boldsymbol{z}_0{}^{(1)}]\!], [\![\boldsymbol{z}_0{}^{(2)}]\!]$;

Algorithm 10: Eval, Full

Input : $\{A_{i,0}\}_{i=1}^{n}, \{A_{i,1}\}_{i=1}^{n}, \boldsymbol{e}, [\![\boldsymbol{z}_0{}^{(1)}]\!], [\![\boldsymbol{z}_0{}^{(2)}]\!], y = y_1 \ldots y_n$

$[\![\overline{\boldsymbol{z}}_0{}^{(1)}]\!] \leftarrow [\![\boldsymbol{z}_0{}^{(1)}]\!]$;

$[\![\overline{\boldsymbol{z}}_0{}^{(2)}]\!] \leftarrow [\![\boldsymbol{z}_0{}^{(2)}]\!]$;

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

  $[\![\overline{\boldsymbol{z}}_i{}^{(1)}]\!] \leftarrow A_{i,y_i} \cdot [\![\overline{\boldsymbol{z}}_{i-1}{}^{(1)}]\!]$;

  $[\![\overline{\boldsymbol{z}}_i{}^{(2)}]\!] \leftarrow A_{i,y_i} \cdot [\![\overline{\boldsymbol{z}}_{i-1}{}^{(2)}]\!]$;

$[\![b]\!] \leftarrow \left\langle \boldsymbol{e}, [\![\overline{\boldsymbol{z}}_n{}^{(1)}]\!] + [\![\overline{\boldsymbol{z}}_n{}^{(2)}]\!] \right\rangle$;

return $[\![b]\!]$;