Low-Communication Multiparty Triple Generation for SPDZ from Ring-LPN

Damiano Abram and Peter Scholl

Aarhus University, Aarhus, Denmark
{damiano.abram, peter.scholl}@cs.au.dk

Abstract. The SPDZ protocol for multi-party computation relies on a correlated randomness setup consisting of authenticated, multiplication triples. A recent line of work by Boyle et al. (Crypto 2019, Crypto 2020) has investigated the possibility of producing this correlated randomness in a *silent preprocessing* phase, which involves a "small" setup protocol with less communication than the total size of the triples being produced. These works do this using a tool called a *pseudorandom correlation generator* (PCG), which allows a large batch of correlated randomness to be compressed into a set of smaller, correlated seeds. However, existing methods for compressing SPDZ triples only apply to the 2-party setting. In this work, we construct a PCG for producing SPDZ triples over large prime fields in the multi-party setting. The security of our PCG is based on the ring-LPN assumption over fields, similar to the work of Boyle et al. (Crypto 2020) in the 2-party setting. We also present a corresponding, actively secure setup protocol, which can be used to generate the PCG seeds and instantiate SPDZ with a silent preprocessing phase. As a building block, which may be of independent interest, we construct a new type of 3-party distributed point function supporting outputs over arbitrary groups (including large prime order), as well as an efficient protocol for setting up our DPF keys with active security.

1 Introduction

Multi-party computation (MPC) allows a set of parties to securely compute on private inputs, while learning nothing but the desired result of the computation. Modern MPC protocols often use a source of secret, *correlated randomness*, which can be distributed to the parties ahead of time, and used to help improve efficiency of the protocol. This is especially important in the dishonest majority setting, where up to $n-1$ out of n parties may be corrupted, since these types of protocols rely on expensive, 'public key'-type cryptographic primitives.

For instance, the SPDZ family of protocols [DPSZ12,DKL+13], which achieves active security with a dishonest majority, uses preprocessed, authenticated multiplication triples, to achieve a very fast online phase where the computation takes place. Multiplication triples, coming from the work of Beaver [Bea92], are triples of random, secret-sharings of values a, b, c over some ring, where $c = a \cdot b$, and allow protocols to offload the heavy work of MPC multiplication to the

preprocessing phase. Unfortunately, producing these triples, although it can be done ahead of time, is still an expensive process in terms of computation, communication and storage costs, since typically a very large number of triples is required, for any reasonably complex computation.

Most current techniques for triple generation are either based on homomorphic encryption [DPSZ12, DKL+13, KPR18] or oblivious transfer [KOS16]. Homomorphic encryption is computationally expensive and also incurs moderately high communication costs (especially due to the use of zero-knowledge proofs for active security), while oblivious transfer is much cheaper computationally, but requires a large amount of bandwidth.

More recently, Boyle et al. [BCG+19b] proposed using *pseudorandom correlation generators* to produce a large amount of correlated randomness without interaction, starting from only a short set of correlated seeds. More concretely, a PCG consists of a seed-generation algorithm, Gen, which outputs a set of correlated seeds $\kappa_0, \ldots, \kappa_{n-1}$, one given to each party. There is then an Expand algorithm, which deterministically expands κ_i into a large amount of correlated randomness R_i. The security requirements are that the expanded outputs $(R_i)_i$ should be indistinguishable from a sample from the target correlation, and furthermore, knowing a subset of the keys should not reveal any information about the missing outputs (beyond what can be deduced from their evaluation). This paradigm offers the potential to greatly reduce communication in the preprocessing of MPC protocols, while also reducing storage costs for the necessary correlated randomness, since the PCG seeds need only be expanded "on-demand".

The first construction of a PCG for authenticated triples [BCG+19b] was based on homomorphic encryption, and not so efficient in practice. However, more recently, the authors proposed another construction [BCG+20] based on a variant of the *ring learning parity with noise* (ring-LPN) assumption. By using distributed point functions [GI14, BGI15] to compress secret-shared, sparse vectors, this construction achieves much better concrete efficiency, as well as a good compression rate.

Unfortunately, both of these PCGs for authenticated, SPDZ-style triples are restricted to the 2-party setting. Note that *unauthenticated* triples, as used in passively secure protocols, can be generated with a PCG in the multi-party setting, with a transformation from [BCG+19b], however, this does not apply to the more complex task of authenticated sharings.

1.1 Our Contributions

In this work, we investigate the possibility of constructing PCGs for SPDZ-style, authenticated triples in the multi-party setting. As our main contributions, we construct such a PCG based on the ring-LPN assumption over large prime fields, and design an actively secure protocol for distributing the PCG seeds among the parties. Our PCG allows expanding short, correlated seeds of size $O(n^3\sqrt{N})$ into N SPDZ triples for n parties. Meanwhile, our actively secure setup protocol produces N SPDZ triples for n parties with $O(n^4\sqrt{N})$ communication. Compared

with previous protocols for SPDZ [KPR18, KOS16], which use $O(n^2 N)$ communication, our protocol scales sublinearly in the number of triples, but is less suitable for a large number of parties. (In the above, we ignore asymptotic factors that only depend on the security parameter.)

Below, we expand on our results in a little more detail.

Background: Construction of [BCG+20]. We first briefly recall the 2-party PCG for authenticated triples from [BCG+20]. Their construction relies on a variant of the ring-LPN (or module-LPN) assumption, which works over the polynomial ring $R = \mathbb{F}[X]/\big(F(X)\big)$, for some finite field $\mathbb{F}$ and fixed polynomial $F(X)$. The assumption, for noise weight t and dimension c, states that the distribution

$$\{\boldsymbol{a}, \langle \boldsymbol{a}, \boldsymbol{e} \rangle \,\big|\, \boldsymbol{a} \leftarrow R^c, \boldsymbol{e} \leftarrow R^c \text{ s.t. } \mathrm{wt}(e_i) = t\}$$

is indistinguishable from random, when each $e_i \in R$ is a sparse polynomial of degree $< N$, with up to t non-zero coefficients. In typical parameters, N will be very large, while c is a small constant, and t the order of the security parameter.

The goal will be to produce a PCG that outputs 2-party, additive shares of a random tuple $(x, y, z, \alpha x, \alpha y, \alpha z)$, where $\alpha \in \mathbb{F}$, $x, y \leftarrow R$ and $z = x \cdot y \in R$. When R is chosen appropriately, such an authenticated triple over R can be locally converted into a large batch of N triples over $\mathbb{F}$.

To obtain a PCG, the construction picks vectors of sparse polynomials $\boldsymbol{u}, \boldsymbol{v} \in R^c$, and computes the tensor product $\boldsymbol{u} \otimes \boldsymbol{v} \in R^{c^2}$. Each of these polynomial products is still somewhat sparse, having at most t^2 non-zero coordinates. The idea is that the sparse $\boldsymbol{u}, \boldsymbol{v}$, as well as their products, can be secret-shared using distributed point functions (DPFs) [GI14, BGI15, BGI16], which provide a way to share sparse vectors in a succinct manner.

Given shares of these values, the parties can locally compute inner products with the public vector $\boldsymbol{a}$, to transform the sparse vectors $\boldsymbol{u}, \boldsymbol{v}$ into pseudorandom polynomials $x = \langle \boldsymbol{a}, \boldsymbol{u} \rangle$ and $y = \langle \boldsymbol{a}, \boldsymbol{v} \rangle$. Similarly, the shares of $\boldsymbol{u} \otimes \boldsymbol{v}$ can be locally transformed into shares of xy, due to bilinearity of the tensor product.

The above blueprint gives additive shares of the (x, y, z) components of the triple. This easily extends to obtain shares of $(\alpha x, \alpha y, \alpha z)$, since multiplying each sparse vector by $\alpha \in \mathbb{F}$ preserves its sparsity, so these can be distributed in the same way.

Using 3-Party Distributed Point Functions. A natural approach to extend the above to more than two parties, is to simply use multi-party DPFs. Unfortunately, existing n-party DPFs [BGI15] scale badly, with a key size growing exponentially in the number of parties. Instead, in the full version of [BCG+20], Boyle et al. sketched an approach using *3-party DPFs*, based on the observation that the product αxy can be broken down into a sum of $\alpha_i x_j y_k$, over parties $i, j, k \in [n]$. This means that each of these terms only needs to be shared between 3 parties, so 3-party DPFs suffice.

However, it turns out this approach is not so straightforward. An immediate challenge is that existing 3-party DPFs only output shares that are XOR-sharings, or shares over $\mathbb{Z}_p$ for small primes p; this excludes the important case of $\mathbb{F}_p$ where p is a large prime, as often used in protocols like SPDZ. Therefore, our first contribution is to construct a 3-party DPF suitable for this setting, by modifying the DPF of Bunn et al. [BKKO20] to work with outputs over any abelian group. Our modification introduces some leakage into the construction: when two specific parties are corrupted, they now learn some information about the secret index of the point function that is being hidden. Fortunately, it turns out that for our application to SPDZ, this leakage is harmless, since it translates to corrupt parties $\{P_j, P_k\}$ learning information on the product $x_j y_k$, which P_j and P_k already know if they collude.

An additional benefit of our DPF, beyond supporting more general outputs, is that our key sizes are smaller than the 3-party DPF of [BKKO20] by around a factor of 3.

PCG for Authenticated Triples. Given our 3-party DPF, we give the full construction of a multi-party PCG for authenticated triples over a large field $\mathbb{F}$. The basic construction for producing N triples with n parties has seeds of size $O(n^3 t^2 \sqrt{N}\lambda)$ bits, where λ is the security parameter and t is roughly λ, although this can be optimized slightly with a more aggressive assumption. Compared with the 2-party PCG of [BCG+20], we incur some extra costs moving to the multi-party setting, since theirs scales with $O(\log N)$ and not $O(\sqrt{N})$. This is due to the $O(\sqrt{N})$ seed size in our DPF, which is also inherited from previous 3-party DPFs [BGI15, BKKO20][1].

Efficient, Actively Secure Distributed Setup for 3-Party DPF. To obtain our triple generation protocol, we need a way of securely setting up the PCG seeds among the parties. The main necessary ingredient is a protocol for distributing the keys in our 3-party DPF. Previously, Bunn et al. [BKKO20] gave a secure protocol for setting up their 3-party DPF keys; however, as well as being very complex, their protocol only has passive security and tolerates 1 out of 3 corruptions. We therefore set out to design an *actively secure* protocol for our 3-party DPF, tolerating any number of corruptions, while only introducing a minimal communication overhead relative to the size of the underlying DPF keys. Our starting point is a lightweight, passively secure setup protocol based on OT and 2-party DPFs, which we combine with a recursive step to generate the necessary "correction word" in the DPF keys. Using recursion here helps to keep the communication overhead down in our protocol. We add active security, by first replacing OT with *authenticated OT*, whereby the receiver's choice bits are authenticated using MACs. We then apply several consistency checks on the DPF keys, including one inspired by a recent OT extension protocol [YWL+20],

[1] In the 2-party setting, there are efficient DPFs with logarithmic key size [BGI15, BGI16].

to prove that the parties behaved honestly. Here, we exploit the fact that the OT choice bits were authenticated, which allows us to reliably perform linear tests on these bits as part of our checks.

Our final setup protocol is very lightweight, and only communicates a small constant factor (2–3x) more information than the size of the DPF keys. On top of the inherent leakage in our DPF, the protocol introduces a small amount of leakage, in the form of allowing the adversary to try and guess some information about the secret point function. This is similar to leakage from other PCG setup protocols based on LPN [BCG+19a, YWL+20], and essentially only translates into an average of one bit of leakage on the (ring)-LPN secret.

Concrete Efficiency. We analyse the concrete efficiency of our actively secure protocol for setting up the PCG seeds, and producing authenticated triples. The main bottleneck is the distributed execution of the 3-party DPF, the only part of the protocol with $\Omega(\sqrt{N})$ complexity. We measure the efficiency of the construction by considering its "stretch", the ratio between the size of the produced triples and the total communication. We observed that the stretch becomes greater than 1 when N is above 2^{24}, meaning producing more than 16 million multiplication triples. When N increases, the stretch improves, reaching values close to 8 for $N = 2^{28}$. This comes, however, at a greater computational cost as the latter scales as $O(N \log(N))$. On the other hand, even for $N = 2^{20}$, our construction performs significantly better than alternative approaches such as Overdrive [KPR18], improving the communication complexity by at least a factor of 10. In this parameter regime, the 2-party PCG of [BCG+20] has practical computational cost, and although we have not implemented our construction, we believe the same will hold since it uses similar building blocks.

2 Notation and Preliminaries

We denote the multiplicative group of a finite field by $\mathbb{F}^\times$. The ideal generated by a polynomial $F(X) \in \mathbb{F}[X]$ is $\big(F(X)\big)$.

When dealing with bit sequences, with an abuse of notation, we identify the sets $\{0,1\}^k$, $\mathbb{F}_2^k$ and $\mathbb{F}_{2^k}$ as different representations of the finite field with 2^k elements. For this reason, when multiplying two elements $a, b \in \{0,1\}^k$, we mean multiplication in $\mathbb{F}_{2^k}$.

Throughout the paper, we will deal with protocols between an ordered set of n parties, $P_1, \ldots, P_n$. We let $\mathcal{H}$ be the set of indices of honest parties, and $\mathcal{C}$ the set of indices of corrupt ones.

The symbol $[m]$ indicates the set $\{0, 1, 2, \ldots, m-1\}$ and $\lfloor \cdot \rfloor$ denotes the integral part of a real number. We represent vectors using bold font, the j-th entry of a vector $\boldsymbol{v}$ is denoted by v_j or $v[j]$. We indicate the scalar product by $\langle \cdot, \cdot \rangle$ The function $\delta_y(\cdot)$ denotes the Kronecker delta function, that is,

$$\delta_y(x) := \begin{cases} 1 & \text{if } x = y, \\ 0 & \text{otherwise.} \end{cases}$$

Given two vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ of dimensions l and m respectively, we denote their outer product by $\boldsymbol{u} \otimes \boldsymbol{v}$. Observe that this is an ml-dimensional vector whose $(im+j)$-th entry is $u_i \cdot v_j$. In a similar way, we define their outer sum $\boldsymbol{u} \boxplus \boldsymbol{v}$ as the ml-dimensional vector whose $(im+j)$-th entry is $u_i + v_j$.

We write $a \xleftarrow{\$} S$, where S is a set, to mean that a is randomly sampled from S. Finally, λ denotes the security parameter and $\mathbb{P}$ represents a probability measure.

Polynomial Rings. Let p be prime and N a positive integer. We will work with the ring $R := \mathbb{F}_p[X]/\big(F(X)\big)$, where $F(X)$ is an irreducible, degree-N polynomial in $\mathbb{Z}[X]$. Similarly to the case of homomorphic encryption [SV14], we will be interested in the case where $F(X)$ factors completely modulo p into a product of distinct, linear terms. In this case, we say that R is fully splittable, and have the isomorphism $R \cong \mathbb{F}_p^N$. This can be ensured, for instance, by choosing N to be a power of 2 and the cyclotomic polynomial $F(X) = X^N + 1$, with $p = 1 \bmod (2N)$.

2.1 Module-LPN

The security of our triple generation protocol relies on the Module-LPN assumption with static leakage, a generalisation of Ring-LPN that was recently studied by Boyle et al. [BCG+20]. We recap here its definition.

Definition 1 (Module-LPN with static leakage). *Let $R := \mathbb{F}_p[X]/\big(F(X)\big)$, for a prime p and $F(X)$ of degree N. Let t and c be two positive integers with $c \geq 2$. Let $\mathcal{HW}_t$ be the distribution that samples t noise positions $\omega[i] \xleftarrow{\$} [N]$ and t payloads $\beta[i] \xleftarrow{\$} \mathbb{F}_p$, outputting the polynomial*

$$e(X) := \sum_{i \in [t]} \beta[i] \cdot X^{\omega[i]}$$

embedded in the ring R. Let $\mathcal{A}$ be a PPT adversary and consider the game $\mathcal{G}^{\text{Module-LPN}}_{R,t,c,\mathcal{A}}(\lambda)$ described in Figure 1. We say that the R^c-LPN$_t$ problem with static leakage is hard if, for PPT adversary $\mathcal{A}$, the advantage

$$Adv^{\text{Module-LPN}}_{R,t,c,\mathcal{A}}(\lambda) := \left| \mathbb{P}\Big(\mathcal{G}^{\text{Module-LPN}}_{R,t,c,\mathcal{A}}(\lambda) = 1\Big) - \frac{1}{2} \right|$$

is negligible in the security parameter λ.

Clearly, in the definition, we assume that the ring R and the values c and t depends on the security parameter λ. Observe that the greater c and t are, the harder the distinguishability becomes. A thorough analysis of the assumption can be found in [BCG+20], including for the case when the polynomial $F(X)$ splits completely into linear factors over $\mathbb{F}$, i.e. when $R \cong \mathbb{F}^N$.

Regarding the leakage, note that in Fig. 1, the adversary's guesses are restricted to *before* it learns the ring-LPN challenge; thus, even though there may be many queries, the resulting leakage is very small: just 1 bit of information on the secret (that is, the fact that all guesses were correct).

$\mathcal{G}^{\text{Module-LPN}}_{R,t,c,\mathcal{A}}(\lambda)$

Initialisation. The challenger activates $\mathcal{A}$ with $\mathbb{1}^\lambda$ and samples a random bit $b \xleftarrow{\$} \{0,1\}$. Then, it samples c elements of the ring $e_0, e_1, \ldots, e_{c-1} \leftarrow \mathcal{HW}_t$. Let the j-th noise positions of e_i be $\omega_i[j]$.

Query. The adversary is allowed to adaptively issue a polynomial number of queries of the form (i, j, I) where $i \in [c]$, $j \in [t]$ and $I \subseteq [N]$. If $\omega_i[j] \in I$, the challenger answers with SUCCESS, otherwise, it sends ABORT and halts.

Challenge. After the Query phase, for every $i \in [c-1]$, the challenger samples $a_i \xleftarrow{\$} R$ and sets

$$u_1 \leftarrow \sum_{i=0}^{c-2} a_i \cdot e_i + e_{c-1}.$$

Moreover, it samples $u_0 \xleftarrow{\$} R$. Finally, it gives $(a_0, a_1, \ldots, a_{c-2}, u_b)$ to the $\mathcal{A}$. The adversary replies with a bit b'. The final output of the game is 1 if and only if $b = b'$.

Fig. 1. The Module-LPN game.

Choice of Error Distribution. The basic module-LPN definition assumes each error polynomial is chosen uniformly, subject to having t non-zero coefficients. We can also improve efficiency with more structured errors, such as *regular errors*, where the non-zero coordinates are more evenly spaced out, so that each is guaranteed to lie in a unique interval of size N/t. We use this variant in our efficiency estimates to improve parameters. Note that it has also been used previously [BCG+19b, BCG+20, YWL+20], and is conjectured to have essentially the same security as the standard assumption.

2.2 Pseudorandom Correlation Generators

To obtain a low communication complexity, our protocol uses pseudorandom correlation generators (PCGs) [BCG+19a, BCG+19b, BCG+20]. An n-party PCG is a pair of algorithms, the first of which outputs n correlated seeds of relatively small size. These can be, later on, locally expanded by the parties to obtain a large amount of desired correlated randomness. Since the expansion phase does not require any communication between the parties and the seed size is small compared to the output, the hope is to design low-communication protocols that securely generate and distribute the seeds to the parties. This allows the secure generation of large amounts of correlated randomness with low communication complexity.

We now give the formal definition of PCG [BCG+19b], starting from the notion of correlation generator.

Definition 2 (Correlation Generator). *An n-party correlation generator is a PPT algorithm $\mathcal{C}_{corr}$ that, on input $\mathbb{1}^\lambda$, outputs n polynomial-size elements in $\{0,1\}^*$.*

Definition 3 (Reverse samplability). *An n-party correlation generator $\mathcal{C}_{corr}$ is reverse-samplable if there exists a PPT algorithm* $\mathsf{RSample}$ *such that, for every $T \subseteq [n]$,*

$$\left\{ (R'_0, R'_1, \ldots, R'_{n-1}) \;\middle|\; \begin{array}{l} (R_0, R_1, \ldots, R_{n-1}) \leftarrow \mathcal{C}_{corr}(1\!\!1^\lambda) \\ R'_i \leftarrow R_i \quad \forall i \in T^{\complement} \\ (R'_i)_{i \in T} \leftarrow \mathsf{RSample}\big(1\!\!1^\lambda, T, (R_i)_{i \in T^{\complement}}\big) \end{array} \right\}$$

is computationally indistinguishable from the output of $\mathcal{C}_{corr}(1\!\!1^\lambda)$.

In general, the i-th output of a correlation generator is given to the i-th party. The reverse samplability property states that given any subset of the outputs of a correlator generator, there exists an efficient algorithm that simulates the remaining outputs. Given this, an n-party PCG is defined as follows.

Definition 4 (PCG). *Let $\mathcal{C}_{corr}$ be a reverse-samplable n-party correlation generator. A PCG for $\mathcal{C}_{corr}$ is a pair of PPT algorithms* $(\mathsf{PCG.Gen}, \mathsf{PCG.Expand})$ *such that*

- *On input $1\!\!1^\lambda$,* $\mathsf{PCG.Gen}$ *outputs n seeds $\kappa_0, \kappa_1, \ldots, \kappa_{n-1}$.*
- *On input (i, κ_i) for $i \in [n]$,* $\mathsf{PCG.Expand}$ *outputs an element $R_i \in \{0,1\}^*$.*
- *(**Correctness**). The distribution of*

$$\left\{ (R_0, R_1, \ldots, R_{n-1}) \;\middle|\; \begin{array}{l} (\kappa_0, \kappa_1, \ldots, \kappa_{n-1}) \leftarrow \mathsf{PCG.Gen}(1\!\!1^\lambda) \\ R_i \leftarrow \mathsf{PCG.Expand}(i, \kappa_i) \quad \forall i \in [n] \end{array} \right\}$$

  is computationally indistinguishable from the distribution of the output of $\mathcal{C}_{corr}(1\!\!1^\lambda)$.
- *(**Security**). For every $T \subseteq [n]$, the following two distributions are computationally indistinguishable*

$$\left\{ \big((\kappa_i)_{i \in T^{\complement}}, (R_i)_{i \in T}\big) \;\middle|\; \begin{array}{l} (\kappa_0, \kappa_1, \ldots, \kappa_{n-1}) \leftarrow \mathsf{PCG.Gen}(1\!\!1^\lambda) \\ R_i \leftarrow \mathsf{PCG.Expand}(i, \kappa_i) \quad \forall i \in T \end{array} \right\} \quad \textit{and}$$

$$\left\{ \big((\kappa_i)_{i \in T^{\complement}}, (R_i)_{i \in T}\big) \;\middle|\; \begin{array}{l} (\kappa_0, \kappa_1, \ldots, \kappa_{n-1}) \leftarrow \mathsf{PCG.Gen}(1\!\!1^\lambda) \\ R_i \leftarrow \mathsf{PCG.Expand}(i, \kappa_i) \quad \forall i \in T^{\complement} \\ (R_i)_{i \in T} \leftarrow \mathsf{RSample}\big(1\!\!1^\lambda, T, (R_i)_{i \in T^{\complement}}\big) \end{array} \right\}$$

Essentially, correctness requires that the joint distribution of the parties' outputs $(R_1, \ldots, R_n)$ is indistinguishable from the target correlation $\mathcal{C}_{\text{corr}}$. The security property states that the knowledge of a subset of the seeds leaks no information about the other outputs, that could not already be inferred from the knowledge of the expansion of the given seeds.

2.3 Distributed Point Functions

In [GI14], Gilboa and Ishai introduced distributed point functions (DPFs). A point function is a function f whose support (i.e. the elements which have non-zero image) contains at most one element. Therefore, if the domain has size N, we can regard f as an N-dimensional vector with at most one non-zero entry, whose i-th entry, for $i \in [N]$, corresponds to the evaluation $f(i)$. We call such vector a unit vector, and often refer to the index of the non-zero entry as the *special position* and its value as the *non-zero element*.

An n-party DPF consists of a pair of algorithms, the first of which takes as input the description of a point function f and outputs n succinct keys. These can be, later on, locally evaluated by the parties on input x to obtain a secret-sharing of $f(x)$. DPFs and PCGs have some similarity, in that in both cases, we have an initial phase in which correlated, succinct keys are generated, followed by an evaluation phase that locally produces the desired output. The analogy between the two notions is the reason why DPFs are often a key building block of PCGs. Our protocol is no exception.

Definition 5 (DPF with leakage). *Let $(\mathbb{G}, +)$ be an abelian group and let N be a positive integer. An n-party distributed point function (DPF) for $(N, \mathbb{G})$ with leakage function Leak is a pair of PPT algorithms $(\mathsf{DPF}_N^n.\mathsf{Gen}, \mathsf{DPF}_N^n.\mathsf{Eval})$ with the following syntax:*

- *On input $\mathbb{1}^\lambda$, $\omega \in [N]$ and $\beta \in \mathbb{G}$, $\mathsf{DPF}_N^n.\mathsf{Gen}$ outputs n keys $\kappa_0, \kappa_1, \ldots, \kappa_{n-1}$.*
- *On input (i, κ_i, x) for $i \in [n]$ and $x \in [N]$, $\mathsf{DPF}_N^n.\mathsf{Eval}$ outputs a value $v_i \in \mathbb{G}$.*

Moreover, the following properties are satisfied

- *(**Correctness**). For every $x, \omega \in [N]$ and $\beta \in \mathbb{G}$,*

$$\mathbb{P}\left(\sum_{i=0}^{n-1} v_i = \beta \cdot \delta_\omega(x) \;\middle|\; \begin{array}{l}(\kappa_0, \kappa_1, \ldots, \kappa_{n-1}) \leftarrow \mathsf{DPF}_N^n.\mathsf{Gen}(\mathbb{1}^\lambda, \omega, \beta) \\ v_i \leftarrow \mathsf{DPF}_N^n.\mathsf{Eval}(i, \kappa_i, x) \quad \forall i \in [n]\end{array}\right) = 1.$$

- *(**Security**). There exists a PPT simulator Sim such that for every $T \subsetneq [n]$, $\omega \in [N]$ and $\beta \in \mathbb{G}$, the following distributions are computationally indistinguishable*

$$\Big\{(\kappa_i)_{i\in T} | (\kappa_0, \kappa_1, \ldots, \kappa_{n-1}) \leftarrow \mathsf{DPF}_N^n.\mathsf{Gen}(\mathbb{1}^\lambda, \omega, \beta)\Big\} \equiv_C$$
$$\Big\{(\kappa_i)_{i\in T} \leftarrow \mathsf{Sim}\big(\mathbb{1}^\lambda, T, \mathsf{Leak}(T, \omega, \beta)\big)\Big\}.$$

Essentially, correctness requires that the evaluation of the keys on x is a secret-sharing of β if $x = \omega$, or of 0 otherwise. Security instead states that the information inferable from a subset of the keys is bounded by the leakage function Leak, which takes as input the special position ω, the non-zero value β and the set of corrupted parties T. In most cases, Leak just outputs the domain size N and the codomain $\mathbb{G}$ of the point function. However, this will not happen in the DPF on which our protocol relies.

We write $\mathsf{DPF}_N^n.\mathsf{FullEval}(i, \kappa_i)$ to mean the result of calling Eval on the entire domain of the function, obtaining a secret-sharing of the full length-N unit vector.

State-of-the-art. Actually, very little is known about DPFs. In [BGI15], the authors presented a 2-party DPF with $O\big(\log(N)\big)$ key size and an n-party construction with $O(\sqrt{N})$ key size. In both cases, the only leakage is N and $\mathbb{G}$, however, while the 2-party construction allows outputs in any group $\mathbb{G}$, the multi-party DPF essentially works only when $\mathbb{G} = (\{0,1\}^l, \oplus)$, or when $\mathbb{G}$ has polynomial order. In [BKKO20], Bunn et al. presented an improved version of the second algorithm for the 3-party case, however, obtaining again $O(\sqrt{N})$ key size. As we will show in Section 3, this construction is also limited to outputs in $\mathbb{G} = (\{0,1\}^l, \oplus)$, and does not extend to e.g. $\mathbb{F}_p$ for a large prime p.

Distributed Sum of Point Functions We use a simple extension of DPFs to sums of point functions, as also done in [BCG+20]. A DSPF scheme $\mathsf{DSPF}^n_{N,t}$ consists of algorithms $(\mathsf{DSPF}^n_{N,t}.\mathsf{Gen}, \mathsf{DSPF}^n_{N,t}.\mathsf{Eval})$, just as a DPF, except now Gen takes as input a pair of length-t vectors $\boldsymbol{\omega}, \boldsymbol{\beta} \in [N]^t \times \mathbb{G}^t$, which define the sum of point functions

$$f_{\boldsymbol{\omega},\boldsymbol{\beta}}(x) = \sum_{i \in [t]} \boldsymbol{\beta}[i] \cdot \delta_{\boldsymbol{\omega}[i]}(x)$$

Observe that $f_{\boldsymbol{\omega},\boldsymbol{\beta}}$ can be represented as a sum of unit vectors. We will refer to the latter as a multi-point vector.

The correctness property of a DSPF is then the same as a DPF, except we require that $\sum_{i \in [n]} v_i = f_{\boldsymbol{\omega},\boldsymbol{\beta}}(x)$, where $v_i = \mathsf{DSPF}^n_{N,t}.\mathsf{Eval}(i, \kappa_i, x)$. The security property is defined the same way as in a DPF.

Given a DPF, constructing a t-point DSPF can be done in the natural way, using one DPF instance for each of the t points, and summing up the t outputs of $\mathsf{DPF}^n_N.\mathsf{Eval}$ to evaluate the DSPF.

3 Generalisation of the 3-party DPF to Prime Fields

In this section, we first recap the 3-party DPF of [BKKO20], and then describe our extension of this to support outputs modulo p for any prime p.

High-level description of [BKKO20]. The scheme assumes N, the domain size, is a perfect square, and the codomain is $\mathbb{F}_{2^l}$. It uses a PRG $G : \{0,1\}^\lambda \longrightarrow \mathbb{F}_{2^l}^{\sqrt{N}}$. DPF keys in their construction do not leak anything beyond the domain and codomain, namely, the leakage function is given by $\mathsf{Leak}(T, \omega, \beta) = (N, \mathbb{F}_{2^l})$ for every subset of parties $T \subsetneq [3]$, special position $\omega \in [N]$ and non-zero value $\beta \in \mathbb{F}_{2^l}$.

During key generation, the unit vector representing the point function is rearranged into a $\sqrt{N} \times \sqrt{N}$ matrix M. If we rewrite $x \in [N]$ as $x'\sqrt{N} + x''$ with $0 \leq x', x'' < \sqrt{N}$, the x-th element of the unit vector is moved to the x'-th row

and x''-th column of the matrix M. We call the row containing β the special row.

$$M := \begin{bmatrix} 0\,0 \cdots \cdots & 0 & \cdots \cdots \cdots & 0 \\ 0\,0 \cdots \cdots & 0 & \cdots \cdots \cdots & 0 \\ \vdots\,\vdots & \vdots & & \vdots \\ 0\,0 & 0 & & 0 \\ 0\,0 \cdots\ 0 & \beta & 0 \cdots \cdots & 0 \\ 0\,0 & 0 & & 0 \\ \vdots\,\vdots & \vdots & & \vdots \\ 0\,0 \cdots \cdots & 0 & \cdots \cdots \cdots & 0 \end{bmatrix} \leftarrow \omega' \qquad \uparrow\ \omega''$$

The algorithm is essentially based on the observation that it is possible to compress a 3-party secret-sharing of a row of zeros. Indeed, it suffices to sample 3 random PRG seeds a_j, b_j, c_j for every row j and give $\{a_j, b_j\}$ to P_0, $\{b_j, c_j\}$ to P_1 and $\{c_j, a_j\}$ to P_2. To decompress, each party just has to evaluate the seeds and XOR the results. We obtain a secret-sharing of zero since

$$\Big(G(a_j) \oplus G(b_j)\Big) \oplus \Big(G(b_j) \oplus G(c_j)\Big) \oplus \Big(G(c_j) \oplus G(a_j)\Big) = \mathbf{0}.$$

In order to not leak ω', the parties need to obtain similar seeds for the special row too. Observe that for every row, an adversary controlling two parties sees two sets of seeds with only one element in common, therefore, security requires that the property to hold for the special row too. For this reason, the algorithm samples 4 PRG seeds $a_{\omega'}, b_{\omega'}, c_{\omega'}, d_{\omega'}$ and gives $\{a_{\omega'}, d_{\omega'}\}$ to P_0, $\{b_{\omega'}, d_{\omega'}\}$ to P_1 and $\{c_{\omega'}, d_{\omega'}\}$ to P_2. Observe how the property is still satisfied.

Although security is guaranteed, the seeds $a_{\omega'}, b_{\omega'}, c_{\omega'}, d_{\omega'}$ are not a compression of the special row. Indeed, by expanding them ad XORing the results as for the other rows, we obtain a secret-sharing of a random vector

$$\boldsymbol{r} := G(a_{\omega'}) \oplus G(b_{\omega'}) \oplus G(c_{\omega'}) \oplus G(d_{\omega'}).$$

Observe that when there exists at least one honest party, one of the seeds remains unknown to the adversary, therefore, $\boldsymbol{r}$ is always indistinguishable from random. The DPF exploits this fact to include the correction word

$$\boldsymbol{CW} := \boldsymbol{r} \oplus (\overbrace{\underbrace{0, 0, \ldots, 0}_{\omega''}, \beta, 0, 0, \ldots, 0}^{\sqrt{N} \text{ elements}})$$

to the key of every party. By adding the correction word to the expansion of the seeds of the special row, we obtain exactly what we desire, however, we must find a way to perform this operation without leaking the position ω'. The algorithm solves the problem by including in the keys a secret-sharing $[[\boldsymbol{y}]]_2$ of the unit

vector having 1 in the special position ω'. Let $y_i[x']$ denote the x'-th bit of P_i's share of $\boldsymbol{y}$. By summing $y_i[x'] \cdot \boldsymbol{CW}$ to the expansion of the seeds, we add the correction word only to the special row. To summarise, the evaluation algorithm retrieves the row corresponding to the point that has to be evaluated, expands the associated seeds and obliviously adds the correction word when necessary.

Prime field generalisation. In order to generate multiplication triples over large prime fields $\mathbb{F}$ following the blueprint described in the introduction, we needed a 3-party DPF with codomain $\mathbb{F}$. Therefore, the first necessary step was to generalise the construction of [BKKO20]. As we have already mentioned, our modification requires weakening security, by introducing additional leakage.

The issue. The main cause of problems is that large prime fields have characteristic different from 2 and therefore addition and subtraction are different operations. Referring to the roadmap in the previous section, we can still compress a secret-sharing of zero by sampling 3 PRG seeds a_j, b_j, c_j and giving $\{a_j, b_j\}$ to P_0, $\{b_j, c_j\}$ to P_1 and $\{c_j, a_j\}$ to P_2. However, the decompression requires attention, indeed, when two parties have a seed in common, one of them has to add its expansion to its secret-sharing, the other one has to subtract it. It is therefore necessary to associate every seed in the keys with a bit, which will be set if and only if the expansion of the seed has to be added. Whenever two parties have a seed in common, the associated bits will be opposites.

This property has to be satisfied by the seeds of the special row too. One possibility would be of course to do exactly the same as for the normal rows, obtaining a secret-sharing of zero. However, that would not allow us to use the correction word $\boldsymbol{CW}$ as it would leak the non-zero value β. The only other possibility would be to sample 4 PRG seeds $a_{\omega'}, b_{\omega'}, c_{\omega'}, d_{\omega'}$ as before and give $\{a_{\omega'}, d_{\omega'}\}$ to P_0, $\{b_{\omega'}, d_{\omega'}\}$ to P_1 and $\{c_{\omega'}, d_{\omega'}\}$ to P_2. Clearly, we have to associate every seed with a bit expressing whether its expansion has to be added or subtracted, but whatever way we do it for $d_{\omega'}$, there will always be two parties with the same bit. If those two parties are corrupted, the value of ω' is leaked to them, compromising security of the DPF. On the other hand, this leakage turned out not to be problematic for our application.

Our solution. We decided to generate the sign bits so that ω' is leaked when the last two parties are corrupted. Since these bits do not need to be random, it is enough to ensure that the first party always subtracts the expansion of its seeds, the second party always adds them and the last party always adds the expansion of the first seed and subtracts the expansion of the second one. This means that the seeds of the second and the third party now have to be ordered. For instance, when $j \neq \omega'$, we can give $\{a_j, b_j\}$ to P_0, (b_j, c_j) to P_1 and (a_j, c_j) to P_2. When instead $j = \omega'$, we can give $\{a_j, d_j\}$ to P_0, (d_j, b_j) to P_1 and (d_j, c_j) to P_2. The construction is secure as long as the seeds in common with the first party are always in the first position of P_1 and P_2's pairs (which are ordered). On the other hand, it is crucial that the seeds of P_0 are an unordered set, otherwise ω' would be leaked to the adversary when P_0 and P_1, or P_0 and P_2 are both corrupted.

The fact that we do not need to protect ω' from an adversary corrupting the last two parties allows us to further improve the efficiency of the construction. For instance, we can secret-share $\boldsymbol{y}$ only between the second and the third party and remove the correction word from the key of the first party. Actually, since $\boldsymbol{y}$ is a unit vector, we can further compress the secret-sharing using the 2-party DPF of [BGI15], which has logarithmic key size.

Also, the seeds (a_j, b_j, c_j) can be somewhat compressed. If we consider the last seeds of the second and the third party, we observe that they coincide for every $j \neq \omega'$. When instead $j = \omega'$, the two seeds are independent. Essentially, they form a secret-sharing over $\mathbb{F}_{2^\lambda}$ of a $\sqrt{N}$-dimensional unit vector having special position ω' and random non-zero element. Such a secret-sharing can again be compressed using a 2-party DPF, such as from [BGI15, BGI16].

As a final optimization, it turns out the remaining seeds can also be compressed by roughly a factor of 2. This technique relies on the fact that we can generate the missing seeds using Random-OT tuples[2], which can themselves be compressed using a PCG based on the LPN assumption with *logarithmic* overhead [BCG+19b]. We omit the details here for ease of presentation, but the technique is used in our 3-party DPF protocol in Section 5.

Construction and Concrete Efficiency. Our 3-party DPF following the above ideas is given in Figure 2. The construction assumes the domain size is a perfect square and has a prime field $\mathbb{F}$ as codomain. The size of the key κ_0 is $\sqrt{N} \cdot 2\lambda$ bits, while the size of κ_1 and κ_2 is dominated by $\sqrt{N} \cdot (\lambda + \log|\mathbb{F}|) + O(\log(N) \cdot \lambda)$ bits. When $|\mathbb{F}| \approx 2^\lambda$, this gives a total of around $6\sqrt{N}\lambda$ bits for all three keys. If we additionally apply the optimization mentioned above, and compress the seeds using random OT and LPN, the total key size falls to $3\sqrt{N}\lambda$ bits (ignoring small $\log N$ terms), which is around 3x smaller than that of [BKKO20] (which only works in groups of small characteristic, but on the other hand, does not leak any information on ω).

Theorem 1. *The construction described in Figure 2 is a 3-party DPF for* $(N, \mathbb{F})$ *with leakage*

$$\mathsf{Leak}(T, \omega, \beta) = \begin{cases} (N, \mathbb{F}) & \text{if } T \neq \{1, 2\}, \\ \left(N, \mathbb{F}, \lfloor \omega / \sqrt{N} \rfloor\right) & \text{if } T = \{1, 2\}. \end{cases}$$

The proof of Theorem 1 is available is Appendix A.

Extension to Distributed Sum of Point Functions. In later sections, we will use a distributed *sum of* point functions, built on top of our 3-party DPF in the naive way, as described in Section 2.3. Here, the leakage function is extended to output $\lfloor \omega_i / \sqrt{N} \rfloor$, for each special position ω_i, for $i \in [t]$, when the set of corruptions is $T = \{1, 2\}$.

[2] Tuples $\big((X_0, X_1), (b, X_b)\big)$ where $X_0, X_1 \xleftarrow{\$} \{0,1\}^\lambda$ and $b \xleftarrow{\$} \{0,1\}$.

Prime field 3-party DPF

Let N be a perfect square and suppose that $\mathbb{G} = \mathbb{F}$. Let $G : \{0,1\}^\lambda \longrightarrow \mathbb{F}^{\sqrt{N}}$ be a PRG and let $\mathsf{DPF}^2_{\sqrt{N}}$ denote a 2-party DPF with domain size $\sqrt{N}$.

DPF.Gen. On input $\mathbb{1}^\lambda$, $\omega \in [N]$ and $\beta \in \mathbb{F}$, perform the following operations:

1. Rewrite ω as $\omega' \cdot \sqrt{N} + \omega''$ where $0 \leq \omega', \omega'' < \sqrt{N}$.
2. Sample $\Delta \xleftarrow{\$} \mathbb{F}_{2^\lambda}$ and compute

$$(\widehat{\kappa}^1_1, \widehat{\kappa}^2_1) \leftarrow \mathsf{DPF}^2_{\sqrt{N}}.\mathsf{Gen}(\mathbb{1}^\lambda, \omega', 1), \qquad (\widehat{\kappa}^1_2, \widehat{\kappa}^2_2) \leftarrow \mathsf{DPF}^2_{\sqrt{N}}.\mathsf{Gen}(\mathbb{1}^\lambda, \omega', \Delta).$$

3. For every $j \in [\sqrt{N}]$ with $j \neq \omega'$, sample $a_j, b_j \xleftarrow{\$} \{0,1\}^\lambda$ and set

$$S^0_j \leftarrow \{a_j, b_j\}, \qquad S^1_j \leftarrow b_j, \qquad S^2_j \leftarrow a_j.$$

4. Sample $a_{\omega'}, d_{\omega'} \xleftarrow{\$} \{0,1\}^\lambda$ and set

$$S^0_{\omega'} \leftarrow \{a_{\omega'}, d_{\omega'}\}, \qquad S^1_{\omega'} \leftarrow d_{\omega'}, \qquad S^2_{\omega'} \leftarrow d_{\omega'}.$$

5. Compute

$$b_{\omega'} \leftarrow \mathsf{DPF}^2_{\sqrt{N}}.\mathsf{Eval}(1, \widehat{\kappa}^1_2, \omega'), \qquad c_{\omega'} \leftarrow \mathsf{DPF}^2_{\sqrt{N}}.\mathsf{Eval}(2, \widehat{\kappa}^2_2, \omega'),$$

$$\boldsymbol{CW} \leftarrow G(a_{\omega'}) - G(b_{\omega'}) + G(c_{\omega'}) - G(d_{\omega'}) + (\underbrace{0, 0, \ldots, \overbrace{0}^{\omega''}}_{\sqrt{N} \text{ elements}}, \beta, 0, 0, \ldots, 0).$$

6. Output $(\kappa_0, \kappa_1, \kappa_2)$ where

$$\kappa_0 := (S^0_j)_{j \in [\sqrt{N}]}, \qquad \kappa_i := \left(\widehat{\kappa}^i_1, \widehat{\kappa}^i_2, (S^i_j)_{j \in [\sqrt{N}]}, \boldsymbol{CW}\right) \quad \text{if } i \in \{1,2\}.$$

DPF.Eval. On input $i \in [3]$, the key κ_i and a point $x \in [N]$, perform the following operations:

1. Rewrite x as $x' \cdot \sqrt{N} + x''$ where $0 \leq x', x'' < \sqrt{N}$.
2. If $i = 0$, pick an arbitrary ordering and rewrite $S^0_{x'}$ as (w^0_1, w^0_2).
3. If $i \in \{1,2\}$, set $w^i_1 \leftarrow S^i_{x'}$ and compute

$$y_i[x'] \leftarrow \mathsf{DPF}^2_{\sqrt{N}}.\mathsf{Eval}(i, \widehat{\kappa}^i_1, x'), \qquad w^i_2 \leftarrow \mathsf{DPF}^2_{\sqrt{N}}.\mathsf{Eval}(i, \widehat{\kappa}^i_2, x').$$

4. Compute

$$\boldsymbol{v^i_{x'}} \leftarrow \begin{cases} -G(w^0_1) - G(w^0_2) & \text{if } i = 0, \\ y_i[x'] \cdot \boldsymbol{CW} + G(w^1_1) + G(w^1_2) & \text{if } i = 1, \\ y_i[x'] \cdot \boldsymbol{CW} + G(w^2_1) - G(w^2_2) & \text{if } i = 2. \end{cases}$$

5. Output $v^i_{x'}[x'']$.

Fig. 2. The prime field 3-party DPF

4 Multiparty PCG for Triple Generation

In this section, we show how to use our 3-party DPF to construct a multi-party, pseudorandom correlation generator for authenticated triple generation.

Authenticated secret-sharing. We produce additively secret-shared values with information-theoretic MACs, as used in SPDZ [DPSZ12, DKL+13]. Here, an n-party secret-sharing of $x \in \mathbb{F}$ is given by a tuple

$$[\![x]\!] := (\alpha_i, x_i, m_{x,i})_{i\in[n]}$$

where (α_i, x_i, m_{x_i}) are known to the i-th party. Each $\alpha_i \in \mathbb{F}$ is fixed for every sharing x, and is a share of the global MAC key $\alpha = \sum_i \alpha_i$. The shares $x_i \in \mathbb{F}$ and MAC shares $m_{x,i} \in \mathbb{F}$ then satisfy

$$\sum_i x_i = x, \quad \sum_i m_{x,i} = \alpha \cdot x$$

We construct a PCG for the correlation which samples a random triple $([\![x]\!], [\![y]\!], [\![z]\!])$, where x, y are random elements of the ring $R = \mathbb{F}[X]/F(X)$, and $z = x \cdot y$ (while the MAC key α is a scalar in $\mathbb{F}$). As discussed in Section 2, when p is a suitable prime and $F(X)$ is e.g. a cyclotomic polynomial of degree N, this is equivalent to a batch of N triples over $\mathbb{F}$, thanks to the CRT isomorphism $R \cong \mathbb{F}^N$.

Note that it is easy to see that this correlation satisfies the reverse-samplable requirement.

4.1 Construction

Our construction is given in Fig. 3 and Fig. 4. We combine 3-party DPFs with the ring-LPN assumption, following the outline in the introduction (also sketched in [BCG+20]).

In more detail, we will compress the x, y terms of the triple using sparse, random polynomials $u^r(X), v^r(X) \in R$, for $r \in [c]$. Recall that if $\boldsymbol{a} \in R^c$ is a public, random vector over R, then

$$x = \langle \boldsymbol{a}, \boldsymbol{u} \rangle, \quad y = \langle \boldsymbol{a}, \boldsymbol{v} \rangle$$

are computationally indistinguishable from random R elements, under the module-LPN assumption.

We sample the u^r, v^r polynomials by first picking sparse u_i^r, v_i^r for each party, and summing up these shares. These are implicitly defined in steps 2 of Fig. 3, which sample the non-zero coefficients and values of the polynomials.

Then, we use 2-party distributed (sums of) point functions to compress additive shares of the cross-products $\alpha_j \cdot u_i^r, \alpha_j \cdot v_i^r$ and $u_i^r \cdot v_j^s$, in steps 3–4. This allows the parties to obtain shares of the MACs $\alpha x, \alpha y$, as well as the product xy.

Finally, to obtain shares of αxy, we decompose this into a sum of products $\alpha_i \cdot x_j \cdot y_k$, for every $i, j, k \in [n]$. By distributing shares of each term $\alpha_i \cdot u_j^r \cdot v_k^s$

using the 3-party DPF from Section 3, the parties can locally recover shares of αxy in the evaluation stage.

Note that due to the leakage in our 3-party DPF, if P_j and P_k are both corrupted, they learn something about the indices of the non-zero entries in $\alpha_i \cdot u_j^r \cdot v_k^s$. However, since these indices are independent of α_i, this leakage does not give away anything that wasn't already known to P_j and P_k.

In Appendix B,we prove the following.

Theorem 2. *Suppose that* $\mathsf{DSPF}^2_{N,t}$, DSPF^2_{2N,t^2} *and* DSPF^3_{2N,t^2} *are secure distributed sums of point functions, and the* R^c*-LPN*$_t$ *assumption (Definition 1) holds. Then the construction in Fig. 3–4 is a secure PCG for n-party authenticated triples over* $R = \mathbb{F}[X]/F(X)$.

Efficiency. Note that we can optimize the construction slightly, with the observation that in any 3-party DPF instance where two of i, j, k are equal, we can instead use a 2-party DPF. This reduces the total number of 3-party DSPFs from $c^2n^2(n-1)$ down to $c^2n(n-1)(n-2)$. Each DSPF has t^2 points and a domain of size $2N$. There are also $O(c^2n^2)$ 2-party DSPFs, however, since these have logarithmic key size, their cost is dominated by the 3-party instances.

As a further optimization, we can rely on module-LPN with a *regular error distribution* [BCG+20], where each of the t non-zero entries in an error vector is sampled to be within a fixed range of length N/t. This reduces the domain size of the DPFs from $2N$ and N down to $2N/t$ and N/t, respectively.

In Section 7.2, we analyze the concrete parameters of our PCG, and the efficiency of our protocol for securely setting up the seeds and producing triples.

5 Distributed Setup for the 3-Party DPF

We now present an actively secure protocol that permits to distribute the keys of the 3-party DPF described in Section 3. We start by giving an overview of the passively secure approach; later we will delve into the details, including active security.

High-level overview. The protocol permits to derive a 3-party secret-sharing of the unit-vector

$$(\underbrace{\overbrace{0,0,\ldots,0}^{\omega},\beta,0,0,\ldots,0}_{N})$$

given secret-shared special position and non-zero value $[[\omega]]_2$ and $[[\beta]]$. Writing $\omega = \omega'\sqrt{N} + \omega''$, and following the blueprint of our 3-party DPF, the protocol samples a random $\Delta \in \mathbb{F}_{2^\lambda}$ and shares the unit vectors

$$\boldsymbol{y} = (\underbrace{\overbrace{0,0,\ldots,0}^{\omega'},1,0,0,\ldots,0}_{\sqrt{N}}), \qquad \boldsymbol{Y} = (\underbrace{\overbrace{0,0,\ldots,0}^{\omega'},\Delta,0,0,\ldots,0}_{\sqrt{N}})$$

$\mathbf{PCG}_{\text{triple}}$

Let $\mathbb{F}$ be a prime field and let N be the number of generated triples. Let t and c be the parameters of the Module-LPN assumption.

Gen: On input $1\!\!1^\lambda$, do the following:

1. Sample MAC key shares $\alpha_i \xleftarrow{\$} \mathbb{F}$, for every $i \in [n]$.
2. For every $i \in [n]$, $r \in [c]$, sample $\boldsymbol{\omega}_i^r, \boldsymbol{\eta}_i^r \xleftarrow{\$} [N]^t$ and $\boldsymbol{\beta}_i^r, \boldsymbol{\gamma}_i^r \xleftarrow{\$} \mathbb{F}^t$.
3. For every $i, j \in [n]$ with $i \neq j$, $r \in [c]$, compute

$$\left(U_{i,j}^{r,0}, U_{i,j}^{r,1}\right) \leftarrow \mathsf{DSPF}_{N,t}^2.\mathsf{Gen}\left(1\!\!1^\lambda,\ \boldsymbol{\omega}_i^r,\ \alpha_j \cdot \boldsymbol{\beta}_i^r\right),$$
$$\left(V_{i,j}^{r,0}, V_{i,j}^{r,1}\right) \leftarrow \mathsf{DSPF}_{N,t}^2.\mathsf{Gen}\left(1\!\!1^\lambda,\ \boldsymbol{\eta}_i^r,\ \alpha_j \cdot \boldsymbol{\gamma}_i^r\right).$$

4. For every $i, j \in [n]$ with $i \neq j$, $r, s \in [c]$, compute

$$\left(C_{i,j}^{r,s,h}\right)_{h\in[2]} \leftarrow \mathsf{DSPF}_{2N,t^2}^2.\mathsf{Gen}\left(1\!\!1^\lambda,\ \boldsymbol{\omega}_i^r \boxplus \boldsymbol{\eta}_j^s,\ \boldsymbol{\beta}_i^r \otimes \boldsymbol{\gamma}_j^s\right).$$

5. For every $i, j, k \in [n]$ with i, j, k not all equal, for $r, s \in [c]$, compute

$$\left(W_{i,j,k}^{r,s,h}\right)_{h\in[3]} \leftarrow \mathsf{DSPF}_{2N,t^2}^3.\mathsf{Gen}\left(1\!\!1^\lambda,\ \boldsymbol{\omega}_j^r \boxplus \boldsymbol{\eta}_k^s,\ \alpha_i \cdot (\boldsymbol{\beta}_j^r \otimes \boldsymbol{\gamma}_k^s)\right).$$

6. For every $i \in [n]$, output the seed

$$\kappa_i \leftarrow \left(\begin{array}{c} \alpha_i, \left(\boldsymbol{\omega}_i^r, \boldsymbol{\beta}_i^r\right)_{r\in[c]}, \left(\boldsymbol{\eta}_i^r, \boldsymbol{\gamma}_i^r\right)_{r\in[c]}, \left(U_{i,j}^{r,0}, U_{j,i}^{r,1}\right)_{\substack{j\neq i\\ r\in[c]}}, \left(V_{i,j}^{r,0}, V_{j,i}^{r,1}\right)_{\substack{j\neq i\\ r\in[c]}} \\ \left(C_{i,j}^{r,s,0}, C_{j,i}^{r,s,1}\right)_{\substack{j\neq i\\ r,s\in[c]}}, \left(W_{i,j,k}^{r,s,0}, W_{k,i,j}^{r,s,1}, W_{j,k,i}^{r,s,2}\right)_{\substack{(j,k)\neq(i,i),\\ r,s\in[c]}} \end{array}\right)$$

Eval: On input the seed κ_i, do the following:

1. For every $r \in [c]$, define the two polynomials

$$u_i^r(X) = \sum_{l\in[t]} \boldsymbol{\beta}_i^r[l] \cdot X^{\boldsymbol{\omega}_i^r[l]}, \quad v_i^r(X) = \sum_{l\in[t]} \boldsymbol{\gamma}_i^r[l] \cdot X^{\boldsymbol{\eta}_i^r[l]}$$

2. For every $r \in [c]$, compute

$$\widetilde{u}_i^r = \alpha_i \cdot u_i^r + \sum_{j\neq i} \left(\mathsf{DSPF}_{N,t}^2.\mathsf{FullEval}(U_{i,j}^{r,0}) + \mathsf{DSPF}_{N,t}^2.\mathsf{FullEval}(U_{j,i}^{r,1})\right)$$
$$\widetilde{v}_i^r = \alpha_i \cdot v_i^r + \sum_{j\neq i} \left(\mathsf{DSPF}_{N,t}^2.\mathsf{FullEval}(V_{i,j}^{r,0}) + \mathsf{DSPF}_{N,t}^2.\mathsf{FullEval}(V_{j,i}^{r,1})\right)$$

(viewing outputs of $\mathsf{FullEval}$ as degree $N-1$ polynomials over $\mathbb{F}$)

Fig. 3. $\mathbf{PCG}_{\text{triple}}$ - Part 1

3. For every $r, s \in [c]$, compute

$$w_i^{r,s} = u_i^r \cdot v_i^s + \sum_{j \neq i} \left(\mathsf{DSPF}^2_{2N,t^2}.\mathsf{FullEval}(C_{i,j}^{r,s,0}) + \mathsf{DSPF}^2_{2N,t^2}.\mathsf{FullEval}(C_{j,i}^{r,s,1}) \right)$$

4. For every $r, s \in [c]$, compute

$$\widetilde{w}_i^{r,s} = \sum_{\substack{j,k \\ (i,i) \neq (j,k)}} \Big(\mathsf{DSPF}^3_{2N,t^2}.\mathsf{FullEval}(W_{i,j,k}^{r,s,0}) + \mathsf{DSPF}^3_{2N,t^2}.\mathsf{FullEval}(W_{k,i,j}^{r,s,1}) + \mathsf{DSPF}^3_{2N,t^2}.\mathsf{FullEval}(W_{j,k,i}^{r,s,2}) \Big) + \alpha_i \cdot u_i^r(X) \cdot v_i^s(X)$$

5. Define the vectors of polynomials $\boldsymbol{u}_i = (u_i^1, \ldots, u_i^c)$, $\boldsymbol{v}^i = (v_i^1, \ldots, v_i^c)$, similarly for $\widetilde{\boldsymbol{u}}_i, \widetilde{\boldsymbol{v}}_i$.
   Let $\boldsymbol{w}_i = (w_i^{0,0}, \ldots, w_i^{c-1,0}, w_i^{0,1}, \ldots, w_i^{c-1,1}, \ldots, w_i^{c-1,c-1})$, and similarly define $\widetilde{\boldsymbol{w}}_i$.
6. Compute the final shares

$$x_i = \langle \boldsymbol{a}, \boldsymbol{u}_i \rangle, y_i = \langle \boldsymbol{a}, \boldsymbol{v}_i \rangle, z_i = \langle \boldsymbol{a} \otimes \boldsymbol{a}, \boldsymbol{w}_i \rangle \quad \text{and}$$
$$m_{x,i} = \langle \boldsymbol{a}, \widetilde{\boldsymbol{u}}_i \rangle, m_{y,i} = \langle \boldsymbol{a}, \widetilde{\boldsymbol{v}}_i \rangle, m_{z,i} = \langle \boldsymbol{a} \otimes \boldsymbol{a}, \widetilde{\boldsymbol{w}}_i \rangle$$

   in $\mathbb{F}_p[X]/\big(F(X)\big)$.
7. Output $(\alpha_i, x_i, y_i, z_i, m_{x,i}, m_{y,i}, m_{z,i})$.

Fig. 4. $\mathbf{PCG}_{\text{triple}}$ - Part 2

between the last two parties using a 2-party DPF. The shares of $\boldsymbol{Y}$ are regarded as vectors of seeds.

As we mentioned in Section 3, to derive the remaining seeds, we rely on oblivious transfer (OT). Observe that for every position $j \in [\sqrt{N}]$, the first party has to generate two random seeds. Moreover, for every j, the last two parties have to learn one of these seeds each. The discovered seeds coincide if and only if $j = \omega'$. We setup these seeds by running two sets of OTs, where the first party is sender in both, and the other two parties play receiver in one set each. The receivers' choice bits are determined based on the shares of $\boldsymbol{y}$, which are random bits that coincide if and only if $j = \omega'$.

Assuming the availability of a 3-party secret sharing of

$$\boldsymbol{v} = (\underbrace{0, 0, \ldots, 0, \overbrace{0}^{\omega''}, \beta, 0, 0, \ldots, 0}_{\sqrt{N}}),$$

the generation of the correction word is very simple: each party can just retrieve its share of $\boldsymbol{v}$ and add or subtract the expansions of its seeds. The correction

word is obtained by broadcasting and adding the results. Once we have the correction word, the DPF setup phase is complete. The only remaining question, then, is how to derive the secret-sharing of $\boldsymbol{v}$: since it is a unit-vector, we will use recursion.

We now discuss the protocol more in detail, including the details of recursion and active security. To simplify the presentation, we introduce some notation and building blocks.

Double exponential representation. We assume that N is a double exponential, that is, $N = 2^{2^h}$ for some $h \in \mathbb{N}$. In practice, this choice is rather restrictive as the value of N grows very quickly. However, we only make this assumption to simplify the description of a recursive step in our protocol, and this step can easily be adapted to the case $N = 2^m$ without significantly affecting the overall complexity[3].

We define the double exponential function $\mathrm{dE}(\cdot)$ as

$$\mathrm{dE}(k) := \begin{cases} 2 & \text{if } k = -1, \\ 2^{2^k} & \text{otherwise.} \end{cases}$$

We also use the following decomposition of integers, using a double exponential basis. Its proof can be found in Appendix C.

Lemma 1. *Any $\omega \in [N]$ can be written in a unique way as*

$$\omega = x(-1) + \sum_{i \in [h]} x(i) \cdot \mathrm{dE}(i)$$

for some $x(i) \in [\mathrm{dE}(i)]$ (depending on ω), for $i \in [h] \cup \{-1\}$, i.e. $0 \leq x(i) < 2^{2^i}$ if $i \in [h]$ and $x(-1) \in \{0, 1\}$.

Notation. Given a number $\omega \in [N]$, we denote its j-th bit by ω_j, whereas the j-th element of its double exponential notation is indicated by $\omega(j)$. Let $\mathcal{K} := [h] \cup \{-1\}$ and define

$$\mathcal{T} := \Big\{ (k, j) \mid k \in \mathcal{K}, j \in [\mathrm{dE}(k)] \Big\}.$$

In the protocol we use h PRGs. The k-th one will be $G_k : \{0,1\}^\lambda \longrightarrow \mathbb{F}^{\mathrm{dE}(k)}$. We will also rely on a tweakable correlation-robust hash function

$$H : \{0,1\}^\lambda \times \{0,1\}^* \longrightarrow \{0,1\}^\lambda.$$

An important fact is that the protocol requires the cardinality of the field $\mathbb{F}$ to be sufficiently close to 2^λ. More specifically, consider the map $\mathrm{Enc} : \{0,1\}^\lambda \longrightarrow \mathbb{F}$ sending every string $(x_0, x_1, \ldots, x_{\lambda-1})$ to $\sum_{i \in [\lambda]} x_i \cdot 2^i$. Let U be the uniform

[3] The protocol is more efficient when m is divisible by a power of 2.

distribution over $\{0,1\}^\lambda$ and let V be the uniform distribution over $\mathbb{F}$. In order to be secure, the protocol requires the statistical distance between V and $\mathrm{Enc}(U)$ to be negligible in the security parameter. It is possible to prove that this condition is satisfied if and only if $|p-2^\lambda|/2^\lambda$, where $p=|\mathbb{F}|$, is negligible in λ.

We also define a set of sign bits u_i^l with $i \in [3]$ and $l \in \{0,1\}$, by

$$u_i^l := \begin{cases} 1 & \text{if } i=1, \text{ or } i=2 \text{ and } l=0 \\ -1 & \text{otherwise.} \end{cases}$$

These parameters will indicate whether we need to add or subtract the expansion of the seeds in the 3-party DPF keys (see Section 3).

Finally, we define some matrices used to translate between different representations. In the protocol, we use the set of matrices $(B_k)_{k\in\mathcal{K}}$, which allow us to map an N-dimensional unit vector having special position $\omega \in [N]$ into a $\mathrm{dE}(k)$-dimensional unit vector with special position $\omega(k)$ and the same non-zero value. We also use a matrix $C \in \mathbb{F}_2^{\log(N)\times|\mathcal{T}|}$. This allows us to retrieve a binary representation of $\eta \in [N]$, given the unit-vector

$$(\overbrace{0,0,\ldots,0}^{\eta(k)},1,0,0,\ldots,0)\Big)\Big}_{\mathrm{dE}(k)}$$

for every $k \in \mathcal{K}$. The construction of the $(B_k)_k$ matrices is shown in Lemma 2, while C is in Lemma 3 (both in Appendix D).

5.1 Resources

The protocol we are going to present relies on an authenticated Random-OT functionality, which we instantiate using similar techniques to the TinyOT protocol [NNOB12]. We assume that every pair of parties (P_i, P_j) has access to an instance $\mathcal{F}^{i,j}_{\text{auth-ROT}}$ of this resource. The functionality $\mathcal{F}^{i,j}_{\text{auth-ROT}}$ provides Random-OT tuples, i.e. upon every call, P_i, the sender, obtains two random values $X_0, X_1 \in \{0,1\}^\lambda$, whereas P_j, the receiver, obtains a random choice bit b and the value X_b. Additionally, $\mathcal{F}^{i,j}_{\text{auth-ROT}}$ permits to perform linear operations on the choice bits it stored. The results of these computations are output to P_i and their correctness is guaranteed even when P_j is corrupted. Finally, the resource can output random bits to P_j. The latter can be used in combination with the choice bits in the computations. A formal description of $\mathcal{F}^{i,j}_{\text{auth-ROT}}$ can be found in Appendix F,where we also show how to implement it with low communication complexity using a Correlated-OT functionality.

In the protocol, we also use a black-box multiparty computation functionality $\mathcal{F}_{\text{MPC}}$ which allows n parties to perform computations over the prime field $\mathbb{F}$ and over $\mathbb{F}_2$. A complete description can be found in Figure 13 (see appendices).The functionality stores the inputs and results of the computations internally, providing the parties with handles. Each of the stored values is associated

with one of the domains $\mathbb{F}$ and $\mathbb{F}_2$ to which the element must belong. In the first case, the handle of x is denoted by $[[x]]$, whereas in the second case, the handle is denoted by $[[x]]_2$. Sometimes, we will abuse the notation and we will write $[[x]]_2$ even if $x \notin \{0,1\}$, in that case, it is understood that the functionality stored x bit by bit and the number of such bits depends only on the actual domain of x. The functionality $\mathcal{F}_{\text{MPC}}$ also features a 2-party DPF functionality, which, on input the indexes of two parties i, j, a value $[[\beta]]$ in $\mathbb{F}$ or $\mathbb{F}_{2^\lambda}$, a power of 2 M and $[[\omega]]_2 \in [M]$, outputs to P_i and P_j a 2-party secret-sharing of the M-dimensional unit vector having β in the ω-th position. The group on which the secret-sharing is defined coincides with the field to which β belongs.

Finally, we will use a functionality $\mathcal{F}_{\text{Rand}}$ which provides all the parties with random values sampled from the queried domains.

5.2 The Protocol

The functionality that our construction is going to implement is described in Figure 5. Observe that when the second and the third party are both corrupted the special position of the unit vector is leaked to the adversary. Since the protocol is based on the 3-party DPF described in Section 3, a leakage of this type was unavoidable. The functionality also allows the adversary to test the inputs in several occasions, every incorrect guess leading to an abort. In the triple generation protocol, the non-zero value β will be uniformly distributed in $\mathbb{F}^\times$, so any attempt of the adversary to guess it will fail with overwhelming probability. The leakage about the special position will not instead constitute a problem as it will be absorbed by the hardness of Module-LPN.

We can finally present our protocol. Its formal description can be found in Figures 6, 7 and 8.

Recursion. The protocol uses the 3-party DPF described in Figure 2 recursively in h levels indexed by $k = 0, 1, \ldots, h-1$. Once the k-th level is completed, the parties obtain a secret-sharing over $\mathbb{F}$ of the unit vector

$$\boldsymbol{v_{k+1}} := (\overbrace{0,0,\ldots,0}^{\widehat{\omega}(k+1)},\underbrace{\beta,0,0,\ldots,0}_{\mathrm{dE}(k+1)}),$$

$$\text{where} \qquad \widehat{\omega}(k+1) := \omega(-1) + \sum_{i=0}^{k} \omega(i) \cdot \mathrm{dE}(i).$$

Observe that $\widehat{\omega}(h) = \omega$.

More in detail, suppose that the parties possess a secret-sharing over $\mathbb{F}$ of $\boldsymbol{v_k}$. We aim to use it to securely generate 3-party DPF keys for the unit vector $\boldsymbol{v_{k+1}}$ (see Figure 2). Using the evaluation algorithm, the parties can then expand the keys to obtain a secret-sharing of $\boldsymbol{v_{k+1}}$.

$\mathcal{F}_{\text{3-DPF}}$

MPC Functionality. The functionality features the procedures Input, LinComb, Mult, Output, 2-DPF and Abort as in $\mathcal{F}_{\text{MPC}}$.

3-DPF. On input $[[\beta]] \in \mathbb{F}$, $[[\omega]]_2 \in [N]$ and three indexes $\sigma_0, \sigma_1, \sigma_2 \in [n]$:

1. If P_{σ_0}, P_{σ_1} and P_{σ_2} are all corrupted, send ω and β to the adversary. If only P_{σ_1} and P_{σ_2} are corrupted, send ω to the adversary.
2. If there exists $i \in [3]$ such that P_{σ_i} is corrupted, for three times, wait for $I \subseteq [N]$ from the adversary. If $\omega \notin I$, send ω to the adversary and abort.
3. If $\beta = 0$, output ZERO to the parties and to the adversary and stop.
4. If there exists $i \in [3]$ such that P_{σ_i} is corrupted, wait for the adversary.
   - If the latter sends $(\beta_\eta, \boldsymbol{v'_\eta})_{\eta \in [N]}$, check whether $\beta_\omega = \beta$. If this is the case, set $\widehat{\boldsymbol{v}} \leftarrow \boldsymbol{v'_\omega}$. Otherwise, send (ω, β) to the adversary and abort.
   - If the latter sends $(I''', \widehat{\boldsymbol{v}})$, check whether $\omega \in I'''$. If this is not the case, send ω to the adversary and abort.
5. If P_{σ_0}, P_{σ_1} and P_{σ_2} are all honest, set $\widehat{\boldsymbol{v}} \leftarrow \mathbf{0}$.
6. For every $\sigma_i \in \mathcal{H}$, sample a random $\boldsymbol{v_i} \in \mathbb{F}^N$ subject to
$$\sum_{\sigma_i \in \mathcal{H}} \boldsymbol{v_i} + \widehat{\boldsymbol{v}} = (\overbrace{0, 0, \ldots, 0}^{\omega \text{ elements}}, \beta, 0, 0, \ldots, 0)_{N \text{ elements}}.$$
7. Output $\boldsymbol{v_i}$ to party P_{σ_i} for every $\sigma_i \in \mathcal{H}$.

Fig. 5. The 3-party DPF functionality

Rearranging $\boldsymbol{v_{k+1}}$ into a matrix. First of all, observe that $\boldsymbol{v_{k+1}}$ is an $N_k := \mathrm{dE}(k+1)$-dimensional unit vector, whose special position is $\widehat{\omega}(k+1)$. Notice that

$$\widehat{\omega}(k+1) = \omega(k) \cdot \sqrt{N_k} + \widehat{\omega}(k) \quad \text{and} \quad 0 \leq \omega(k), \widehat{\omega}(k) < \mathrm{dE}(k) = \sqrt{N_k}.$$

In other words, when we rearrange $\boldsymbol{v_{k+1}}$ into a square matrix, following the procedure described in Section 3, the special position ends up at the intersection between the $\omega(k)$-th row and the $\widehat{\omega}(k)$-th column. Observe that it is easy to obtain a secret-sharing of $\omega(k)$ over $\mathbb{F}_2$ given a secret-sharing of ω over $\mathbb{F}_2$. Indeed, $\omega(k)$ is described by a 2^k-bit substring of the bit representation of ω.

The vectors $\boldsymbol{y_0}$, $\boldsymbol{y_1}$ and $\boldsymbol{y_2}$. Following the blueprint of the 3-party DPF described in Section 3, the first ingredient needed to generate $\boldsymbol{v_{k+1}}$ is the vectors $\boldsymbol{y_{k,1}}$ and $\boldsymbol{y_{k,2}}$, i.e. a secret-sharing over $\mathbb{F}$ of

$$\boldsymbol{y^k} := (\overbrace{0, 0, \ldots, 0}^{\omega(k)}, 1, 0, 0, \ldots, 0)_{\mathrm{dE}(k)} = \boldsymbol{y_{k,1}} + \boldsymbol{y_{k,2}}.$$

$\Pi_{\text{3-DPF}}$

The environment has access to the the procedures Input, LinComb, Mult, Output and 2-DPF of $\mathcal{F}_{\text{MPC}}$.

3-DPF. On input a number $[[\omega]]_2 \in [N]$, $[[\beta]] \in \mathbb{F}$ and indexes $\sigma_0, \sigma_1, \sigma_2 \in [n]$:

Generation of $\boldsymbol{y_{k,0}}$, $\boldsymbol{y_{k,1}}$ and $\boldsymbol{y_{k,2}}$.

1. The parties call 2-DPF over $\mathbb{F}$ with special position $[[\omega]]_2$, non-zero element 1 and indexes σ_1 and σ_2. If the latter aborts, the parties abort. Let $\boldsymbol{y_i}$ be the output received by P_{σ_i} for $i \in \{1,2\}$.
2. For every $k \in \mathcal{K}$ and $i \in \{1,2\}$, P_{σ_i} computes $\boldsymbol{y_{k,i}} \leftarrow B_k \cdot \boldsymbol{y_i}$. P_{σ_0} sets $\boldsymbol{y_{k,0}} \leftarrow \boldsymbol{0}$.

From $\mathbb{F}$ secret-sharing to binary secret-sharing.

3. If there exists $(k,j) \in \mathcal{T}$ and $i \in \{1,2\}$ such that $y^j_{k,i} = 0$, P_{σ_i} aborts.
4. For every $k \in \mathcal{K}$ and $i \in \{1,2\}$, P_{σ_i} sets $\boldsymbol{b_{k,i}} \leftarrow \boldsymbol{y_{k,i}} \bmod 2$.

Seed generation - Part 1.

5. The parties generate $[[\Delta]]_2 \leftarrow \mathsf{Rand}(\mathbb{F}_{2^\lambda})$.
6. The parties call 2-DPF over $\mathbb{F}_{2^\lambda}$ with special position $[[\omega]]_2$, non-zero element $[[\Delta]]_2$ and indexes σ_1 and σ_2. If the latter aborts, the parties abort. Let $\boldsymbol{T_i}$ be the output received by P_{σ_i} for $i \in \{1,2\}$.
7. For every $k \in \mathcal{K}$ and $i \in \{1,2\}$, P_{σ_i} computes $\boldsymbol{T_{k,i}} \leftarrow B_k \cdot \boldsymbol{T_i}$.
8. For every $(k,j) \in \mathcal{T}$ and $i \in \{1,2\}$, P_{σ_i} computes $Y^j_{k,i} \leftarrow H\big(T^j_{k,i}, (k,j)\big)$.

Seed generation - Part 2.

9. For ever $(k,j) \in \mathcal{T}$:
   (a) P_{σ_0} and P_{σ_1} call $\mathcal{F}^{\sigma_0,\sigma_1}_{\text{auth-ROT}}$ with P_{σ_0} as sender. P_{σ_0} obtains $(X^j_k[0], X^j_k[1]) \in \mathbb{F}_{2^\lambda} \times \mathbb{F}_{2^\lambda}$, P_{σ_1} receives $(t^j_{k,1}, X^j_k[2]) \in \mathbb{F}_2 \times \mathbb{F}_{2^\lambda}$, where $X^j_k[2] = X^j_k[t^j_{k,1}]$.
   (b) P_{σ_1} sends $w^j_{k,1} \leftarrow b^j_{k,1} \oplus t^j_{k,1}$ to P_{σ_0}.
   (c) P_{σ_0} and P_{σ_2} call $\mathcal{F}^{\sigma_0,\sigma_2}_{\text{auth-ROT}}$ with P_{σ_0} as sender. P_{σ_0} obtains $(W^j_k[0], W^j_k[1]) \in \mathbb{F}_{2^\lambda} \times \mathbb{F}_{2^\lambda}$, P_{σ_2} receives $(t^j_{k,2}, W^j_k) \in \mathbb{F}_2 \times \mathbb{F}_{2^\lambda}$, where $W^j_k = W^j_k[t^j_{k,2}]$.
   (d) P_{σ_2} sends $w^j_{k,2} \leftarrow b^j_{k,2} \oplus t^j_{k,2}$ to P_{σ_0}.
   (e) P_{σ_0} sends to P_{σ_2}

$$Z^j_k[0] \leftarrow W^j_k[w^j_{k,2}] \oplus X^j_k[w^j_{k,1}], \qquad Z^j_k[1] \leftarrow W^j_k[w^j_{k,2} \oplus 1] \oplus X^j_k[w^j_{k,1} \oplus 1].$$

   (f) P_{σ_2} computes $X^j_k[3] \leftarrow W^j_k \oplus Z^j_k[b^j_{k,2}]$
10. For every $(k,j) \in \mathcal{T}$
    - P_{σ_0} sets $S^j_{k,0,0} \leftarrow X^j_k[0]$ and $S^j_{k,0,1} \leftarrow X^j_k[1]$.
    - P_{σ_1} sets $S^j_{k,1,0} \leftarrow X^j_k[2]$ and $S^j_{k,1,1} \leftarrow Y^j_{k,1}$.
    - P_{σ_2} sets $S^j_{k,2,0} \leftarrow X^j_k[3]$ and $S^j_{k,2,1} \leftarrow Y^j_{k,2}$.

Fig. 6. $\Pi_{\text{3-DPF}}$ - Part 1

First check.

11. For every $i \in \{1,2\}$ and $k \in \mathcal{K}$, P_{σ_0} and P_{σ_i} call $\mathsf{LinearCombination}$ in $\mathcal{F}_{\text{auth-ROT}}^{\sigma_0,\sigma_i}$ to compute
$$t_{k,i} \leftarrow \bigoplus_{j \in [\mathrm{dE}(k)]} [[t_{k,i}^j]]_2.$$
12. For every $k \in \mathcal{K}$, P_{σ_0} computes
$$\psi_k \leftarrow t_{k,1} \oplus t_{k,2} \oplus \bigoplus_{j \in [\mathrm{dE}(k)]} (w_{k,1}^j \oplus w_{k,2}^j).$$
If any of them is different from 1, it makes the protocol abort.

Second check.

13. For every $i \in \{1,2\}$, P_{σ_0} and P_{σ_i} call Random in $\mathcal{F}_{\text{auth-ROT}}^{\sigma_0,\sigma_i}$ for λ times. Let $R_i \in \{0,1\}^\lambda$ be the binary string obtained by P_{σ_i}.
14. For every $i \in \{1,2\}$, P_i inputs R_i into $\mathcal{F}_{\text{MPC}}$ with domain $\mathbb{F}_2$.
15. The parties call $\mathcal{F}_{\text{Rand}}$ to obtain a random matrix $V \in \mathbb{F}_2^{\lambda \times \log(N)}$
16. For every $i \in \{1,2\}$, P_{σ_0} and P_{σ_i} call $\mathsf{LinearCombination}$ in $\mathcal{F}_{\text{auth-ROT}}^{\sigma_0,\sigma_i}$ to compute $\Phi_i \leftarrow [[R_i]]_2 \oplus V \cdot C \cdot [[\boldsymbol{t_i}]]_2$ where $\boldsymbol{t_i}$ is the $|\mathcal{T}|$-dimensional vector having $t_{k,i}^j$ in the $(\mathrm{dE}(k)+j)$-th position.
17. P_{σ_0} computes $\Phi \leftarrow \Phi_1 \oplus \Phi_2 \oplus V \cdot C \cdot (\boldsymbol{w} \oplus \boldsymbol{1})$ where $\boldsymbol{w} \oplus \boldsymbol{1}$ is the $|\mathcal{T}|$-dimensional vector having $w_{k,1}^j \oplus w_{k,2}^j \oplus 1$ in the $(\mathrm{dE}(k)+j)$-th position.
18. Using $\mathcal{F}_{\text{MPC}}$ the parties open $\Phi' \leftarrow [[R_1]]_2 \oplus [[R_2]]_2 \oplus V \cdot [[\omega]]_2$. If $\Phi \neq \Phi'$, P_{σ_0} makes the protocol abort.

Base case.

19. For every $j \in [2]$ the parties set
$$x_0^j \leftarrow \mathrm{Enc}(X_{-1}^j[0]), \quad x_1^j \leftarrow \mathrm{Enc}(X_{-1}^j[1]), \quad x_2^j \leftarrow \mathrm{Enc}(X_{-1}^j[2]),$$
$$x_3^j \leftarrow \mathrm{Enc}(X_{-1}^j[3]), \quad x_4^j \leftarrow \mathrm{Enc}(Y_{-1,1}^j), \quad x_5^j \leftarrow \mathrm{Enc}(Y_{-1,2}^j).$$
P_{σ_0} sets $s_0^j \leftarrow -x_0^j - x_1^j$. P_{σ_1} sets $s_1^j \leftarrow x_2^j + x_4^j$. P_{σ_2} sets $s_2^j \leftarrow x_3^j - x_5^j$. Then, for each $i \in [3]$, P_{σ_i} sets $z_i \leftarrow s_i^0 \oplus s_i^1$.
20. The parties perform the following operations
$$[[z_i]] \leftarrow \mathsf{Input}(P_{\sigma_i}, z_i) \quad \forall i \in [3]$$
$$[[CW]] \leftarrow ([[z_0]] + [[z_1]] + [[z_2]])^{-1} \cdot [[\beta]]$$
$$CW \leftarrow \mathsf{Output}([[CW]])$$
If $CW = 0$, the parties stop and output ZERO. If the operation cannot be performed due to a zero denominator, all the parties stop and output $\perp$.
21. Each party P_{σ_i} sets $v_{0,i}^j \leftarrow s_i^j \cdot CW$ for $j \in \{0,1\}$. Let $\boldsymbol{v_{0,i}} := (v_{0,i}^0, v_{0,i}^1)$ for every $i \in [3]$.

Fig. 7. $\Pi_{\text{3-DPF}}$ - Part 2

Generation of the correction words.

22. For each $k \in [h]$ the parties compute the following operations
    (a) For every $i \in [3]$, P_{σ_i} broadcasts

$$\boldsymbol{CW_{k,i}} \leftarrow \boldsymbol{v_{k,i}} - \sum_{j \in [\mathrm{dE}(k)]} u_i^0 \cdot G_k(S^j_{k,i,0}) - \sum_{j \in [\mathrm{dE}(k)]} u_i^1 \cdot G_k(S^j_{k,i,1}).$$

    (b) The parties set $\boldsymbol{CW_k} \leftarrow \boldsymbol{CW_{k,0}} + \boldsymbol{CW_{k,1}} + \boldsymbol{CW_{k,2}}$.
    (c) Each party P_{σ_i} sets for every $j \in [\mathrm{dE}(k)]$

$$\boldsymbol{v^j_{k+1,i}} \leftarrow u_i^0 \cdot G_k(S^j_{k,i,0}) + u_i^1 \cdot G_k(S^j_{k,i,1}) + y^j_{k,i} \cdot \boldsymbol{CW_k}.$$

    Let $\boldsymbol{v_{k+1,i}} := (\boldsymbol{v^0_{k+1,i}} \parallel \boldsymbol{v^1_{k+1,i}} \parallel \ldots \parallel \boldsymbol{v^{\mathbf{dE(k)-1}}_{k+1,i}})$.

Final check.

23. The parties call $\mathcal{F}_{\mathrm{Rand}}$ to sample $\boldsymbol{\chi} = (\chi_0, \chi_1, \ldots, \chi_{N-1}) \in \mathbb{F}^N$.
24. Perform the following operations
    (a) $[[d_i]] \leftarrow \mathsf{Input}(P_{\sigma_i}, \langle \boldsymbol{\chi}, \boldsymbol{y_i} \rangle)$ for each $i \in \{1, 2\}$.
    (b) $[[\zeta_i]] \leftarrow \mathsf{Input}(P_{\sigma_i}, \langle \boldsymbol{\chi}, \boldsymbol{v_{h,i}} \rangle)$ for each $i \in [3]$.
    (c) $[[\rho]] \leftarrow [[\zeta_0]] + [[\zeta_1]] + [[\zeta_2]] - ([[d_1]] + [[d_2]]) \cdot [[\beta]]$
    (d) $\rho \leftarrow \mathsf{Output}([[\rho]])$
    (e) If $\rho \neq 0$, P_{σ_i} outputs ABORT and stops. Otherwise, P_{σ_i} outputs $\boldsymbol{v_{h,i}}$.

Fig. 8. $\Pi_{\text{3-DPF}}$ - Part 3

At the beginning of our protocol, using the 2-party DPF procedure in $\mathcal{F}_{\mathrm{MPC}}$, the second and third party obtain a secret-sharing of the unit vector

$$\boldsymbol{y} := (\underbrace{\overbrace{0, 0, \ldots, 0}^{\omega}, 1, 0, 0, \ldots, 0}_{N}).$$

By locally applying the matrix B_k on the shares, this also gives the shares $\boldsymbol{y_{k,1}}$ and $\boldsymbol{y_{k,2}}$. We recall that B_k maps an N-dimensional unit vector having special position $\omega \in [N]$ into a $\mathrm{dE}(k)$-dimensional unit vector with special position $\omega(k)$ and the same non-zero value.

From $\mathbb{F}$-secret-sharing to binary secret-sharing. In the previous paragraph, we described how it is possible to obtain a 2-party secret-sharing over $\mathbb{F}$ of the unit vector $\boldsymbol{y^k}$. In order to securely generate the seeds in the DPF key, we will need to convert this to a 2-party secret-sharing over the binary field $\mathbb{F}_2$. Using a standard trick , we can do this conversion without any interaction.

Recall that $|\mathbb{F}| = p$ for a large prime p. Suppose that the two parties have shares $b_1, b_2 \in [p]$, where $b_1 + b_2 \equiv b \bmod p$, for some $b \in \{0, 1\}$, as we do for each entry of $\boldsymbol{y^k}$. If the shares are random, then with overwhelming probability both of them are non-zero, so over the integers, $2 \leq b_1 + b_2 < 2p$ and therefore $b_1 +$

$b_2 = b + p$. Reducing both sides modulo 2, we get that $(b_1 \bmod 2) \oplus (b_2 \bmod 2) = b \oplus 1$. In other words, for every $j \in [\mathrm{dE}(k)]$, the second and the third parties can obtain bits $b^j_{k,1} := y^j_{k,1} \bmod 2$ and $b^j_{k,2} := y^j_{k,2} \bmod 2$ that coincide if and only if $j = \omega(k)$.

This procedure works only if both b_1 and b_2 are non-zero, and for that reason, the parties abort if this is not satisfied. When the second and the third party are both honest, the property condition holds with overwhelming probability. If one of the parties is corrupted, however, $\mathcal{F}_{\mathrm{MPC}}$ allows the adversary to choose its shares. An attacker can exploit this fact to retrieve information about ω, indeed, it can select its shares so that the protocol aborts only if ω assumes particular values (selective failure attack). The corresponding leakage is modelled in step 2 of $\mathcal{F}_{\text{3-DPF}}$ (see Figure 5).

The seed generation - Part 1. We now turn to the task of generating the PRG seeds used in the DPF. We start with the method for obtaining the last seeds of the second and the third party, which, following the idea from Section 3, we compress using a 2-party DPF. Recall that these seeds coincide for every position $j \neq \omega(k)$, whereas, when $j = \omega(k)$, they are independent. Using the 2-party DPF command of $\mathcal{F}_{\mathrm{MPC}}$, we can obtain a 2-party secret-sharing over $\mathbb{F}_{2^\lambda}$ of

$$\underbrace{(\overbrace{0, 0, \ldots, 0}^{\omega}, \Delta, 0, 0, \ldots, 0)}_{N}.$$

where Δ is sampled randomly by $\mathcal{F}_{\mathrm{MPC}}$. Then, by applying the matrix B_k, this can be translated into shares of

$$\underbrace{(\overbrace{0, 0, \ldots, 0}^{\omega(k)}, \Delta, 0, 0, \ldots, 0)}_{\mathrm{dE}(k)}.$$

for any $k \in \mathcal{K}$. The only problem is that, in this way, the entries of the shares in the special position are not independent, due to the fixed correlation Δ. Therefore, to turn these shares into independent, random seeds, we apply the correlation-robust hash function H to each entry.

The seed generation - Part 2. Generating and distributing the remaining seeds is more complex. We have previously explained how the second and third parties derive, for each $j \in [\mathrm{dE}(k)]$, bits $b^j_{k,1}$ and $b^j_{k,2}$ that coincide if and only if $j = \omega(k)$. Now, for every $j \in [\mathrm{dE}(k)]$, the second and the third party must learn one of the seeds of the first party. The discovered seeds will coincide if and only if $j = \omega(k)$. We can therefore generate and distribute the remaining seeds using oblivious transfer (OT). Specifically, for every $j \in [\mathrm{dE}(k)]$, the first and the second party can obtain their missing seeds by means of a "sender-random" OT, i.e. an OT where the sender's messages, corresponding to the seeds of the first party, are random, while the receiver can choose its input. The first party will be the sender, while the second party will be the receiver with choice bit $b^j_{k,1}$. The

third party can then receive its missing seed by means of another, now standard, OT. The sender, corresponding to the first party, will choose its messages to be the same as in the "sender-random" OT, while the choice bit of the receiver, the third party, will be $b^j_{k,2}$. The two OTs are implemented using the random-OT functionality $\mathcal{F}_{\text{auth-ROT}}$. Note that this functionality ensures that the choice bits are authenticated, which we rely on later, to check consistency of this stage and achieve active security.

The correction word (Fig. 8). After obtaining the seeds, the only missing piece of the DPF key is the correction word. Computing it is rather straightforward as each party can just retrieve its share of $\boldsymbol{v_k}$ and add or subtract the expansions of its seeds using G_k[4]. The correction word is obtained by broadcasting and adding the results. Observe that if recursion had not been used, at this point of the protocol, the parties would have needed to generate a secret-sharing of the $\sqrt{N}$-dimensional vector $\boldsymbol{v_{h-1}}$. Direct approaches would have required $O(\sqrt{N})$ communication, recursion instead allows us to compute that with $O(\sqrt[4]{N})$ complexity.

The base case $k = 0$. We have explained how to derive a secret-sharing of $\boldsymbol{v_{k+1}}$ given a secret-sharing of $\boldsymbol{v_k}$. It remains to describe how to deal with the base case, i.e. how to derive a secret-sharing of $\boldsymbol{v_0}$. Observe that $\boldsymbol{v_0}$ is a 2-dimensional unit vector, where β occupies the $\omega(-1)$-th position.

By using the same procedure described in the seed generation, for each position of $\boldsymbol{v_0}$, the parties can obtain pairs of elements in $\{0,1\}^\lambda$ of the form

$$\begin{aligned}
&\textbf{if } \boldsymbol{j \neq \omega(-1)}: && \{a^j_{-1}, b^j_{-1}\}, && (b^j_{-1}, c^j_{-1}), && (a^j_{-1}, c^j_{-1}),\\
&\textbf{if } \boldsymbol{j = \omega(-1)}: && \{a^j_{-1}, d^j_{-1}\}, && (d^j_{-1}, b^j_{-1}), && (d^j_{-1}, c^j_{-1}).
\end{aligned}$$

This time, we do not regard them as seeds anymore, but using the encoding map Enc, we convert them into random elements in the field $\mathbb{F}$. Observe that by combining the elements with the coefficients u^b_i, we can derive a secret-sharing of zero when $j \neq \omega(-1)$ and a secret-sharing of a random value $z \in \mathbb{F}$ when $j = \omega(-1)$. Obtaining a secret-sharing of $\boldsymbol{v_0}$ is now easy, we simply need to multiply each secret-sharing we have just computed by $\beta \cdot z^{-1}$. The operation can be performed using $\mathcal{F}_{\text{MPC}}$.

Achieving active security. The protocol we just described allows the adversary to deviate in several points. In order to regain control on the execution, we relied on three different checks. Only the combination of all of them guarantees security.

The first issue we encounter is in the seed generation. An adversary corrupting both the second and third party can indeed discover all the seeds of the first party by always inputting different choice bits in the OTs. With this attack, the adversary would be able to retrieve β once the correction word is computed. So, we designed the first check to fail in these situations. Specifically, using

[4] Whether we need to add or subtract is specified by the sign multipliers u^b_i.

$\mathcal{F}_{\text{auth-ROT}}$, the check recomputes the sum of the OT bits input by the receivers for every recursion level k. If the result is different from 1, the protocol aborts.

When the second or third party are corrupted, by cleverly choosing the choice bits of the OTs, the adversary can move the non-zero value β to a different position $\eta \neq \omega$. The second check makes sure that this attack fails with overwhelming probability. This is achieved by recomputing η from the OT inputs using the matrix C and $\mathcal{F}_{\text{auth-ROT}}$. The result is obliviously compared to ω using $\mathcal{F}_{\text{MPC}}$, the protocol aborts when they do not match.

The third check, inspired by [YWL+20], is probably the most important. Essentially, it draws a random N-dimensional vector $\boldsymbol{\chi} \in \mathbb{F}^N$ and checks that the result of the linear combination $\langle \boldsymbol{\chi}, \boldsymbol{v_h} \rangle$ coincides with $\chi_\omega \cdot \beta$. The procedure counteracts any malicious behaviour in the generation of the correction words. Moreover, in combination with the first check, it makes sure that, for every level k, there exists only one position for which the choice bits of the OTs coincide. On the other hand, the third check causes some leakage which is modelled in step 4 of $\mathcal{F}_{\text{3-DPF}}$ (see Figure 5). We prove the following in Appendix D.

Theorem 3. *Let $N = \text{dE}(h)$ be a double power of 2 and assume that $\mathbb{F}$ is a security-parameter-dependent prime field of cardinality p such that $|p - 2^\lambda|/2^\lambda$ is negligible in λ. Let $G_k : \{0,1\}^\lambda \longrightarrow \mathbb{F}^{\text{dE}(k)}$ be a PRG for every $k \in [h]$ and let $H : \{0,1\}^\lambda \times \{0,1\}^* \longrightarrow \{0,1\}^\lambda$ be a tweakable correlation-robust hash function. Then the protocol $\Pi_{\text{3-DPF}}$ UC-realises $\mathcal{F}_{\text{3-DPF}}$ in the $(\mathcal{F}_{\text{MPC}}, \mathcal{F}_{\text{auth-ROT}}, \mathcal{F}_{\text{Rand}})$-hybrid model. Moreover, if all the parties are honest, $\Pi_{\text{3-DPF}}$ aborts with negligible probability.*

Complexity. The protocol $\Pi_{\text{3-DPF}}$ achieves low communication complexity. As a matter of fact, in [BCG+20], the authors described how to implement the 2-party DPF procedure of $\mathcal{F}_{\text{MPC}}$ with $O\big(\log(N) \cdot \mathsf{poly}(\lambda)\big)$ communication. We also observe that the seed generation needs $O(\sqrt{N})$ OTs. Hence, $\Pi_{\text{3-DPF}}$ has $O\big(\sqrt{N} \cdot \mathsf{poly}(\lambda)\big)$ communication complexity. A more detailed analysis of efficiency can be found in Section 7.

6 Offline Phase

We can finally describe our Offline phase protocol Π_{Offline}, which achieves sublinear communication complexity. It can be broken down into 3 procedures: an initialisation procedure in which the MAC key α is generated and secret-shared, a triple generation procedure and an input mask generation procedure. The latter is used to produce, for every $j \in [n]$, random authenticated secret-shared elements $[\![a_j]\!]$ whose value is known only to party P_j. As for multiplication triples, input masks constitute an essential part of SPDZ as they are needed to provide the inputs of the computation.

The protocol Π_{Offline} closely resembles $\mathsf{PCG}_{\text{triple}}$. We start by summarising its working and later, we will provide a more formal description. The protocol permits to generate N multiplication triples with $O\big(\sqrt{N} \cdot \mathsf{poly}(\lambda)\big)$ communication complexity and N input masks with $O\big(\log(N) \cdot \mathsf{poly}(\lambda)\big)$ communication

complexity. The bottleneck of the triple generation is the 3-party DPF. If future research proves the existence of 3-party DPFs with logarithmic key size, we will probably be able to design multiparty triple generation protocols with logarithmic communication complexity.

Multiplication triples. The protocol uses the functionality $\mathcal{F}_{\text{3-DPF}}$ as a black box. During the initialisation procedure, each party P_i samples a random share α_i for the MAC key and inputs it in $\mathcal{F}_{\text{3-DPF}}$, fundamentally committing to its choice.

The multiplication triples are derived by executing the seed generation and the evaluation of $\mathsf{PCG}_{\text{triple}}$ inside $\mathcal{F}_{\text{3-DPF}}$: at the very beginning, each party P_i samples random special positions $\boldsymbol{\omega_i^r}, \boldsymbol{\eta_i^r} \in [N]^t$ and random non-zero elements $\boldsymbol{\beta_i^r}, \boldsymbol{\gamma_i^r} \in \mathbb{F}^t$ for every $r \in [c]$. These values are input in $\mathcal{F}_{\text{3-DPF}}$. Using 2-DPF and 3-DPF, it is then possible for P_i to obtain $\boldsymbol{u_i}, \widetilde{\boldsymbol{u}}_{\boldsymbol{i}}, \boldsymbol{v_i}, \widetilde{\boldsymbol{v}}_{\boldsymbol{i}}, \boldsymbol{w_i}$ and $\widetilde{\boldsymbol{w}}_{\boldsymbol{i}}$. Finally, by sampling a random $\boldsymbol{a} \in R^c$ using $\mathcal{F}_{\text{Rand}}$, the parties can compute the final output, i.e. random authenticated secret-shared elements $[\![x]\!], [\![y]\!], [\![z]\!] \in R$ such that $z = x \cdot y$. We recall that $R = \mathbb{F}[X]/\big(F(X)\big)$ where $F(X)$ is a degree-N polynomial. If $F(X)$ has N distinct roots in $\mathbb{F}$, the tuple $\big([\![x]\!], [\![y]\!], [\![z]\!]\big)$ can be converted into N random multiplication triples by evaluating the shares[5] over the roots of $F(X)$.

Input Masks. The generation of inputs masks is very similar but simpler. At the beginning, the party P_j to which the masks are addressed samples random special positions $\boldsymbol{\omega^r} \in [N]^t$ and random non-zero elements $\boldsymbol{\beta^r} \in \mathbb{F}^t$ for every $r \in [c]$, inputting them in $\mathcal{F}_{\text{3-DPF}}$. These values will be used to define the sparse polynomial

$$u^r(X) \leftarrow \sum_{l \in [t]} \beta^r[l] \cdot X^{\omega^r[l]}$$

Later on, for every $i \neq j$, P_i and P_j can obtain a secret-sharing of $\alpha_i \cdot u^r(X)$ using 2-DPF. Finally, by sampling a random $\boldsymbol{a} \in R^c$ using $\mathcal{F}_{\text{Rand}}$ and relying on the hardness of Module LPN, the shares can be converted into a random authenticated secret-shared element $[\![x]\!] \in R$. Since P_j knows $u^r(X)$ for every $r \in [c]$, it can also learn x. As a last operation, the shares are rerandomised using a PRG. Observe that from $[\![x]\!]$, we can derive N masks using the same trick described for the multiplication triples.

Leakage. The main difference between Π_{Offline} and $\mathsf{PCG}_{\text{triple}}$ is that every execution of 2-DPF and 3-DPF has additional leakage. At first, it might seem that the main issue arises when the last two players P_j and P_k of the 3-party DPF procedure are corrupted. In such cases, the special positions $(\boldsymbol{\omega_j^r} \boxplus \boldsymbol{\eta_k^s})_{r,s \in [c]}$ are indeed revealed to the adversary. Notice, however, that this is no problem at all, as the leaked values were chosen by the adversary itself at the beginning of the protocol.

Regarding the remaining leakage, observe that when the adversary tries the guess any non-zero element during 3-DPF, the procedure aborts with overwhelming probability. Indeed, the non-zero values are uniformly distributed in $\mathbb{F}^\times$, assuming that at least one party involved in 3-DPF is honest. Moreover, any leak-

[5] The shares are elements of R and therefore polynomials

age concerning the special positions is absorbed by the hardness of module-LPN with static leakage.

6.1 A Formal Description of the Offline Phase

Now, we formally describe the Offline phase protocol Π_{Offline}. The corresponding functionality is a typical Offline-phase functionality and it is described in Figure 9, Π_{Offline} is instead formalised in Figures 10, 11 and 12. However, first, we have to clarify the notation used in the resource $\mathcal{F}_{\text{3-DPF}}$.

$\mathcal{F}_{\text{Offline}}$

MPC Functionality. The functionality features the procedures Input, LinComb, Mult, Output and Abort as in $\mathcal{F}_{\text{MPC}}$ as well as the following operations.

Initialisation. The functionality samples $\alpha \xleftarrow{\$} \mathbb{F}$ and waits for $(\alpha_i)_{i\in\mathcal{C}}$ from the corrupted parties. Then, it samples a random α_i for every $i \in \mathcal{H}$ subject to $\sum_{i\in[n]} \alpha_i = \alpha$. Finally, it outputs α_i to every honest party P_i.

Input Masks. On input (Mask, P_j) from every party, the functionality waits for $x \in R$ from the adversary if P_j is corrupted. Otherwise, it samples $x \xleftarrow{\$} R$. The functionality waits for a pair $(\widehat{x}, \widehat{m}_x)$ from the adversary. Finally, it samples a random pair $(x_i, m_{x,i})$ for every $i \in \mathcal{H}$ subject to

$$\sum_{i\in\mathcal{H}} x_i + \widehat{x} = x, \qquad \sum_{i\in\mathcal{H}} m_{x,i} + \widehat{m}_x = \alpha \cdot x.$$

At the end, it outputs $(x_i, m_{x,i})$ to every honest party P_i.

Triple Generation. On input (Triple) from every party, the functionality samples $x, y \xleftarrow{\$} R$ and sets $z \leftarrow x \cdot y$. Then, it waits for a tuple $(\widehat{x}, \widehat{m}_x, \widehat{y}, \widehat{m}_y, \widehat{z}, \widehat{m}_z)$ from the adversary. Finally, it samples a random tuple $(x_i, m_{x,i}, y_i, m_{y,i}, z_i, m_{z,i})$ for every $i \in \mathcal{H}$ subject to

$$\sum_{i\in\mathcal{H}} x_i + \widehat{x} = x, \qquad \sum_{i\in\mathcal{H}} y_i + \widehat{y} = y, \qquad \sum_{i\in\mathcal{H}} z_i + \widehat{z} = z,$$
$$\sum_{i\in\mathcal{H}} m_{x,i} + \widehat{m}_x = \alpha \cdot x, \quad \sum_{i\in\mathcal{H}} m_{y,i} + \widehat{m}_y = \alpha \cdot y, \quad \sum_{i\in\mathcal{H}} m_{z,i} + \widehat{m}_z = \alpha \cdot z.$$

At the end, it outputs $(x_i, m_{x,i}, y_i, m_{y,i}, z_i, m_{z,i})$ to every honest party P_i.

Fig. 9. The offline phase functionality

Resources. Observe that in $\mathsf{PCG}_{\text{triple}}$, many operations are performed on vectors, repeating them entry by entry. In order to not overload the notation with indexes, we decided to abuse the notation in $\mathcal{F}_{\text{3-DPF}}$ to include secret-sharings of vectors. Specifically, a vector of secret-shared elements $([[v_0]], [[v_1]], \ldots, [[v_{l-1}]])$ will be denoted by $[[\boldsymbol{v}]]$, in a similar way, a vector $([[u_0]]_2, [[u_1]]_2, \ldots, [[u_{l-1}]]_2)$

will be denoted by $[[\boldsymbol{u}]]_2$. Recall that in the second case, an entry u_i might not be a bit. Indeed, $[[u_i]]_2$ means that u_i is stored in $\mathcal{F}_{\text{3-DPF}}$ bit by bit.

Operations between vectors. When we perform operations over secret-shared vectors, e.g. $[[\boldsymbol{v}]] + [[\boldsymbol{w}]]$, it is understood that the operation is performed entry-wise. Observe that the outer product $\otimes$ can easily be computed in MPC by performing multiplications between the entries of the two secret-shared vectors. We also assume that $\mathcal{F}_{\text{3-DPF}}$ features a procedure IntAdd, which, on input two $\mathbb{F}_2$-secret-shared integers $[[a]]_2, [[b]]_2 \in [N]$, outputs an $\mathbb{F}_2$-secret-sharing of their addition $[[a+b]]_2$ over $\mathbb{Z}$. Observe that the procedure can be implemented using a ripple-carry adder. The operation requires only the computation of XORs (no communication) and $\log(N)$ ANDs ($2n$ bits of communication each). Using IntAdd, we can easily compute the outer sum $\boxplus$. We recall that the outer sum of t-dimensional integer vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ is a t^2-dimensional integer vector whose $(it+j)$-th entry is $a_i + b_j$.

Sum of point functions. Finally, we also assume that $\mathcal{F}_{\text{3-DPF}}$ features two sum-of-point-function procedures 2-DSPF and 3-DSPF. In both cases, the output is a secret-shared multi-point vector over the field $\mathbb{F}$, a 2-party secret-sharing for 2-DSPF and a 3-party secret-sharing for 3-DSPF. The inputs are the length of the output, the parties among which the result is shared, and two same-dimensional vectors of secret-shared elements, the first of which indicating the special positions, the second one representing the non-zero elements. Observe that we can easily implement 2-DSPF and 3-DSPF by running 2-DPF and 3-DPF multiple times and outputting the sum of the shares. Therefore, the functionality $\mathcal{F}_{\text{3-DPF}}$ will model leakage and influence of the adversary accordingly. To summarise, 2-DSPF and 3-DSPF can be regarded as shorthands for multiple executions of 2-DPF and 3-DPF respectively and the inputs will be given in vectorised form.

Theorem 4. *Let $F(X)$ be a degree-N polynomial over the prime field $\mathbb{F}$ and let $t, c \in \mathbb{N}$. Define the ring $R := \mathbb{F}[X]/\big(F(X)\big)$ and let $G : \{0,1\}^\lambda \longrightarrow R^2$ be a PRG. If the R^c-LPN$_t$ problem with static leakage is hard, then the protocol $\Pi_{Offline}$ UC-implements $\mathcal{F}_{Offline}$ in the $(\mathcal{F}_{3\text{-}DPF}, \mathcal{F}_{Rand})$-hybrid model. Moreover, if all the parties are honest, the protocol aborts with negligible probability.*

The proof of Theorem 4 can be found in Appendix E.

7 Efficiency

7.1 Complexity Analysis

We now analyse the communication complexity of the protocols presented in this paper. All the results indicate to total amount of communication sent over the $\binom{n}{2}$ channels and broadcast medium connecting the players.

$\Pi_{\mathbf{Offline}}$

PARAMETERS: Let N be a power of 2. Take a degree N polynomial $F(X)$ over the prime field $\mathbb{F}$. Define the ring $R := \mathbb{F}[X]/\big(F(X)\big)$ and consider Module-LPN parameters $t, c \in \mathbb{N}$.

PROCEDURES: The environment has access to the the procedures $\mathsf{Input}, \mathsf{LinComb}, \mathsf{Mult}$, and Output of $\mathcal{F}_{\text{3-DPF}}$.

Initialisation. Each party P_i samples $\alpha_i \xleftarrow{\$} \mathbb{F}^\times$ and inputs the value in $\mathcal{F}_{\text{3-DPF}}$ to obtain $[[\alpha_i]]$. Then, it outputs α_i.

Input Masks. Let P_j be the party to which the input masks are addressed.

1. For all $r \in [c]$, P_j samples $\boldsymbol{\beta^r} \xleftarrow{\$} \mathbb{F}^{\times t}$, $\boldsymbol{\omega^r} \xleftarrow{\$} [N]^t$. Then, it computes the polynomial
$$e^r(X) \leftarrow \sum_{l \in [t]} \beta^r[l] \cdot X^{\omega^r[l]}$$
2. For every $r \in [c]$, the parties compute the following operations using $\mathcal{F}_{\text{3-DPF}}$.
$$[[\boldsymbol{\beta^r}]] \leftarrow \mathsf{Input}(P_j, \boldsymbol{\beta^r}), \qquad [[\boldsymbol{\omega^r}]]_2 \leftarrow \mathsf{Input}(P_j, \boldsymbol{\omega^r}).$$
3. For every $i \neq j$ and $r \in [c]$, the parties compute $[[\boldsymbol{\mu_i^r}]] \leftarrow [[\alpha_i]] \cdot [[\boldsymbol{\beta^r}]]$.
4. For every $i \in [n]$ with $i \neq j$ and $r \in [c]$, the parties call $\mathcal{F}_{\text{3-DPF}}$ to compute
$$(\widetilde{\boldsymbol{e}}_{\boldsymbol{i}}^{\boldsymbol{r,0}}, \widetilde{\boldsymbol{e}}_{\boldsymbol{i}}^{\boldsymbol{r,1}}) \leftarrow \mathsf{2\text{-}DSPF}(N, [[\boldsymbol{\omega^r}]]_2, [[\boldsymbol{\mu_i^r}]], i, j)$$
and obtain a 2-party secret-sharing among P_i and P_j of the N-dimensional t-point vector with special positions $[[\boldsymbol{\omega^r}]]_2$ and non-zero elements $[[\boldsymbol{\mu_i^r}]]$. If $\mathcal{F}_{\text{3-DPF}}$ outputs ZERO, the protocol aborts. Let $\widetilde{\boldsymbol{e}}_{\boldsymbol{i}}^{\boldsymbol{r,0}}$ and $\widetilde{\boldsymbol{e}}_{\boldsymbol{i}}^{\boldsymbol{r,1}}$ denote the shares obtained by P_i and P_j respectively. We regard them as degree-$(N-1)$ polynomials $\widetilde{e}_i^r(X)$ and $\widetilde{e}_i^{r,1}(X)$.
5. For every $r \in [c]$, P_j computes
$$\widetilde{e}_j^r(X) \leftarrow \alpha_j \cdot e^r(X) + \sum_{j \neq i} \widetilde{e}_i^{r,1}(X).$$
6. For each $(i,k) \in [n]^2$ with $i \neq k$, P_i samples a random seed $s_{i,k} \xleftarrow{\$} \{0,1\}^\lambda$ and sends it to P_k.
7. The parties call $\mathcal{F}_{\text{Rand}}$ to obtain $a_i \xleftarrow{\$} R$ for every $i \in [c-1]$. Let $\boldsymbol{a} \leftarrow (a_0, a_1, \ldots, a_{c-2}, 1)$.
8. Each party P_i computes $m'_{x,i} \leftarrow \langle \boldsymbol{a}, \widetilde{\boldsymbol{e}}_{\boldsymbol{i}} \rangle$. Moreover, P_j computes $x \leftarrow \langle \boldsymbol{a}, \boldsymbol{e} \rangle$.
9. Each party P_i with $i \neq j$, sets
$$(x_i, m_{x,i}) \leftarrow (0, m'_{x,i}) + \sum_{k \neq i} \Big(G(s_{i,k}) - G(s_{k,i}) \Big).$$
P_j instead computes
$$(x_j, m_{x,j}) \leftarrow (x, m'_{x,j}) + \sum_{k \neq j} \Big(G(s_{j,k}) - G(s_{k,j}) \Big).$$
10. Each party P_i with $i \neq j$ outputs $(x_i, m_{x,i})$, P_j outputs $(x, x_j, m_{x,j})$.

Fig. 10. The offline phase protocol - Part 1

Triple Generation.

1. For all $i \in [n]$ and $r \in [c]$, P_i samples $\boldsymbol{\beta}_i^r \xleftarrow{\$} \mathbb{F}^{\times t}$, $\boldsymbol{\omega}_i^r \xleftarrow{\$} [N]^t$, $\boldsymbol{\gamma}_i^r \xleftarrow{\$} \mathbb{F}^{\times t}$ and $\boldsymbol{\eta}_i^r \xleftarrow{\$} [N]^t$. Then, it computes the polynomials

$$u_i^r(X) \leftarrow \sum_{l \in [t]} \beta_i^r[l] \cdot X^{\omega_i^r[l]}, \qquad v_i^r(X) \leftarrow \sum_{l \in [t]} \gamma_i^r[l] \cdot X^{\eta_i^r[l]}$$

2. For every $i \in [n]$ and $r \in [c]$, the parties compute the following operations using $\mathcal{F}_{\text{3-DPF}}$.

$$[[\boldsymbol{\beta}_i^r]] \leftarrow \mathsf{Input}(P_i, \boldsymbol{\beta}_i^r), \qquad [[\boldsymbol{\gamma}_i^r]] \leftarrow \mathsf{Input}(P_i, \boldsymbol{\gamma}_i^r),$$
$$[[\boldsymbol{\omega}_i^r]]_2 \leftarrow \mathsf{Input}(P_i, \boldsymbol{\omega}_i^r), \qquad [[\boldsymbol{\eta}_i^r]]_2 \leftarrow \mathsf{Input}(P_i, \boldsymbol{\eta}_i^r).$$

3. For every $i, j \in [n]$ with $i \neq j$ and $r \in [c]$, the parties compute

$$[[\boldsymbol{\mu}_{i,j}^r]] \leftarrow [[\alpha_i]] \cdot [[\boldsymbol{\beta}_j^r]], \qquad [[\boldsymbol{\nu}_{i,j}^r]] \leftarrow [[\alpha_i]] \cdot [[\boldsymbol{\gamma}_j^r]].$$

4. For every $i, j \in [n]$ with $i \neq j$ and $r \in [c]$, the parties call $\mathcal{F}_{\text{3-DPF}}$ to compute

$$(\widetilde{\boldsymbol{u}}_{i,j}^{r,\mathbf{0}}, \widetilde{\boldsymbol{u}}_{i,j}^{r,\mathbf{1}}) \leftarrow \text{2-}\mathsf{DSPF}(N, [[\boldsymbol{\omega}_j^r]]_2, [[\boldsymbol{\mu}_{i,j}^r]], i, j)$$

   and obtain a 2-party secret-sharing among P_i and P_j of the N-dimensional t-point vector with special positions $[[\boldsymbol{\omega}_j^r]]_2$ and non-zero elements $[[\boldsymbol{\mu}_{i,j}^r]]$. If $\mathcal{F}_{\text{3-DPF}}$ outputs ZERO, the protocol aborts. Let $\widetilde{\boldsymbol{u}}_{i,j}^{r,\mathbf{0}}$ and $\widetilde{\boldsymbol{u}}_{i,j}^{r,\mathbf{1}}$ denote the shares obtained by P_i and P_j respectively. We regard them as degree-$(N-1)$ polynomials $\widetilde{u}_{i,j}^{r,0}(X)$ and $\widetilde{u}_{i,j}^{r,1}(X)$.
5. For every $i, j \in [n]$ with $i \neq j$ and $r \in [c]$, the parties call $\mathcal{F}_{\text{3-DPF}}$ to compute

$$(\widetilde{\boldsymbol{v}}_{i,j}^{r,\mathbf{0}}, \widetilde{\boldsymbol{v}}_{i,j}^{r,\mathbf{1}}) \leftarrow \text{2-}\mathsf{DSPF}(N, [[\boldsymbol{\eta}_j^r]]_2, [[\boldsymbol{\nu}_{i,j}^r]], i, j)$$

   and obtain a 2-party secret-sharing among P_i and P_j of the N-dimensional t-point vector with special positions $[[\boldsymbol{\eta}_j^r]]_2$ and non-zero elements $[[\boldsymbol{\nu}_{i,j}^r]]$. If $\mathcal{F}_{\text{3-DPF}}$ outputs ZERO, the protocol aborts. Let $\widetilde{\boldsymbol{v}}_{i,j}^{r,\mathbf{0}}$ and $\widetilde{\boldsymbol{v}}_{i,j}^{r,\mathbf{1}}$ denote the shares obtained by P_i and P_j respectively. We regard them as degree-$(N-1)$ polynomials $\widetilde{v}_{i,j}^{r,0}(X)$ and $\widetilde{v}_{i,j}^{r,1}(X)$.
6. For every $i, j \in [n]$ and $r, s \in [c]$, the parties compute

$$[[\boldsymbol{\rho}_{i,j}^{r,s}]] \leftarrow [[\boldsymbol{\beta}_i^r]] \otimes [[\boldsymbol{\gamma}_j^s]], \qquad [[\boldsymbol{\zeta}_{i,j}^{r,s}]] \leftarrow [[\boldsymbol{\omega}_i^r]]_2 \boxplus [[\boldsymbol{\eta}_j^s]]_2.$$

7. For every $i, j \in [n]$ with $i \neq j$ and $r, s \in [c]$, the parties call $\mathcal{F}_{\text{3-DPF}}$ to compute

$$(\boldsymbol{w}_{i,j}^{r,s,\mathbf{0}}, \boldsymbol{w}_{i,j}^{r,s,\mathbf{1}}) \leftarrow \text{2-}\mathsf{DSPF}(2N, [[\boldsymbol{\zeta}_{i,j}^{r,s}]]_2, [[\boldsymbol{\rho}_{i,j}^{r,s}]], i, j)$$

   and obtain a 2-party secret-sharing among P_i and P_j of the $2N$-dimensional t^2-point vector with special positions $[[\boldsymbol{\zeta}_{i,j}^{r,s}]]_2$ and non-zero elements $[[\boldsymbol{\rho}_{i,j}^{r,s}]]$. Let $\boldsymbol{w}_{i,j}^{r,s,\mathbf{0}}$ and $\boldsymbol{w}_{i,j}^{r,s,\mathbf{1}}$ denote the shares obtained by P_i and P_j respectively. We regard them as degree-$(2N-1)$ polynomials $w_{i,j}^{r,s,0}(X)$ and $w_{i,j}^{r,s,1}(X)$.

Fig. 11. The offline phase protocol - Part 2

8. For every not-all-equal $i, j, k \in [n]$ and $r, s \in [c]$, the parties compute

$$[[\boldsymbol{\tau}^{\boldsymbol{r,s}}_{\boldsymbol{i,j,k}}]] \leftarrow [[\alpha_i]] \cdot [[\boldsymbol{\rho}^{\boldsymbol{r,s}}_{\boldsymbol{j,k}}]].$$

9. For every not-all-equal $i, j, k \in [n]$ and $r, s \in [c]$, the parties call $\mathcal{F}_{\text{3-DPF}}$ to compute

$$(\widetilde{\boldsymbol{w}}^{\boldsymbol{r,s,0}}_{\boldsymbol{i,j,k}}, \widetilde{\boldsymbol{w}}^{\boldsymbol{r,s,1}}_{\boldsymbol{i,j,k}}, \widetilde{\boldsymbol{w}}^{\boldsymbol{r,s,2}}_{\boldsymbol{i,j,k}}) \leftarrow \mathsf{3\text{-}DSPF}(2N, [[\boldsymbol{\zeta}^{\boldsymbol{r,s}}_{\boldsymbol{j,k}}]]_2, [[\boldsymbol{\tau}^{\boldsymbol{r,s}}_{\boldsymbol{i,j,k}}]], i, j, k)$$

and obtain a 3-party secret-sharing among P_i, P_j and P_k of the $2N$-dimensional t^2-point vector with special positions $[[\boldsymbol{\zeta}^{\boldsymbol{r,s}}_{\boldsymbol{j,k}}]]_2$ and non-zero elements $[[\boldsymbol{\tau}^{\boldsymbol{r,s}}_{\boldsymbol{i,j,k}}]]$. Let $\widetilde{\boldsymbol{w}}^{\boldsymbol{r,s,0}}_{\boldsymbol{i,j,k}}$, $\widetilde{\boldsymbol{w}}^{\boldsymbol{r,s,1}}_{\boldsymbol{i,j,k}}$ and $\widetilde{\boldsymbol{w}}^{\boldsymbol{r,s,2}}_{\boldsymbol{i,j,k}}$ denote the shares obtained by P_i, P_j and P_k respectively. We regard them as degree-$(2N-1)$ polynomials $\widetilde{w}^{r,s,0}_{i,j,k}(X)$, $\widetilde{w}^{r,s,1}_{i,j,k}(X)$ and $\widetilde{w}^{r,s,2}_{i,j,k}(X)$.

10. For every $r \in [c]$, each party P_i computes

$$\widetilde{u}^r_i(X) \leftarrow \alpha_i \cdot u^r_i(X) + \sum_{j \neq i} \Big(\widetilde{u}^{r,0}_{i,j}(X) + \widetilde{u}^{r,1}_{j,i}(X) \Big),$$
$$\widetilde{v}^r_i(X) \leftarrow \alpha_i \cdot v^r_i(X) + \sum_{j \neq i} \Big(\widetilde{v}^{r,0}_{i,j}(X) + \widetilde{v}^{r,1}_{j,i}(X) \Big).$$

11. For every $r, s \in [c]$, each party P_i computes over R

$$w^{rc+s}_i(X) \leftarrow u^r_i(X) \cdot v^s_i(X) + \sum_{j \neq i} \Big(w^{r,s,0}_{i,j}(X) + w^{r,s,1}_{j,i}(X) \Big),$$
$$\widetilde{w}^{rc+s}_i(X) \leftarrow \alpha_i \cdot u^r_i(X) \cdot v^s_i(X) + \sum_{(j,k) \neq (i,i)} \Big(\widetilde{w}^{r,s,0}_{i,j,k}(X) + \widetilde{w}^{r,s,1}_{k,i,j}(X) + \widetilde{w}^{r,s,2}_{j,k,i}(X) \Big).$$

12. The parties call $\mathcal{F}_{\text{Rand}}$ to obtain $a_i \xleftarrow{\$} R$ for every $i \in [c-1]$. Let $\boldsymbol{a} \leftarrow (a_0, a_1, \ldots, a_{c-2}, 1)$.
13. Each party P_i outputs

$$x_i \leftarrow \langle \boldsymbol{a}, \boldsymbol{u_i} \rangle, \qquad y_i \leftarrow \langle \boldsymbol{a}, \boldsymbol{v_i} \rangle, \qquad z_i \leftarrow \langle \boldsymbol{a} \otimes \boldsymbol{a}, \boldsymbol{w_i} \rangle,$$
$$m_{x,i} \leftarrow \langle \boldsymbol{a}, \widetilde{\boldsymbol{u}}_{\boldsymbol{i}} \rangle, \qquad m_{y,i} \leftarrow \langle \boldsymbol{a}, \widetilde{\boldsymbol{v}}_{\boldsymbol{i}} \rangle, \qquad m_{z,i} \leftarrow \langle \boldsymbol{a} \otimes \boldsymbol{a}, \widetilde{\boldsymbol{w}}_{\boldsymbol{i}} \rangle.$$

Fig. 12. The offline phase protocol - Part 3

The 3-party DPF protocol. The communication complexity of the 3-DPF procedure is

- 2 executions of the 2-party DPF in $\mathcal{F}_{\text{MPC}}$,
- $|\mathcal{T}| \cdot (2\lambda + 2)$ bits of communication for the OTs in the seed generation,
- $2(h+1) + 2\lambda$ bits for the first check,
- $6\lambda + 2n \cdot \lambda$ bits for the second check (including MAC check),
- $(3 + 11n)\lambda$ bits for the base case (including MAC checks),
- $3(|\mathcal{T}| - 2)\lambda$ bits for the correction words,
- $(4 + 9n)\lambda$ for the final check (including MAC checks).

Furthermore, the estimation needs to take into account the cost for the generation of preprocessing material used to implement $\mathcal{F}_{\text{auth-ROT}}$ and $\mathcal{F}_{\text{MPC}}$. Specifically,

- 3 multiplication triples over $\mathbb{F}$
- 7 input masks over $\mathbb{F}$
- 2λ input masks over $\mathbb{F}_2$
- $2|\mathcal{T}| + 2\lambda$ Correlated-OT tuples.
- 1 random secret-shared element over $\mathbb{F}_{2^\lambda}$.

Observe that $h = \log(\log(N))$, whereas

$$|\mathcal{T}| = 2 + \sum_{k=0}^{h-1} \mathrm{dE}(k) = \sqrt{N} + \sqrt[4]{N} + \sqrt[8]{N} + \cdots + \sqrt[2^h]{N} + 2 \leq 2\sqrt{N}.$$

The communication complexity of the 2-party DPF protocol of [BCG+20] is dominated by $(2\lambda + 3)\log(N) + 12\lambda$ bits. Also the generation of M Correlated-OT tuples can be performed with logarithmic communication in M and linearly in λ [BCG+19a]. In conclusion, the dominating term of the communication complexity of our protocol is $10\sqrt{N} \cdot \lambda + 22n \cdot \lambda$.

Triple Generation. Our triple generation protocol is particularly efficient from a communication point of view. The cost of the triple generation procedure is indeed

- $2n(n-1) \cdot ct$ executions of 2-DPF with output length N,
- $n(n-1) \cdot c^2t^2$ executions of 2-DPF with output length $2N$,
- $(n^3 - 1) \cdot c^2t^2$ executions of 3-DPF with output length $2N$,
- $2n \cdot ct \cdot \big(\lambda + \log(N)\big)$ bits of communication for the inputs,
- $4n^2(n-1) \cdot ct \cdot \lambda$ bits for the multiplications in step 3,
- $2n(n^3 - 1) \cdot c^2t^2 \cdot \lambda$ bits for the multiplications in step 8,
- $2n^3 \cdot c^2t^2 \cdot \lambda$ bits for the outer product,
- $2n^3 \cdot c^2t^2 \cdot \log(N)$ bits for IntAdd,
- $O(\lambda)$ complexity for the MAC checks[6].

[6] It is fundamental to run a check on the inputs of 2-DPF and 3-DPF before executing the procedures. Clearly, the MAC checks can be batched.

Considered the complexity analysis of 3-DPF in the previous paragraph, we conclude that the dominating term of the communication complexity of the procedure is $10\sqrt{2}\cdot n^3\cdot c^2t^2\cdot\sqrt{N}\cdot\lambda+24n^4\cdot c^2t^2\cdot\lambda$. We recall that every execution of Triple Generation permits to produce N fresh multiplication triples.

Observe that the protocol uses some preprocessing material for the implementation of $\mathcal{F}_{\text{3-DPF}}$, the generation of which does not actually affect the overall asymptotic complexity. Specifically, we need

- $2n\cdot ct$ input masks over $\mathbb{F}$,
- $2n\cdot ct\cdot\log(N)$ input masks over $\mathbb{F}_2$,
- $2n(n-1)\cdot ct+(n^3+n^2-1)\cdot c^2t^2$ multiplication triples over $\mathbb{F}$,
- $n^2\cdot c^2t^2\cdot\log(N)$ AND triples over $\mathbb{F}_2$.

We finally highlight that the protocol can be improved to achieve higher efficiency. However, the asymptotic complexity would not change. For instance, we use the 3-party DPF among P_i, P_j and P_k even if $i=j$, $j=k$ or $k=i$. In those cases, using a 2-party DPF would be sufficient.

Input Masks. The communication complexity of the input mask generation is asymptotically better than the triple generation. As a matter of fact, we do not need to rely on the 3-party DPF procedure which constitutes the bottleneck of the other procedure. We can generate input masks with logarithmic complexity in N: $(n-1)tc$ executions of 2-DPF, the communication of $tc\big(\lambda+\log(N)\big)$ bits for the inputs, $2tc\cdot n(n-1)\cdot\lambda$ bits for the multiplications and $n(n-1)\cdot\lambda$ for the seeds. We recall that in [BCG+20], the authors presented a protocol for 2-DPF with $O\big(\lambda\cdot\log(N)\big)$ communication complexity. The dominating terms are therefore $2n\cdot tc\cdot\log(N)\cdot\lambda+2n^2\cdot tc\cdot\lambda$. We recall that every execution of Input Masks generates N masks.

Clearly, the protocol uses also preprocessing material for the implementation of $\mathcal{F}_{\text{3-DPF}}$, however, as for the triple generation, it does not affect the asymptotic complexity. Specifically, we need

- tc inputs masks over $\mathbb{F}$,
- $tc\cdot\log(N)$ input masks over $\mathbb{F}_2$,
- $tc\cdot(n-1)$ multiplication triples over $\mathbb{F}$.

7.2 Concrete Efficiency

In Tables 1 and 2, we estimate the concrete communication cost of our protocol, for several sets of parameters with $n=3$ parties and 80-bit computational security. [7]. Since both the cost and number of 3-party DPFs are much larger than the 2-party case, these are by far the dominating factor. Therefore, to simplify the analyis, we ignore all other costs. From the analysis in Section 7.1,the total cost of a single 3-party DPF protocol is dominated by $\mathcal{T}\cdot(3\log p+2\lambda+2)$ bits for all 3 parties. Recall that $\mathcal{T}$ (which comes from the recursion step) depends on the

[7] Ring-LPN parameters are chosen as in [BCG+20].

N	2^{20}			2^{24}			2^{28}		
c	2	4	8	2	4	8	2	4	8
$w = ct$	96	40	32	96	40	32	96	40	32
Comm. (MB)	308	114	109	1120	417	418	4329	1641	1650
Stretch	0.16	0.44	0.46	0.72	1.93	1.92	2.98	7.85	7.81

Table 1. Estimated seed size for producing N triples with the 3-party PCG over a 128-bit field, with 80-bit computational security.

N	2^{20}			2^{24}			2^{28}		
c	2	4	8	2	4	8	2	4	8
$w = ct$	96	40	32	96	40	32	96	40	32
Comm. (MB)	701	288	186	2688	1041	688	10312	3960	2708
Stretch	0.07	0.17	0.27	0.30	0.77	1.17	1.25	3.25	4.76

Table 2. Estimated seed size for producing N triples with the 3-party PCG over a 128-bit field, with 128-bit computational security.

DPF domain size, N, and is upper-bounded by $\sqrt{N} + 2\sqrt[4]{N}$. The total number of 3-party DPFs needed in our triple generation protocol is $n(n-1)(n-2)c^2t^2$. We also rely on the variant of module-LPN with regular errors, which allows reducing the domain size of the DPFs from $2N$ down to $2N/t$.

Putting this together, we obtain the per-party communication costs in the table. The "stretch" of the protocol is defined as the ratio of the size of the uncompressed triples ($3N$ field elements) and the total communication cost. We see that, when producing around a million triples ($N = 2^{20}$), the stretch is still less than 1, meaning that the PCG does not achieve a good compression factor. Nevertheless, even at this level, we do achieve a protocol for generating triples with much lower communication than methods based on alternative techniques. For instance, using the Overdrive protocol based on homomorphic encryption [KPR18] requires almost 2GB of communication to generate the same number of triples, which is more than 10x our protocol.

When moving to larger batch sizes, the stretch improves, going up to almost 8x with $N = 2^{28}$ and $c \in \{4, 8\}$. This gives a strong saving in communication, but comes with larger computational costs due to the higher degree polynomial operations needed for arithmetic in the ring R. In practice, since these operations cost $O(N \log N)$, it seems likely that the smaller sizes of $N \leq 2^{24}$ will be preferable.

Acknowledgements

We thank the anonymous reviewers for their feedback, which helped to improve the paper. This work has been supported by the Independent Research Fund Denmark (DFF) under project number 0165-00107B (C3PO) and a starting grant from the Aarhus University Research Foundation.

References

AS21. Damiano Abram and Peter Scholl. Low-Communication Multiparty Triple Generation for SPDZ from Ring-LPN. Cryptology ePrint Archive, 2021, 2021.

BCG+19a. Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Peter Rindal, and Peter Scholl. Efficient two-round OT extension and silent non-interactive secure computation. In *ACM CCS 2019*. ACM Press, November 2019.

BCG+19b. Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In *CRYPTO 2019, Part III*, LNCS. Springer, Heidelberg, August 2019.

BCG+20. Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators from ring-LPN. In *CRYPTO 2020, Part II*, LNCS. Springer, Heidelberg, August 2020.

BDOZ11. Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In *EUROCRYPT 2011*, LNCS. Springer, Heidelberg, May 2011.

Bea92. Donald Beaver. Efficient multiparty protocols using circuit randomization. In *CRYPTO'91*, LNCS. Springer, Heidelberg, August 1992.

BGI15. Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In *EUROCRYPT 2015, Part II*, LNCS. Springer, Heidelberg, April 2015.

BGI16. Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In *ACM CCS 2016*. ACM Press, October 2016.

BKKO20. Paul Bunn, Jonathan Katz, Eyal Kushilevitz, and Rafail Ostrovsky. Efficient 3-party distributed ORAM. In *SCN 20*, LNCS. Springer, Heidelberg, September 2020.

DKL+13. Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. Practical covertly secure MPC for dishonest majority - or: Breaking the SPDZ limits. In *ESORICS 2013*, LNCS. Springer, Heidelberg, September 2013.

DPSZ12. Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *CRYPTO 2012*, LNCS. Springer, Heidelberg, August 2012.

GI14. Niv Gilboa and Yuval Ishai. Distributed point functions and their applications. In *EUROCRYPT 2014*, LNCS. Springer, Heidelberg, May 2014.

KOS16. Marcel Keller, Emmanuela Orsini, and Peter Scholl. MASCOT: Faster malicious arithmetic secure computation with oblivious transfer. In *ACM CCS 2016*. ACM Press, October 2016.

KPR18. Marcel Keller, Valerio Pastro, and Dragos Rotaru. Overdrive: Making SPDZ great again. In *EUROCRYPT 2018, Part III*, LNCS. Springer, Heidelberg, April / May 2018.

NNOB12. Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. In *CRYPTO 2012*, LNCS. Springer, Heidelberg, August 2012.

SV14. N. P. Smart and F. Vercauteren. Fully homomorphic simd operations. *Des. Codes Cryptography*, 71(1):57–81, April 2014.

YWL+20. Kang Yang, Chenkai Weng, Xiao Lan, Jiang Zhang, and Xiao Wang. Ferret: Fast extension for correlated OT with small communication. In *ACM CCS 20*. ACM Press, November 2020.

A Security of the Prime Field 3-Party DPF

Proof (of Theorem 1).

Correctness. Let the evaluation point be $x \in [N]$ and rewrite it as $x' \cdot \sqrt{N} + x''$ where $0 \leq x', x'' < \sqrt{N}$. If we add the vectors $\boldsymbol{v}^{\boldsymbol{i}}_{\boldsymbol{x'}}$ computed during the evaluation, we obtain

$$\sum_{i\in[3]} \boldsymbol{v}^{\boldsymbol{i}}_{\boldsymbol{x'}} = \boldsymbol{CW} \cdot \sum_{i=1}^{2} y_i[x'] - G(w_1^0) - G(w_2^0) + G(w_1^1) + G(w_2^1) + G(w_1^2) - G(w_2^2).$$

Suppose that $x' \neq \omega'$, then $\sum_{i=1}^{2} y_i[x'] = 0$, moreover, as $w_2^2 = w_2^1$,

$$\begin{aligned}\sum_{i\in[3]} \boldsymbol{v}^{\boldsymbol{i}}_{\boldsymbol{x'}} &= -G(w_1^0) - G(w_2^0) + G(w_1^1) + G(w_2^1) + G(w_1^2) - G(w_2^2) = \\ &= -G(a_{x'}) - G(b_{x'}) + G(b_{x'}) + G(w_2^1) + G(a_{x'}) - G(w_2^1) = \mathbf{0}.\end{aligned}$$

Therefore, the output of the evaluation is a secret-sharing of 0.

If instead $x' = \omega'$, then $\sum_{i=1}^{2} y_i[x'] = 1$, moreover,

$$\begin{aligned}\sum_{i\in[3]} \boldsymbol{v}^{\boldsymbol{i}}_{\boldsymbol{x'}} &= \boldsymbol{CW} - G(w_1^0) - G(w_2^0) + G(w_1^1) + G(w_2^1) + G(w_1^2) - G(w_2^2) = \\ &= \boldsymbol{CW} - G(a_{\omega'}) - G(d_{\omega'}) + G(b_{\omega'}) + G(d_{\omega'}) + G(d_{\omega'}) - G(c_{\omega'}) = \\ &= (\underbrace{\overbrace{0, 0, \ldots, 0}^{\omega''}, \beta, 0, 0, \ldots, 0}_{\sqrt{N} \text{ elements}}).\end{aligned}$$

The output of the evaluation is therefore a secret-sharing of β if $x = \omega$, of 0 otherwise.

Security. We construct the simulator. Let $T \subsetneq [3]$ be the set of corrupted parties (the set of corresponding to the keys that we must simulate) and let Sim_2 be the simulator of the 2-party DPF.

Case 1: $T = \{i\}$ for $i \in \{1, 2\}$. The value S^i_j can be simulated perfectly for every $j \in [\sqrt{N}]$ by sampling random elements in $\{0,1\}^\lambda$. Moreover, by the PRG security, we can model $\boldsymbol{CW}$ with a random vector in $\mathbb{F}^{\sqrt{N}}$. As a matter of fact, the seed $a_{\omega'}$ is known only to P_0. Finally, we can simulate the DPF keys $\widehat{\kappa}^i_1$ and $\widehat{\kappa}^i_2$ using Sim_2. It is easy to see that any effective distinguisher can be converted into a successful attacker against the security of the 2-party DPF.

$\mathcal{F}_{\text{MPC}}$

Input. On input $(\mathsf{Input}, D, i, x)$ from P_i and (Input, D, i) from all the other parties, the functionality checks that $D \in \{\mathbb{F}_2, \mathbb{F}\}$ and $x \in D$. If this is the case, it stores x with handle $[[x]]_D$ and domain D.

Linear Combination. On input $(\mathsf{LinComb}, m, a_0, \ldots, a_m, [[x_0]], \ldots, [[x_{m-1}]])$ from every party, where $x_i \in \mathbb{F}$ for every $i \in [m]$, the functionality stores $y \leftarrow a_m + \sum_{i=0}^{m-1} x_i \cdot a_i$ with domain $\mathbb{F}$ and handle $[[y]]$.

Multiplication. On input $(\mathsf{Mult}, [[x_1]], [[x_2]])$ from every party, where $x_i \in \mathbb{F}$ for every $i \in \{1, 2\}$, $\mathcal{F}_{\text{MPC}}$ stores $y \leftarrow x_1 \cdot x_2$ with domain $\mathbb{F}$ and handle $[[y]]$.

Division. On input $(\mathsf{Div}, [[x_1]], [[x_2]])$ from every party, where $x_i \in \mathbb{F}$ for every $i \in \{1, 2\}$, the functionality check whether $x_2 \neq 0$: In such case, it stores $y \leftarrow x_1/x_2$ with domain $\mathbb{F}$ and handle $[[y]]$. Then, it outputs OK to the adversary. If instead $x_2 = 0$, it outputs ZD to every party.

Random. On input (Rand, D) from every party, the functionality checks that $D \in \{\mathbb{F}, \mathbb{F}_2, \mathbb{F}_{2^\lambda}\}$. If this is the case, it stores $y \xleftarrow{\$} D$ with domain D and handle $[[y]]_D$.

Output. On input $(\mathsf{Output}, [[x]])$ from every party, the functionality sends x to the adversary and waits for a reply. If the answer is OK, the functionality outputs x to every honest party. Otherwise, it aborts.

2-DPF. On input $(\mathsf{2\text{-}DPF}, N, [[\omega]]_2, [[\beta]], \sigma_1, \sigma_2)$ from every party, where σ_1 and σ_2 are different indexes in $[n]$, N is a power of 2, ω is the bit representation of an integer in $[N]$ and β belongs either to $\mathbb{F}$ or $\mathbb{F}_{2^\lambda}$, the functionality does the following.

- If P_{σ_1} and P_{σ_2} are both corrupted, it sends β and ω to the adversary.
- If one party P_σ among P_{σ_1} and P_{σ_2} is corrupted, it waits for the adversary to send $\boldsymbol{y_\sigma}$ in $\mathbb{F}^N$ (if $\beta \in \mathbb{F}$) or in $\mathbb{F}_{2^\lambda}$ (if $\beta \in \mathbb{F}_{2^\lambda}$). Moreover, it waits for a set $I \subseteq [N]$. If $\omega \notin I$, it aborts. Otherwise, denoting by θ the index of the honest party among P_{σ_1} and P_{σ_2}, it outputs to P_θ

$$\boldsymbol{y_\theta} \leftarrow (\underbrace{\overbrace{0, 0, \ldots, 0}^{\omega}, \beta, 0, 0, \ldots, 0}_{N}) - \boldsymbol{y_\sigma}.$$

- If P_{σ_1} and P_{σ_2} are both honest, it samples $\boldsymbol{y_1}$ uniformly in $\mathbb{F}^N$ (if $\beta \in \mathbb{F}$) or in $\mathbb{F}_{2^\lambda}$ (if $\beta \in \mathbb{F}_{2^\lambda}$). Then, it computes

$$\boldsymbol{y_2} \leftarrow (\underbrace{\overbrace{0, 0, \ldots, 0}^{\omega}, \beta, 0, 0, \ldots, 0}_{N}) - \boldsymbol{y_1}.$$

  Finally, it outputs $\boldsymbol{y_i}$ to P_{σ_i} for every $i \in \{1, 2\}$.

IntAdd. On input $(\mathsf{IntAdd}, [[x_1]]_2, [[x_2]]_2)$ from every party, where x_i is the bit representation of an integer for every $i \in \{1, 2\}$, the functionality stores the bit representation of $y \leftarrow x_1 + x_2$ with domain $\mathbb{F}_2$ and handle $[[y]]_2$.

Abort. On input (Abort) from the adversary, the functionality aborts.

Fig. 13. The multiparty computation functionality

Case 2: $T = \{0\}$. By the correctness of $\mathsf{DPF}^2_{\sqrt{N}}$, we know that $b_{\omega'} + c_{\omega'} = \Delta$. Therefore, since the latter is random, $b_{\omega'}$ is uniformly distributed in $\{0,1\}^\lambda$ from the distinguisher perspective. As a consequence, by the PRG security, the correction word $\boldsymbol{CW}$ is indistinguishable from a random vector in $\mathbb{F}^{\sqrt{N}}$. The simulation of S^0_j is instead straightforward: we just need to sample 2 random elements in $\{0,1\}^\lambda$ for every $j \in [\sqrt{N}]$.

Case 3: $T = \{1,2\}$. Observe that the leakage function provides the simulator with ω', therefore, we can simulate the seeds and the 2-party DPF keys with perfect security. The correction word $\boldsymbol{CW}$ can instead be modelled using a random vector in $\mathbb{F}^{\sqrt{N}}$. As a matter of fact, the seed $a_{\omega'}$ is known only to P_0. Therefore, by the PRG security, $\boldsymbol{CW}$ is indistinguishable from a random element in $\mathbb{F}^{\sqrt{N}}$.

Case 4: $T = \{0,i\}$ *with* $i \in \{1,2\}$. By the correctness of $\mathsf{DPF}^2_{\sqrt{N}}$, we know that $b_{\omega'} + c_{\omega'} = \Delta$. Therefore, since the latter is random, either $b_{\omega'}$ or $c_{\omega'}$ is uniformly distributed in $\{0,1\}^\lambda$ from the distinguisher perspective. As a consequence, by the PRG security, the correction word $\boldsymbol{CW}$ is indistinguishable from a random vector in $\mathbb{F}^{\sqrt{N}}$.

The simulation of S^0_j and S^i_j is straightforward: for every $j \in [\sqrt{N}]$, we sample two random seeds $a_j, b_j \xleftarrow{\$} \{0,1\}^\lambda$ and we set $S^i_j \leftarrow a_j$ and $S^0_j \leftarrow \{a_j, b_j\}$. Observe that this is a perfect simulation even if $j = \omega'$.

Finally, we can simulate the DPF keys $\widehat{\kappa}^i_1$ and $\widehat{\kappa}^i_2$ using Sim_2. It is easy to see that any effective distinguisher can be converted into a successful attacker against the security of the 2-party DPF. □

B Security Proof of the PCG Construction (Theorem 2)

Proof (sketch).

Correctness. We start by showing the outputs form a consistent authenticated triple.

First, looking at the intermediate shares output by $\mathsf{DSPF}^2_{2N,t^2}.\mathsf{FullEval}$, we have

$$\sum_i w_i^{r,s} = \sum_{i\in[n]} \Big(u_i^r \cdot v_i^s + \sum_{j\neq i} \Big(\mathsf{DSPF}^2_{2N,t^2}.\mathsf{FullEval}(C_{i,j}^{r,s,0}) + \\ + \mathsf{DSPF}^2_{2N,t^2}.\mathsf{FullEval}(C_{j,i}^{r,s,1}) \Big) \Big) =$$
$$= \sum_{i\in[n],j\in[n]} (u_i^r \cdot v_j^s) =$$
$$= \sum_i u_i^r \cdot \sum_i v_i^s =$$
$$= u^r(X) \cdot v^s(X)$$

where the polynomial product is over $\mathbb{F}[X]$ (without reduction by $F(X)$).

Similarly,

$$\sum_i \widetilde{w}_i^{r,s} = \sum_{i\in[n]} \Big(\alpha_i \cdot u_i^r \cdot v_i^s + \sum_{\substack{j,k \\ (i,i)\neq(j,k)}} \Big(\mathsf{DSPF}^3_{2N,t^2}.\mathsf{FullEval}(W_{i,j,k}^{r,s,0}) + \\ + \mathsf{DSPF}^3_{2N,t^2}.\mathsf{FullEval}(W_{k,i,j}^{r,s,1}) + \\ + \mathsf{DSPF}^3_{2N,t^2}.\mathsf{FullEval}(W_{j,k,i}^{r,s,2}) \Big) \Big) =$$
$$= \sum_{i,j,k\in[n]} \alpha_i \cdot u_j^r \cdot v_k^s =$$
$$= \sum_i \alpha_i \cdot \sum_i u_i^r \cdot \sum_i v_i^s$$

Then, we have that the output shares satisfy

$$\sum_i z_i = \left\langle \boldsymbol{a} \otimes \boldsymbol{a}, \sum_i \boldsymbol{w}_i \right\rangle =$$
$$= \langle \boldsymbol{a} \otimes \boldsymbol{a}, \boldsymbol{u} \otimes \boldsymbol{v} \rangle =$$
$$= \langle \boldsymbol{a}, \boldsymbol{u} \rangle \cdot \langle \boldsymbol{a}, \boldsymbol{v} \rangle$$

which is $x \cdot y$, as required.

Similarly, it can be seen that the shares of $m_{x,i}, m_{y,i}, m_{z,i}$ form consistent sharings of the MACs $\alpha x, \alpha y, \alpha z$.

For correctness, we also need to argue that these shares are all indistinguishable from random, conditioned on forming a valid triple. Note that each of the shares x_i, y_i is a module-LPN sample, so is pseudorandom under this assumption. The remaining shares are all full evaluations of DSPFs, which are pseudorandom by the security of the DPF (since any secure DPF with additively-shared outputs must have pseudorandom shares [BGI15]).

Security. For the security property, we need to show that for any subset $T \subset [n]$ of corrupted parties, the distribution

$$\{(\kappa_i)_{i\in T}, (x_j, y_j, z_j, m_{x,j}, m_{y,j}, m_{z,j})_{j\notin T}\}$$

is indistinguishable from the one where the honest expanded outputs x_j, y_j etc. are replaced with *reverse-sampled* outputs, chosen at random, conditioned on forming a correct multiplication triple together with the corrupt parties' outputs.

This can be shown in a sequence of hybrids where, similarly to the correctness case, we rely on module-LPN for the pseudorandomness of the x_j, y_j shares, and the correctness and security of the DPFs for the shares of z and the MACs. We omit the details. □

C Double Exponential Notation

Proof (of Lemma 1). We proceed by induction over h. If $h = 0$, the result is straightforward. Indeed, $N = 2$, so, either $\omega = 0$ or $\omega = 1$, corresponding to $x(-1) = 0$ and $x(-1) = 1$ respectively. Observe that the representation is unique (notice that $[0] = \emptyset$).

Suppose now that the claim is true for $h-1$, we prove it for h. Take $\omega \in [N]$. Define

$$y(h-1) := \omega \bmod \mathrm{dE}(h-1), \qquad x(h-1) := \frac{\omega - y(h-1)}{\mathrm{dE}(h-1)}$$

Observe that both values are natural numbers, moreover, $0 \le y(h-1), x(h-1) < \mathrm{dE}(h-1)$. The inequality for $x(h-1)$ follows from the fact that

$$x(h-1) = \frac{\omega - y(h-1)}{\mathrm{dE}(h-1)} < \frac{\mathrm{dE}(h)}{\mathrm{dE}(h-1)} = \mathrm{dE}(h-1).$$

By inductive hypothesis, we can rewrite $y(h-1)$ as

$$y(h-1) = x(-1) + \sum_{i\in[h-1]} x(i)\cdot \mathrm{dE}(i)$$

where $x(i) \in [\mathrm{dE}(i)]$ for every $i \in [h-1] \cup \{-1\}$. So, we have that

$$\omega = x(-1) + \sum_{i\in[h]} x(i)\cdot \mathrm{dE}(i)$$

where $x(i) \in [\mathrm{dE}(i)]$ for every $i \in [h] \cup \{-1\}$. To understand why the representation is unique, suppose that we have another double exponential representation for ω. Let it be $x'(-1), x'(0), \ldots, x'(h-1)$. We have that

$$y(h-1) = \omega \bmod \mathrm{dE}(h-1) = x'(-1) + \sum_{i\in[h-1]} x'(i)\cdot \mathrm{dE}(i).$$

By inductive hypothesis, the double exponential representation of $y(h-1)$ is unique, therefore, $x(i) = x'(i)$ for every $i \in [h-1] \cup \{-1\}$. The proof is concluded by observing that

$$x'(h-1) = \frac{\omega - y(h-1)}{\mathrm{dE}(h-1)} = x(h-1).$$

□

D Security Proof of $\Pi_{\text{3-DPF}}$

Lemma 2. *For every $k \in [h]$, we define.*

$$\mathbb{1}_k := \big(1\ 1\ \ldots\ 1\big) \in \mathbb{F}_2^{\mathrm{dE}(k)}, \qquad \mathbb{0}_k := \big(0\ 0\ \ldots\ 0\big) \in \mathbb{F}_2^{\mathrm{dE}(k)}$$

$$B_k := \left(\begin{array}{ccccc|ccccc|c|ccccc} \mathbb{1}_k & \mathbb{0}_k & \mathbb{0}_k & \ldots & \mathbb{0}_k & \mathbb{1}_k & \mathbb{0}_k & \mathbb{0}_k & \ldots & \mathbb{0}_k & \ldots & \mathbb{1}_k & \mathbb{0}_k & \mathbb{0}_k & \ldots & \mathbb{0}_k \\ \mathbb{0}_k & \mathbb{1}_k & \mathbb{0}_k & \ldots & \mathbb{0}_k & \mathbb{0}_k & \mathbb{1}_k & \mathbb{0}_k & \ldots & \mathbb{0}_k & \ldots & \mathbb{0}_k & \mathbb{1}_k & \mathbb{0}_k & \ldots & \mathbb{0}_k \\ \mathbb{0}_k & \mathbb{0}_k & \mathbb{1}_k & \ldots & \mathbb{0}_k & \mathbb{0}_k & \mathbb{0}_k & \mathbb{1}_k & \ldots & \mathbb{0}_k & \ldots & \mathbb{0}_k & \mathbb{0}_k & \mathbb{1}_k & \ldots & \mathbb{0}_k \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \ldots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbb{0}_k & \mathbb{0}_k & \mathbb{0}_k & \ldots & \mathbb{1}_k & \mathbb{0}_k & \mathbb{0}_k & \mathbb{0}_k & \ldots & \mathbb{1}_k & \ldots & \mathbb{0}_k & \mathbb{0}_k & \mathbb{0}_k & \ldots & \mathbb{1}_k \end{array}\right) \in \mathbb{F}_2^{\mathrm{dE}(k) \times N}$$

$$B_{-1} := \left(\begin{array}{cc|cc|c|cc} 1 & 0 & 1 & 0 & \ldots & 1 & 0 \\ 0 & 1 & 0 & 1 & \ldots & 0 & 1 \end{array}\right) \in \mathbb{F}_2^{2 \times N}$$

Let $\boldsymbol{v}$ be an N-dimensional unit vector over a field $\mathbb{F}$ having special position ω and non-zero element β. Then, for every $k \in \mathcal{K}$, $\boldsymbol{v_k} := B_k \cdot \boldsymbol{v}$ is a $\mathrm{dE}(k)$-dimensional unit vector having special position $\omega(k)$ and non-zero element β.

Proof. Observe that $\boldsymbol{v_k}$ will be equal to β times the ω-th column of B_k. Every column of B_k has only one element different from zero, its value is 1. Therefore, $\boldsymbol{v_k}$ will be a unit vector with non-zero value β.

It remains to prove that the special position is $\omega(k)$. The matrix B_k can be split into blocks of size $\mathrm{dE}(k) \times \mathrm{dE}(k+1)$. There are $N/\mathrm{dE}(k+1)$ blocks in total. The ω-th column will be the l-th column of its block where $l := \omega \bmod \mathrm{dE}(k+1)$.

Each block can be further slit into $\mathrm{dE}(k)$ sequences of identical columns. The columns of the j-th sequence will be unit vectors with special position j. The ω-th column of B_k will belong to the m-th sequence of its block where

$$m := \frac{l - \big(\omega \bmod \mathrm{dE}(k)\big)}{\mathrm{dE}(k)} \qquad \text{if } k \geq 0, \qquad\qquad m := l \bmod 2 \quad \text{if } k = -1.$$

The special position of $\boldsymbol{v_k}$ will therefore be m. Remember that, by Lemma 1,

$$\omega = \omega(-1) + \sum_{i \in [h]} \omega(i) \cdot \mathrm{dE}(i)$$

so it is immediate to prove that $m = \omega(k)$. □

Lemma 3. *For every $k \in [h]$, let C_k be the $2^k \times \mathrm{dE}(k)$ matrix in $\mathbb{F}_2$ whose j-th column is the bit representation of j. We define the matrix C as*

$$C := \begin{pmatrix} C_0 & 0 & 0 & \dots & 0 \\ 0 & C_0 & 0 & \dots & 0 \\ 0 & 0 & C_1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & C_{h-1} \end{pmatrix} \in \mathbb{F}_2^{\log(N) \times |\mathcal{T}|}$$

Let $\eta \in [N]$ and, for every $k \in \mathcal{K}$, let $\boldsymbol{\delta_k}$ be $\mathrm{dE}(k)$-dimensional unit vector with special position $\eta(k)$ and non-zero element 1. Let $\boldsymbol{\delta}$ be the vector obtained by concatenating $\boldsymbol{\delta_k}$ for every $k \in \mathcal{K}$. Then, we have that $\boldsymbol{z} := C \cdot \boldsymbol{\delta}$ is the $\log(N)$-bit string representing η.

Proof. Observe that the number of rows in C is

$$1 + \sum_{k=0}^{h-1} 2^k = 2^h = \log(N).$$

We know that $\boldsymbol{z}$ can be split into the concatenation of $\boldsymbol{z_k} := C_k \cdot \boldsymbol{\delta_k}$ for every $k \in \mathcal{K}$ (we use the convention $C_{-1} := C_0$). Therefore, $\boldsymbol{z_k}$ is a 2^k-bit sequence representing $\eta(k)$. Now, it is immediate to see that the binary vector $\boldsymbol{z}$ represents the integer

$$\eta(-1) + \sum_{k \in [h]} \eta(k) \cdot \mathrm{dE}(k) = \eta.$$

□

We are now ready to prove the security of $\Pi_{\text{3-DPF}}$.

Proof (Theorem 3). Consider the simulator $\mathcal{S}_{\text{3-DPF}}$ described in Figures 14, 15 and 16. We prove that no PPT adversary is able to distinguish between the protocol $\Pi_{\text{3-DPF}}$ and the composition of $\mathcal{F}_{\text{3-DPF}}$ with $\mathcal{S}_{\text{3-DPF}}$.

Throughout the proof, we denote the index of the corrupted party among P_{σ_1} and P_{σ_2} by σ. If P_{σ_1} and P_{σ_2} are both honest, we set $\sigma \leftarrow 1$. The index of the other party is denoted by θ.

We start by observing that the simulation of 2-DPF is perfect.

Claim 3.1. *The simulation of the conversion from $\mathbb{F}$-secret-sharing to binary secret-sharing is perfect.*

Proof of the claim. The only communications involved in this phase of the protocol are abort notifications from P_{σ_1} and P_{σ_2}. If both the parties are corrupted, the result is clear as the simulator makes the functionality abort whenever the adversary requests it.

If only one among P_{σ_1} and P_{σ_2} is corrupted. The simulator still makes the functionality abort when the adversary requests it. However, we also need to

$\mathcal{S}_{\text{3-DPF}}$

MPC Functionality. The simulator directly forwards the messages between the functionality and the adversary.

3-DPF.

1. If P_{σ_0}, P_{σ_1} and P_{σ_2} are all corrupted, the simulator receives ω and β from the functionality and runs the protocol with the adversary.
2. $\mathcal{S}_{\text{3-DPF}}$ simulates 2-DPF:
   - If P_{σ_1} and P_{σ_2} are both corrupted, the simulator receives ω from the functionality and forwards it to the adversary together with 1.
   - If only one among P_{σ_1} and P_{σ_2} is corrupted, let σ and θ denote the index of the malicious party and the honest party respectively. $\mathcal{S}_{\text{3-DPF}}$ waits for $\boldsymbol{y_\sigma}$ and $I \subseteq [N]$ from the adversary and forwards the latter to the functionality. It then passes the reply to the adversary.
   - If P_{σ_1} and P_{σ_2} are both honest, $\mathcal{S}_{\text{3-DPF}}$ sets $\sigma \leftarrow 1$ and $\theta \leftarrow 2$. Then, it generates $\boldsymbol{y_\sigma} \xleftarrow{\$} \mathbb{F}^N$.
3. If P_{σ_1} and P_{σ_2} are not both corrupted, $\mathcal{S}_{\text{3-DPF}}$ computes
   (a) $\boldsymbol{y_{k,\sigma}} \leftarrow B_k \cdot \boldsymbol{y_\sigma}$ for every $k \in \mathcal{K}$
   (b) $\mathcal{Z} \leftarrow \{(k,j) \in \mathcal{T} \mid y^j_{k,\sigma} = 0\}$
   (c) $\mathcal{U} \leftarrow \{(k,j) \in \mathcal{T} \mid y^j_{k,\sigma} = 1\}$
   (d) $I' := \left\{ \eta \in [N] \;\middle|\; \begin{array}{ll} \eta(k) = j & \forall (k,j) \in \mathcal{Z} \\ \eta(k') \neq j' & \forall (k',j') \in \mathcal{U} \end{array} \right\}$
   (e) $\mathcal{S}_{\text{3-DPF}}$ sends I' to the functionality. If $\mathcal{F}_{\text{3-DPF}}$ aborts, the simulator tells the adversary that there exists $(k,j) \in \mathcal{T}$ such that $y^j_{k,\theta} = 0$.
   (f) If P_σ is honest and $\mathcal{Z} \neq \emptyset$, the simulator tells the adversary that there exists $(k,j) \in \mathcal{T}$ such that $y^j_{k,\sigma} = 0$.
   (g) If P_σ is corrupted and the adversary requests an abort, claiming that there exists $(k,j) \in \mathcal{T}$ such that $y^j_{k,\sigma} = 0$, the simulator sends Abort to the functionality.
   (h) For every $(k,j) \in \mathcal{T}$, $\mathcal{S}_{\text{3-DPF}}$ sets $b^j_{k,\sigma} \leftarrow \begin{cases} y^j_{k,\sigma} \bmod 2 & \text{if } y^j_{k,\sigma} \neq 0, \\ 1 & \text{otherwise.} \end{cases}$
   (i) For every $(k,j) \in \mathcal{T}$, $\mathcal{S}_{\text{3-DPF}}$ sets $b^j_{k,\theta} \leftarrow \delta_0(j) \oplus b^j_{k,\sigma} \oplus 1$.
4. $\mathcal{S}_{\text{3-DPF}}$ simulates again 2-DPF:
   - If P_{σ_1} and P_{σ_2} are both corrupted, the simulator sends ω to the adversary together with a value $\Delta \xleftarrow{\$} \mathbb{F}_{2^\lambda}$.
   - If only one among P_{σ_1} and P_{σ_2} is corrupted, $\mathcal{S}_{\text{3-DPF}}$ waits for $\boldsymbol{T_\sigma}$ and $I'' \subseteq [N]$ from the adversary and forwards the latter to the functionality. It then passes the reply to the adversary.
   - If P_{σ_1} and P_{σ_2} are both honest, $\mathcal{S}_{\text{3-DPF}}$ generates $\boldsymbol{T_\sigma} \xleftarrow{\$} \mathbb{F}^N$.
5. For every $k \in \mathcal{K}$, the simulator sets $\boldsymbol{T_{k,\sigma}} \leftarrow B_K \cdot \boldsymbol{T_\sigma}$.
6. For every $(k,j) \in \mathcal{T}$, the simulator sets $Y^j_{k,\sigma} \leftarrow H\big(T^j_{k,\sigma}, (k,j)\big)$.
7. $\mathcal{S}_{\text{3-DPF}}$ simulates the second part of the seed generation by interacting with the corrupted parties using internal copies of $\mathcal{F}_{\text{auth-ROT}}$ and modelling $b^j_{k,i}$ when P_{σ_i} is honest using the values computed in step 3.

Fig. 14. The simulator $\mathcal{S}_{\text{3-DPF}}$ - Part 1

8. If there exists only one corrupted party among P_{σ_1} and P_{σ_2}, $\mathcal{S}_{\text{3-DPF}}$ computes $\boldsymbol{\epsilon_k} \leftarrow \boldsymbol{b_{k,\sigma}} \oplus \boldsymbol{w_{k,\sigma}} \oplus \boldsymbol{t_{k,\sigma}}$ for every $k \in \mathcal{K}$.
9. If P_{σ_1} and P_{σ_2} are both corrupted, for every $(k,j) \in \mathcal{T}$, $\mathcal{S}_{\text{3-DPF}}$ computes $\epsilon_k^j \leftarrow w_{k,1}^j \oplus w_{k,2}^j \oplus t_{k,1}^j \oplus t_{k,2}^j \oplus \delta_{\omega(k)}(j) \oplus 1$.
10. $\mathcal{S}_{\text{3-DPF}}$ simulates the first check using the internal copies of $\mathcal{F}_{\text{auth-ROT}}$. If an abort occurs, $\mathcal{S}_{\text{3-DPF}}$ sends Abort to the functionality.
11. $\mathcal{S}_{\text{3-DPF}}$ simulates the second check using the internal copies of $\mathcal{F}_{\text{auth-ROT}}$ and $\mathcal{F}_{\text{MPC}}$ and simulating ω with 0 when P_{σ_1} and P_{σ_2} are not both corrupted. If an abort occurs, $\mathcal{S}_{\text{3-DPF}}$ sends Abort to the functionality.
12. $\mathcal{S}_{\text{3-DPF}}$ simulates the base case:
    (a) It adds the values $z_i' \in \mathbb{F}$ received from every corrupted party P_{σ_i} obtaining z'.
    (b) If the functionality sent ZERO, the simulator sends $CW \leftarrow 0$ to the adversary.
    (c) Otherwise, the simulator sends $CW \xleftarrow{\$} \mathbb{F}^{\times}$ to the adversary.
13. For every $k \in [h]$, $\mathcal{S}_{\text{3-DPF}}$ simulates the generation of the correction word $\boldsymbol{CW_k}$ using $\boldsymbol{CW_{k,i}} \xleftarrow{\$} \mathbb{F}^{\text{dE}(k)}$ for every honest party P_{σ_i}. Moreover, it adds the vectors $\boldsymbol{CW'_{k,i}} \in \mathbb{F}^{\text{dE}(k)}$ received from every corrupted party P_{σ_i}, obtaining $\boldsymbol{CW'_k}$.
14. For every $\eta \in [N]$ and $(k,j) \in \mathcal{T}$ with $j \neq \eta(k)$, $\mathcal{S}_{\text{3-DPF}}$ computes
    (a) $S_{k,0,b}^{j,\eta} \leftarrow X_k^j[b]$ for every $b \in \{0,1\}$
    (b) $S_{k,i,1}^{j,\eta} \leftarrow Y_{k,\sigma}^j$ for every honest $i \in \{1,2\}$
    (c) $y_{k,\sigma}^{j,\eta} \leftarrow y_{k,\sigma}^j$, $y_{k,\theta}^{j,\eta} \leftarrow -y_{k,\sigma}^j$, $y_{k,0}^{j,\eta} \leftarrow 0$
    (d) $b_{k,\sigma}^{j,\eta} \leftarrow b_{k,\sigma}^j$
    (e) $b_{k,\theta}^{j,\eta} \leftarrow b_{k,\sigma}^j \oplus 1$
    (f) $S_{k,1,0}^{j,\eta} \leftarrow X_k^j[t_{k,1}^{j,\eta}]$ where $t_{k,1}^{j,\eta} \leftarrow w_{k,1}^j \oplus b_{k,1}^{j,\eta}$
    (g) $S_{k,2,0}^{j,\eta} \leftarrow W_k^j \oplus Z_k^j[b_{k,2}^{j,\eta}]$
15. For every $\eta \in [N]$ and $j \neq \eta(-1)$, $\mathcal{S}_{\text{3-DPF}}$ computes

$$\widehat{v}_{0,\eta}^j \leftarrow -CW \cdot \sum_{\substack{i=0 \\ \sigma_i \in \mathcal{H}}}^{2} \left(u_i^0 \cdot \text{Enc}(S_{-1,i,0}^{j,\eta}) + u_i^1 \cdot \text{Enc}(S_{-1,i,1}^{j,\eta}) \right)$$

$$\widehat{v}_{0,\eta}^{\eta(-1)} \leftarrow CW \cdot z' - \widehat{v}_{0,\eta}^j$$

$$\boldsymbol{\widehat{v}_0^\eta} \leftarrow (\widehat{v}_{0,\eta}^0, \widehat{v}_{0,\eta}^1)$$

Fig. 15. The simulator $\mathcal{S}_{\text{3-DPF}}$ - Part 2

16. For every $\eta \in [N]$ and $k \in [h]$, $\mathcal{S}_{\text{3-DPF}}$ computes
    (a) For every $j \neq \eta(k)$

$$\widehat{\boldsymbol{v}}^{\boldsymbol{j}}_{\boldsymbol{k+1,\eta}} \leftarrow - \sum_{\substack{i=0 \\ \sigma_i \in \mathcal{H}}}^{2} \left(u_i^0 \cdot G_k(S^{j,\eta}_{k,i,0}) + u_i^1 \cdot G_k(S^{j,\eta}_{k,i,1}) + y^{j,\eta}_{k,i} \cdot \boldsymbol{CW_k} \right)$$

    (b) For $j = \eta(k)$

$$\widehat{\boldsymbol{v}}^{\boldsymbol{j}}_{\boldsymbol{k+1,\eta}} \leftarrow \widehat{\boldsymbol{v}}^{\boldsymbol{\eta}}_{\boldsymbol{k}} - \boldsymbol{CW'_k} - \sum_{l \neq j} \widehat{\boldsymbol{v}}^{\boldsymbol{l}}_{\boldsymbol{k+1,\eta}} + \boldsymbol{CW_k} \cdot \sum_{\substack{i=0 \\ \sigma_i \in \mathcal{C}}}^{2} \sum_{l} y^{l,\eta}_{k,i}$$

    (c) $\widehat{\boldsymbol{v}}_{\boldsymbol{k+1,\eta}} \leftarrow \left(\widehat{\boldsymbol{v}}^{\boldsymbol{0}}_{\boldsymbol{k+1,\eta}} \parallel \widehat{\boldsymbol{v}}^{\boldsymbol{1}}_{\boldsymbol{k+1,\eta}} \parallel \dots \parallel \widehat{\boldsymbol{v}}^{\boldsymbol{\mathrm{dE}(k)-1}}_{\boldsymbol{k+1,\eta}} \right)$

17. $\mathcal{S}_{\text{3-DPF}}$ simulates the final check
    (a) It sends $\chi \xleftarrow{\$} \mathbb{F}^N$ to the adversary
    (b) It waits for d'_i from every corrupted party P_{σ_i} with $i \in \{1, 2\}$. It then adds the received values to obtain d'.
    (c) It waits for ζ'_i from every corrupted party P_{σ_i}. It then adds the received values to obtain ζ'.
    (d) If P_{σ_0} is honest and there exists any $(k, j) \in \mathcal{T}$ such that $\epsilon^j_k = 1$ the simulator sends $\rho \xleftarrow{\$} \mathbb{F}$ to the adversary and Abort to the functionality.
    (e) $\mathcal{S}_{\text{3-DPF}}$ computes $d \leftarrow \langle \boldsymbol{\chi}, \boldsymbol{y_\sigma} \rangle$ if only one party among P_{σ_1} and P_{σ_2} is corrupted. If instead both P_{σ_1} and P_{σ_2} are corrupted, it sets $d \leftarrow \chi_\omega$. Finally, if both P_{σ_1} and P_{σ_2} are honest it sets $d \leftarrow 0$.
    (f) If $d \neq d'$, the simulator computes for every $\eta \in [N]$,

$$\zeta'_\eta \leftarrow \langle \boldsymbol{\chi}, \widehat{\boldsymbol{v}}_{\boldsymbol{h,\eta}} \rangle.$$

    Then, it sets $\beta_\eta \leftarrow (\zeta' - \zeta'_\eta)/(d' - d)$. Finally, the simulator sends $(\beta_\eta, \widehat{\boldsymbol{v}}_{\boldsymbol{h,\eta}})_{\eta \in [N]}$ to the functionality. If the answer is $(\omega, \beta) \in [N] \times \mathbb{F}$, the simulator sends $\rho \leftarrow \zeta' - \zeta'_\omega - (d' - d) \cdot \beta$ to the adversary. Otherwise, it sends 0.
    (g) If $d = d'$, the simulator computes for every $\eta \in [N]$,

$$\zeta'_\eta \leftarrow \langle \boldsymbol{\chi}, \widehat{\boldsymbol{v}}_{\boldsymbol{h,\eta}} \rangle.$$

    Then, it sets

$$I''' := \{\eta \in [N] \mid \zeta' = \zeta'_\eta\}$$

    and samples $\widehat{\eta} \xleftarrow{\$} I'''$. Finally, it sends $(I''', \widehat{\boldsymbol{v}}_{\boldsymbol{h,\widehat{\eta}}})$ to the functionality and waits for a reply. If the answer is $\omega \in [N]$, the simulator sends $\rho \leftarrow \zeta' - \zeta'_\omega$ to the adversary. Otherwise, it sends 0.

Fig. 16. The simulator $\mathcal{S}_{\text{3-DPF}}$ - Part 3

check that an abort occurs when there would exist $(k, j) \in \mathcal{T}$ such that $y^j_{k,\theta} = 0$. Notice that in the protocol, by Lemma 2,

$$y^j_{k,\theta} = \delta_{\omega(k)}(j) - y^j_{k,\sigma},$$

so, $y^j_{k,\theta} = 0$ if and only if $j = \omega(k)$ and $y^j_{k,\sigma} = 1$, or $j \neq \omega(k)$ and $y^j_{k,\sigma} = 0$. In other words, that happens if and only if $\omega \notin I'$.

When both P_{σ_1} and P_{σ_2} are honest, observe that $\boldsymbol{y_\sigma}$ has the same distribution as in the protocol and the simulator makes the functionality abort when there exists $(k, j) \in \mathcal{T}$ such that $y^j_{k,\sigma} = 0$. Moreover, by querying I' to the functionality, $\mathcal{S}_{\text{3-DPF}}$ makes the simulation abort whenever there would exist $(k, j) \in \mathcal{T}$ such that $y^j_{k,\theta} = 0$. ■

If all the parties are honest, the probability that the protocol aborts during the conversion from $\mathbb{F}$-secret-sharing to binary secret-sharing is negligible. Indeed, $\boldsymbol{y_1}$ is random. Since B_k is full-rank for every $k \in \mathcal{K}$, $\boldsymbol{y_{k,1}}$ is random in $\mathbb{F}^{\mathrm{dE}(k)}$. Hence, for every $(k, j) \in \mathcal{T}$, the probability that $y^j_{k,1} \in \{0, 1\}$ is $2 \cdot |\mathbb{F}|^{-1}$ which is negligible. If $y^j_{k,1} \notin \{0, 1\}$, $y^j_{k,2} \neq 0$. Observe that $\mathcal{T}$ has a polynomial number of elements, so we conclude by the union bound.

We notice that the simulation of the first part of the seed generation is perfect. Indeed, $\mathcal{S}_{\text{3-DPF}}$ just needs to model 2-DPF and we already argued that this can be done with information theoretical security. The simulation of the second part is perfect too, indeed, the only difference between protocol and simulation is that, in the latter, different values for $b^j_{k,\theta}$ are used. However, all the information is masked by uniformly random bits $t^j_{k,\theta}$ unknown to the adversary, resulting in exactly the same view.

Claim 3.2. *Consider the protocol and suppose that both P_{σ_1} and P_{σ_2} are honest. Then, for every $(k, j) \in \mathcal{T}$, $b^j_{k,2} = \delta_{\omega(k)} \oplus b^j_{k,1} \oplus 1$.*

If instead there exists only one $\sigma \in \{\sigma_1, \sigma_2\}$ such that P_σ is corrupted, for every $(k, j) \in \mathcal{T}$,

$$b^j_{k,\theta} = \begin{cases} \delta_{\omega(k)}(j) \oplus b^j_{k,\sigma} \oplus 1 & \text{if } y^j_{k,\sigma} \neq 0, \\ \delta_{\omega(k)}(j) & \text{otherwise.} \end{cases}$$

Proof of the claim. Let σ denote the index of the corrupted party, if it exists. Otherwise, let σ be 1. Let θ denote the index of the other party.

By Lemma 2, we know that $y^j_{k,\sigma} + y^j_{k,\theta} = \delta_{\omega(k)}(j)$ for every $(k, j) \in \mathcal{T}$. Here the addition denotes the sum modulo $p := |\mathbb{F}|$.

Suppose that $y^j_{k,\sigma}$ and $y^j_{k,\theta}$ are both different from 0. By denoting the addition over the integers by $\boxplus$, we have that $2 \leq y^j_{k,\sigma} \boxplus y^j_{k,\theta} \leq 2p - 2$. Since the sum must reduce to $\delta_{\omega(k)}(j)$ modulo p, we have that

$$y^j_{k,\sigma} \boxplus y^j_{k,\theta} = \delta_{\omega(k)}(j) \boxplus p.$$

If we reduce both sides of the equation modulo 2, we obtain that $b^j_{k,\sigma} \oplus b^j_{k,\theta} = \delta_{\omega(k)}(j) \oplus 1$.

Observe that when both P_{σ_1} and P_{σ_2} are honest, we can assume that $y^j_{k,\sigma}$ and $y^j_{k,\theta}$ are different from 0, otherwise, the protocol would have aborted in step 3. So this terminates the first part of the claim.

If instead P_σ is corrupted, $y^j_{k,\sigma}$ can be 0. In that case, we have that $b^j_{k,\theta} = y^j_{k,\theta} \bmod 2 = \delta_{\omega(k)}(j)$. ■

Claim 3.3. *The simulation of the first check is perfect.*

Proof of the claim. If P_{σ_1} and P_{σ_2} are both corrupted, the procedure is perfectly simulated, so we can restrict our analysis to the other cases.

If P_{σ_1} and P_{σ_2} are both honest, observe that $b^j_{k,\sigma}$ is perfectly simulated for every $(k,j) \in \mathcal{T}$. Independently on whether P_σ is corrupt, the only discrepancy between the protocol and the simulation is the value of $b^j_{k,\theta}$. However, observe that by Claim 3.2, the two values differ by $\delta_0(j) \oplus \delta_{\omega(k)}(j)$, so, the distribution of $\bigoplus_{j\in[\mathrm{dE}(k)]} b^j_{k,\theta}$ is the same in both protocol and simulation.

In the protocol, for every honest party P_{σ_i} with $i \in \{1,2\}$, we have that

$$t_{k,i} := \bigoplus_{j\in[\mathrm{dE}(k)]} t^j_{k,i} = \bigoplus_{j\in[\mathrm{dE}(k)]} (w^j_{k,i} \oplus b^j_{k,i}) = \bigoplus_{j\in[\mathrm{dE}(k)]} b^j_{k,i} \oplus \bigoplus_{j\in[\mathrm{dE}(k)]} w^j_{k,i}.$$

In other words, we know that $t_{k,\theta}$ depends only on $\bigoplus_{j\in[\mathrm{dE}(k)]} b^j_{k,i}$ and therefore, the simulation is perfect. ■

For every $(k,j) \in \mathcal{T}$ and $i \in \{1,2\}$, we define ${b'}^j_{k,i} := w^j_{k,i} \oplus t^j_{k,i}$. Observe that when P_i is honest, ${b'}^j_{k,i} = b^j_{k,i}$.

Claim 3.4. *If all the parties are honest, the first check never aborts.*

If there exists $k \in \mathcal{K}$ such that

$$\bigoplus_{j\in[\mathrm{dE}(k)]} \epsilon^j_k = 1$$

and P_{σ_0} is honest, the first check always aborts.

Proof of the claim. Since the simulation is perfect, if $\mathcal{S}_{3\text{-DPF}}$ makes the functionality abort, also $\Pi_{3\text{-DPF}}$ aborts. Observe that for every $k \in \mathcal{K}$,

$$\psi_k = t_{k,1} \oplus t_{k,2} \oplus \bigoplus_{j\in[\mathrm{dE}(k)]} (w^j_{k,1} \oplus w^j_{k,2}) = \bigoplus_{j\in[\mathrm{dE}(k)]} (t^j_{k,1} \oplus w^j_{k,1} \oplus t^j_{k,2} \oplus w^j_{k,2}).$$

If P_{σ_1} and P_{σ_2} are both honest, the value computed by P_{σ_0} for every $k \in \mathcal{K}$ is

$$\psi_k = \bigoplus_{j\in[\mathrm{dE}(k)]} (b^j_{k,\sigma} \oplus b^j_{k,\theta}) = \bigoplus_{j\in[\mathrm{dE}(k)]} (b^j_{k,\sigma} \oplus b^j_{k,\theta}) = \bigoplus_{j\in[\mathrm{dE}(k)]} (\delta_{\omega(k)}(j) \oplus 1) = 1.$$

So, the protocol never aborts.

If exactly one of P_{σ_1} and P_{σ_2} is corrupted, the value computed by P_{σ_0} for every $k \in \mathcal{K}$ is

$$\psi_k = \bigoplus_{j\in[\mathrm{dE}(k)]} (b'^j_{k,\sigma} \oplus b^j_{k,\theta}) = \bigoplus_{j\in[\mathrm{dE}(k)]} (\epsilon^j_k \oplus b^j_{k,\sigma} \oplus b^j_{k,\theta}) =$$
$$= \bigoplus_{j\in[\mathrm{dE}(k)]} \epsilon^j_k \oplus \bigoplus_{j\in[\mathrm{dE}(k)]} (b^j_{k,\sigma} \oplus b^j_{k,\theta}) = \bigoplus_{j\in[\mathrm{dE}(k)]} \epsilon^j_k \oplus 1.$$

Finally, if P_{σ_1} and P_{σ_2} are both corrupted, the value computed by P_{σ_0} for every $k \in \mathcal{K}$ is

$$\psi_k = \bigoplus_{j\in[\mathrm{dE}(k)]} (\epsilon^j_k \oplus \delta_{\omega(k)}(j) \oplus 1) = \bigoplus_{j\in[\mathrm{dE}(k)]} \epsilon^j_k \oplus 1.$$

So, if $\bigoplus_{j\in[\mathrm{dE}(k)]} \epsilon^j_k = 1$, the protocol always aborts. ■

Claim 3.5. *The second check is perfectly simulated. Moreover, if all the parties are honest, the check never fails.*

Proof of the claim. If P_{σ_1} and P_{σ_2} are both corrupted, the simulation is clearly perfect.

For every honest party P_{σ_i} with $i \in \{1, 2\}$, Φ_i is perfectly simulated. Indeed, in both the protocol and the simulation, R_i is random and unknown to the adversary, masking all the information.

Suppose that only one among P_{σ_1} and P_{σ_2} is corrupted. Let R'_σ be the value input into $\mathcal{F}_{\mathrm{MPC}}$ by P_σ. Observe that, in the protocol,

$$\Phi \oplus \Phi' = R_\sigma \oplus R'_\sigma \oplus V \cdot \Big(\omega \oplus C \cdot (\boldsymbol{t_1} \oplus \boldsymbol{t_2} \oplus \boldsymbol{w} \oplus \mathbf{1})\Big)$$

Take now $(k, j) \in \mathcal{T}$. The $(\mathrm{dE}(k) + j)$-th entry of $\boldsymbol{t_1} \oplus \boldsymbol{t_2} \oplus \boldsymbol{w} \oplus \mathbf{1}$ is

$$t^j_{k,1} \oplus t^j_{k,2} \oplus w^j_{k,1} \oplus w^j_{k,2} \oplus 1 = b'^j_{k,\sigma} \oplus b^j_{k,\theta} \oplus 1 = b'^j_{k,\sigma} \oplus b^j_{k,\sigma} \oplus \delta^j_{\omega(k)}.$$

Now, define $\boldsymbol{\epsilon}$ and $\boldsymbol{\delta}$ as the $|\mathcal{T}|$-dimensional vectors having $b'^j_{k,\sigma} \oplus b^j_{k,\sigma}$ and $\delta_{\omega(k)}(j)$ in the $(\mathrm{dE}(k) + j)$-th entry, respectively. Observe that, by Lemma 3, $C \cdot \boldsymbol{\delta} = \omega$. So we have that

$$\Phi \oplus \Phi' = R_\sigma \oplus R'_\sigma \oplus V \cdot \Big(\omega \oplus C \cdot \boldsymbol{\epsilon} \oplus C \cdot \boldsymbol{\delta}\Big) = R_\sigma \oplus R'_\sigma \oplus V \cdot C \cdot \boldsymbol{\epsilon}.$$

By substituting ω with 0 and $\delta_{\omega(k)}(j)$ with $\delta_0(j)$, we understand that the same relation holds in the simulation. That proves that Φ' is perfectly simulated.

Finally, we can prove the claim when both P_{σ_1} and P_{σ_2} are honest, using exactly the same argument. The only difference is that $R'_\sigma = R_\sigma$ and $b'^j_{k,\sigma} = b^j_{k,\sigma}$ for every $(k, j) \in \mathrm{dE}(k)$. Observe that $\boldsymbol{\epsilon} = \mathbf{0}$, hence $\Phi' = \Phi$. As a consequence, the second check always succeeds when all the parties are honest. ■

Claim 3.6. *Consider the protocol and suppose that P_σ is corrupted. Then, by the correlation robustness of H, the adversary cannot distinguish $(Y_{k,\theta}^{\omega(k)})_{k\in\mathcal{K}}$ form a random sequence of $h+1$ elements in $\{0,1\}^\lambda$.*

Proof of the claim. By Lemma 2, we know that, for every $k \in \mathcal{K}$, $T_{k,\sigma}^{\omega(k)} \oplus T_{k,\theta}^{\omega(k)} = \Delta$. Observe that Δ is uniformly distributed in $\{0,1\}^\lambda$ from every party's perspective, including the corrupted ones. Since we use a different "tag" (k,j) for every evaluation of the hash function, by the correlation robustness of H, we conclude that the values $(Y_{k,\theta}^{\omega(k)})_{k\in\mathcal{K}}$ look random and independent from every party's perspective except P_θ itself. ■

Claim 3.7. *Consider the protocol and suppose that P_{σ_1} and P_{σ_2} are both honest. Then, by the correlation robustness of H, $(\boldsymbol{Y_{k,1}}, \boldsymbol{Y_{k,2}})_{k\in\mathcal{K}}$ are indistinguishable from $h+1$ random pairs of vectors over $\{0,1\}^\lambda$ having coinciding entries for every $j \neq \omega(k)$.*

Proof of the claim. By Lemma 2, we know that for every $j \neq \omega(k)$, $T_{k,1}^j = T_{k,2}^j$. Hence, for all such (k,j), we have $Y_{k,1}^j = Y_{k,2}^j$. We also know that $T_{k,1}^{\omega(k)} \oplus T_{k,2}^{\omega(k)} = \Delta$. Observe that Δ is uniformly distributed in $\{0,1\}^\lambda$ from every party's perspective.

Consider the vector $\boldsymbol{CT_1}$ having $T_{k,1}^j$ in the $(\mathrm{dE}(k)+j)$-th entry for every $(k,j) \in \mathcal{T}$ and observe that $\boldsymbol{CT_1}$ is obtained from $\boldsymbol{T_1}$ by applying a linear function. The corresponding matrix B consists of all the rows of B_k for every $k \in \mathcal{K}$. Such matrix is not, however, full-rank. Indeed, the sum of all the rows of B_k is the vector entirely made of ones for every $k \in \mathcal{K}$. Let $\widehat{B}$ be the matrix derived from B by removing the last row of B_k for every $k \in \mathcal{K}$ and adding the row entirely made of ones.

For every $(k,j) \in \mathcal{T}$ with $j \neq \mathrm{dE}(k) - 1$, consider the difference between the $c_1 := \big(j \cdot \mathrm{dE}(k)\big)$-th and the $c_2 := \big((j+1) \cdot \mathrm{dE}(k)\big)$-th column of $\widehat{B}$. Let it be $\boldsymbol{d_{k,j}}$. For every $l \neq k$, we have that the c_1-th and the c_2-th column of B_l coincide. Indeed, if $l > k$

$$c_1 < c_2 = (j+1) \cdot \mathrm{dE}(k) < \mathrm{dE}(k+1) \leq \mathrm{dE}(l).$$

Remember that the first $\mathrm{dE}(l)$ columns of B_l are all identical. If instead $l < k$, we know that $\mathrm{dE}(l+1)$ divides $\mathrm{dE}(k)$ and therefore also c_1 and c_2. Since B_l is split into several identical blocks of $\mathrm{dE}(l+1)$ columns, the c_1-th and c_2-th column of B_l must coincide. If instead we consider B_k, the c_1-th and c_2-th column are unit vectors having a 1 in the j-th and $(j+1)$-th position respectively.

To summarise, all the entries of $\boldsymbol{d_{k,j}}$ are zeros with the only exceptions of the $\big(\mathrm{dE}(k)+j\big)$-th one, being 1, and the $\big(\mathrm{dE}(k)+j+1\big)$-th one, being -1. All these vectors are linearly independent as they can be rearranged into a triangular matrix. Moreover, they are all independent of any column of $\widehat{B}$, as they all have the last entry equal to 0. Therefore, we have obtained $|\mathcal{T}|-h$ independent vectors belonging to the space generated by the columns of $\widehat{B}$. Observe that $|\mathcal{T}| - h$ is

exactly the number of rows in $\widehat{B}$, hence, $\widehat{B}$ is full-rank. Let $\widehat{T} := \bigoplus_{j\in[N]} T_1^j$. Since $\boldsymbol{T_1}$ is random, we conclude that the values $\widehat{T}, (T_{k,1}^j)_{j\neq \mathrm{dE}(k)-1}$ are random and independent elements in $\{0,1\}^\lambda$.

It is now easy to see that, for every $k \in \mathcal{K}$,

$$T_{k,1}^{\mathrm{dE}(k)-1} = \widehat{T} \oplus \bigoplus_{j\neq \mathrm{dE}(k)-1} T_{k,1}^j.$$

Finally, we can conclude the proof of the claim by the correlation robustness of H. The procedure actually needs three hybrids. In the first one, relying of the secrecy of Δ, for every $k \in \mathcal{K}$, we substitute $Y_{k,1}^{\omega(k)}$ with a random element in $\{0,1\}^\lambda$. In the second one, relying on the secrecy of $\widehat{T}$, for every $k \in \mathcal{K}$, we substitute $Y_{k,1}^{\mathrm{dE}(k)-1}$ and, if $\mathrm{dE}(k)-1 \neq \omega(k)$, $Y_{k,2}^j$ with the same random element in $\{0,1\}^\lambda$. In the last one, relying on the fact that $(T_{k,1}^j)_{j\neq \mathrm{dE}(k)-1}$ are random and independent, for every $j \neq \mathrm{dE}(k)-1$, we substitute $Y_{k,1}^j$ and, if $j \neq \omega(k)$, $Y_{k,2}^j$ with the same random element in $\{0,1\}^\lambda$. ■

Claim 3.8. *Consider the protocol and suppose that P_{σ_0} is honest. For every $(k,j) \in \mathcal{T}$ and $i \in \{1,2\}$, define $b'^j_{k,i} := t_{k,i}^j \oplus w_{k,i}^j$. If $b'^j_{k,1} = b'^j_{k,2}$, one element among $S_{k,0,0}^j$ and $S_{k,0,1}^j$ is uniformly distributed in $\{0,1\}^\lambda$ from every party's perspective except P_{σ_0}.*

Proof of the claim. Observe that the adversary can obtain information about $X_k^j[t_{k,1}^j \oplus 1]$ from exactly one the values $Z_k^j[0]$ and $Z_k^j[1]$. Specifically, the former if $b'^j_{k,1} = 1$, the latter if $b'^j_{k,1} = 0$. The analysis of the two cases is identical, for simplicity, we restrict to the first situation. The information in $Z_k^j[0]$ is masked by $W_k^j[w_{k,2}^j]$. Observe that $w_{k,2}^j = t_{k,2}^j \oplus 1$ as $b'^j_{k,2} = b'^j_{k,1} = 1$. The value $W_k^j[t_{k,2}^j \oplus 1]$ is random in $\{0,1\}^\lambda$ and is known to no party other than P_{σ_0}. Hence, $Z_k^j[0]$ does not contain any information about $X_k^j[t_{k,1}^j \oplus 1]$ either. That proves that $X_k^j[t_{k,1}^j \oplus 1]$ is uniformly distributed in $\{0,1\}^\lambda$ from every party's perspective except P_{σ_0}. ■

Claim 3.9. *If the first check succeeds, for every honest party P_{σ_i} and for every $k \in \mathcal{K}$, there exists $j \in [\mathrm{dE}(k)]$ and $b \in \{0,1\}$ such that $S_{k,i,b}^j$ is random in $\{0,1\}^\lambda$ from every party's perspective except P_{σ_i}.*

Proof of the claim. If $i \in \{1,2\}$, by Claim 3.6, the element $S_{k,i,1}^{\omega(k)} = Y_{k,i}^{\omega(k)}$ is random in $\{0,1\}^\lambda$ from every other party's perspective.

If P_{σ_0} is honest, the first check passes if and only if $\psi_k = 1$ for every $k \in \mathcal{K}$. Let $b'^j_{k,i} := t_{k,i}^j \oplus w_{k,i}^j$ for every $i \in \{1,2\}$ and $(k,j) \in \mathcal{T}$. Observe that

$$\psi_k = t_{k,1} \oplus t_{k,2} \oplus \bigoplus_{j\in[\mathrm{dE}(k)]} (w_{k,1}^j \oplus w_{k,2}^j) = \bigoplus_{j\in[\mathrm{dE}(k)]} (b'^j_{k,1} \oplus b'^j_{k,2}).$$

Hence, we know that there exists $j \in [\mathrm{dE}(k)]$ such that $b'^j_{k,1} = b'^j_{k,2}$ ($\mathrm{dE}(k)$ is always even). The result follows from Claim 3.8. ■

Claim 3.10. *Let $p := |\mathbb{F}|$. Since $|p - 2^\lambda|/2^\lambda$ is negligible, the simulation of the basic case is unconditionally secure.*

Proof of the claim. Consider the protocol. By Claim 3.9, we know that there exists $j \in [2]$ such that at least one among $X^j_{-1}[0], X^j_{-1}[1], X^j_{-1}[2], X^j_{-1}[3], Y^j_{-1,1}, Y^j_{-1,2}$ is random in $\{0,1\}^\lambda$ from the adversary's perspective. Denote such element by X_{-1}, let P_{σ_i} be the honest party owning it.

When the encoding map Enc is applied to X_{-1}, the distribution of the result x is statistically indistinguishable from the uniform distribution over $\mathbb{F}$. Observe that x is then added with other values, obtaining z_i. Since all the elements in the sum are independent of x, the adversary cannot distinguish between the distribution of z_i and the uniform distribution over $\mathbb{F}$.

In conclusion the distribution of CW in the protocol is indistinguishable from the distribution that outputs 0 if $\beta = 0$ and a random element in $\mathbb{F}^\times$, otherwise. This is exactly the distribution used in the simulation.

Observe that in the protocol, the procedure can always fail due to a zero denominator. Since z_i looks random in $\mathbb{F}$ from the adversary's perspective, the probability of such event is negligible. ■

Claim 3.11. *Since G_k is a PRG for every $k \in [h]$, the simulation of the correction word is computationally secure.*

Proof of the claim. Consider the protocol and let $k \in [h]$. By Claim 3.9, we know that for every honest party P_{σ_i}, there exists $j \in [\mathrm{dE}(k)]$ such at least one among $S^j_{k,i,0}$ and $S^j_{k,i,1}$ is random in $\{0,1\}^\lambda$ from the adversary's perspective. Denote such element by $S_{k,i}$.

When the PRG G_k is applied to $S_{k,i}$, the distribution of the result is computationally indistinguishable from the uniform distribution over $\mathbb{F}^{\mathrm{dE}(k)}$. Observe that the expansion of $S_{k,i}$ is one of the terms of $\boldsymbol{CW_{k,i}}$. Hence, the distribution of $\boldsymbol{CW_{k,i}}$ is indistinguishable from the uniform distribution over $\mathbb{F}^{\mathrm{dE}(k)}$. The latter is the distribution used in the simulation. ■

Clearly, the distribution of $\boldsymbol{\chi}$ is the same in both the protocol and the simulation.

Claim 3.12. *Suppose P_{σ_0} is honest and either P_{σ_1} or P_{σ_2} is corrupted. Assume that there exists $m \in \mathcal{K}$ with the following properties*

- $\epsilon_m^{\omega(m)} = 1$
- *There exists $l \neq \omega(m)$ such that $\epsilon_m^l = 1$ and $\epsilon_m^j = 0$ for every $j \neq l, \omega(m)$.*

Then, the second check fails with overwhelming probability.

Proof of the claim. The value Φ computed by P_{σ_0} is

$$\Phi = R_1 \oplus R_2 \oplus V \cdot C \cdot (\boldsymbol{t_1} \oplus \boldsymbol{t_2} \oplus \boldsymbol{w} \oplus \mathbf{1}).$$

For every $(k, j) \in \mathcal{T}$, the $(\mathrm{dE}(k) + j)$-th entry of $\boldsymbol{s} := \boldsymbol{t_1} \oplus \boldsymbol{t_2} \oplus \boldsymbol{w} \oplus \mathbf{1}$ is

$$t^j_{k,1} \oplus t^j_{k,2} \oplus w^j_{k,1} \oplus w^j_{k,2} \oplus 1 = \epsilon^j_k \oplus \delta_{\omega(k)}(j)$$

if both P_{σ_1} and P_{σ_2} are corrupted,

$$t^j_{k,1} \oplus t^j_{k,2} \oplus w^j_{k,1} \oplus w^j_{k,2} \oplus 1 = b'^j_{k,\sigma} \oplus b^j_{k,\theta} \oplus 1 = \\ = \epsilon^j_k \oplus b^j_{k,\sigma} \oplus b^j_{k,\theta} \oplus 1 = \epsilon^j_k \oplus \delta_{\omega(k)}(j)$$

if only P_σ is corrupted[8].

Split now $\boldsymbol{s}$ into $h+1$ blocks indexed by $k \in \mathcal{K}$, the k-th one, denoted by $\boldsymbol{s_k}$, having $\mathrm{dE}(k)$ bits. Observe that $\boldsymbol{s_m}$ is a unit vector having special position l and non-zero element 1.

Let $\boldsymbol{z} := C \cdot \boldsymbol{s}$. Similarly to Lemma 3, we split $\boldsymbol{z}$ into $h+1$ blocks indexed by $k \in \mathcal{K}$, the k-th one, denoted by $\boldsymbol{z_k}$, having 2^k bits (1 bit if $k = -1$). Observe that $\boldsymbol{z}_m$ coincides with the bit representation of l.

Let R'_1 and R'_2 be the values input by the parties P_{σ_1} and P_{σ_2} in $\mathcal{F}_{\mathrm{MPC}}$, we define $E := R_1 \oplus R_2 \oplus R'_1 \oplus R'_2$. The second check passes if and only if $\Phi \oplus \Phi' = 0$ and therefore if and only if $E = V \cdot (\boldsymbol{z} \oplus \omega)$. The matrix V is random in $\mathbb{F}_2^{\lambda \times \log(N)}$ and independent of E. Indeed, it is sampled after the adversary input R'_1 and R'_2 in $\mathcal{F}_{\mathrm{MPC}}$. Moreover, $\boldsymbol{z} \oplus \omega \neq 0$ as the m-th blocks $\boldsymbol{z}_m$ and $\omega(m)$ are different. Hence, the probability that $E = V \cdot (\boldsymbol{z} \oplus \omega)$ is $2^{-\lambda}$. ■

Claim 3.13. *Consider the protocol and suppose that P_{σ_0} is honest. Assume the existence of $m \in \mathcal{K}$ for which there are $r \neq l$ such that $b'^r_{m,1} = b'^r_{m,2}$ and $b'^l_{m,1} = b'^l_{m,2}$. Then, the distribution of ρ is computationally indistinguishable from the uniform distribution over $\mathbb{F}$ and the third check fails with overwhelming probability.*

Proof of the claim. We prove this claim in the hybrid in which the expansion via a PRG G of every seed S unknown to the adversary is substituted with a random vector of appropriate length. However, we will still denote such vector by $G(S)$.

In this proof, we will continuously rely on the fact that, for every $k \in [h]$

$$\boldsymbol{CW_{k,0}} = \boldsymbol{v_{k,0}} - \sum_{j \in [\mathrm{dE}(k)]} \boldsymbol{v^j_{k+1,0}}.$$

This is a consequence of the fact that $y^j_{k,0} = 0$ for every $j \in [\mathrm{dE}(k)]$.

By Claim 3.8, we know that one value among $S^r_{m,0,0}$ and $S^r_{m,0,1}$ and one value among $S^l_{m,0,0}$ and $S^l_{m,0,1}$ is uniformly distributed in $\{0,1\}^\lambda$ from every other party's perspective, including the corrupted parties. We denote the first one by S^r_m and the second one by S^l_m. Clearly, S^r_m and S^l_m are also mutually independent.

If $m = -1$, S^r_m is independent of CW. Indeed, S^l_m hides the information with unconditional security. When instead $m \geq 0$, no information concerning $G_m(S^r_m)$ is leaked by $\boldsymbol{CW_{m,0}}$ as $G_m(S^l_m)$ protects its privacy with computational security.

If $m = -1$, we understand that $v^r_{0,0}$ is unconditionally indistinguishable from a random element in $\mathbb{F}$. Observe that $v^{1-r}_{0,0}$ is however not independent of $v^r_{0,0}$, as

[8] The last equality follows from Claim 3.2.

the sum of the entries of the vector $\boldsymbol{v_{0,0}}$ is bounded to be $z_0 \cdot CW$. The important fact is that this is a linear dependency and $z_0 \cdot CW$ is independent of $v^k_{0,0}$.

If $m \geq 0$, we know that $G_m(S^r_m)$ is a random vector in $\mathbb{F}^{\mathrm{dE}(m)}$. We understand that $\boldsymbol{v^r_{m+1,0}}$ is a random vector in $\mathbb{F}^{\mathrm{dE}(m)}$ too. Again, that does not mean that $\boldsymbol{v^r_{m+1,0}}$ is independent of $(\boldsymbol{v^j_{m+1,0}})_{j \neq r}$. Indeed, the sum of all the vectors $\boldsymbol{v^j_{m+1,0}}$ is bounded to be equal to $\boldsymbol{v_{m,0} - CW_{m,0}}$. The latter is independent of $\boldsymbol{v^r_{m+1,0}}$. Moreover, $\boldsymbol{v^r_{m+1,0}}$ is independent of $(\boldsymbol{v^j_{m+1,0}})_{j \neq r,l}$.

We now proceed by induction. Suppose that we proved the existence of values r_k, l_k and $V_k \in \mathbb{F}$ such that $v^{r_k}_{k,0}$ is a random element in $\mathbb{F}$ independent of $(v^j_{k,0})_{j \neq r_k, l_k}, V_k$. Moreover, suppose that

$$\sum_j v^j_{k,0} = V_k.$$

Observe that these properties are satisfied for $k = m + 1$.

By Claim 3.9, we know that there exists $\tau \in \mathrm{dE}(k)$ and $B \in \{0,1\}$ such that $S^\tau_{k,0,B}$ is uniformly distributed in $\{0,1\}^\lambda$ from the adversary's perspective. As a consequence, $G_k(S^\tau_{k,0,b})$ is a random element in $\mathbb{F}^{\mathrm{dE}(k)}$. We conclude that $\boldsymbol{CW_{k,0}}$ is independent of

$$\Big(G_k(S^j_{k,0,b})\Big)_{(j,b) \neq (\tau,B)}$$

Let now $G^i_k(S^j_{k,0,b})$ denote the i-th entry in $G_k(S^j_{k,0,b})$. Since $v^{r_k}_{k,0}$ is random in $\mathbb{F}$, $G^{r_k}_k(S^\tau_{k,0,B})$ is independent of $\boldsymbol{CW_{k,0}}$ too, and therefore it is random in $\mathbb{F}$ from the adversary's perspective.

Define now $r_{k+1} := \tau \cdot \mathrm{dE}(k) + r_k$ and $l_{k+1} := \tau \cdot \mathrm{dE}(k) + l_k$. Since $G^{r_k}_k(S^\tau_{k,0,B})$ is random, the r_k-th entry of $\boldsymbol{v^\tau_{k+1,0}}$ is random in $\mathbb{F}$. Such entry corresponds to $v^{r_{k+1}}_{k+1,0}$.

Let now U_k denote the sum of the entries in $\boldsymbol{CW_{k,0}}$ and observe that

$$\sum_j v^j_{k+1,0} = V_k - U_k.$$

Since the $G^{r_k}_k(S^\tau_{k,0,B})$ is independent[9] of (V_k, U_k), we conclude that $v^{r_{k+1}}_{k+1,0}$ is independent of $V_{k+1} := V_k - U_k$. Finally, since $v^{r_k}_{k,0}$ is independent of $(v^j_{k,0})_{j \neq r_k, l_k}$, it is easy to see that $v^{r_{k+1}}_{k+1,0}$ is independent of $(v^j_{k+1,0})_{j \neq r_{k+1}, l_{k+1}}, V_{k+1}$.

By induction over k, we can therefore conclude that there exist r_h, l_h and V_h such that $v^{r_h}_{h,0}$ is random in $\mathbb{F}$ and independent of $(v^j_{h,0})_{j \neq r_h, l_h}, V_h$ from the adversary's perspective. Moreover,

$$v^{l_h}_{h,0} = V_h - \sum_{j \neq l_h} v^j_{h,0}.$$

[9] This fact is a consequence of the independence between $v^{r_k}_{k,0}$ and V_k. Observe indeed that both $v^{r_{k+1}}_{k+1,0}$ and $v^{r_k}_{k,0}$ must be added to obtain U_k, the latter masks all the information about the former.

Observe now that

$$\zeta_0 = \langle \boldsymbol{\chi}, \boldsymbol{v_{h,0}} \rangle = \sum_{j \in [N]} \chi_j \cdot v^j_{h,0} = \sum_{j \neq l_h} (\chi_j - \chi_{l_h}) \cdot v^j_{h,0} + \chi_{l_h} \cdot V_h.$$

So, when $\chi_{r_h} \neq \chi_{l_h}$, event that occurs with overwhelming probability $1 - |\mathbb{F}|^{-1}$, no PPT adversary can distinguish ζ_0 from a random element in $\mathbb{F}$. Since ζ_0 is independent of $\zeta_1, \zeta_2, d_1, d_2$ and β, the distribution of ρ is indistinguishable from the uniform distribution over $\mathbb{F}$. Hence, the third check aborts with overwhelming probability. ■

Claim 3.14. *If P_{σ_0} is honest and there exists $(k, j) \in \mathcal{T}$ such that $\epsilon^j_k = 1$, no PPT adversary can distinguish between protocol and simulation.*

Proof of the claim. If there exists $m \in \mathcal{K}$, for which the number of $j \in [\mathrm{dE}(m)]$ such that $\epsilon^j_m = 1$ is odd, the first check aborts in both the protocol and the simulation.

If there exists $m \in \mathcal{K}$, for which there are exactly two positions $j, l \in [\mathrm{dE}(m)]$ for which $\epsilon^l_m = \epsilon^j_m = 1$ and $j = \omega(m)$, by Claim 3.12, the protocol aborts at the second check with overwhelming probability. We have proven that $\mathcal{S}_{3\text{-DPF}}$ perfectly models the second check, hence, the same happens in the simulation.

In all other cases, there exist $m \in \mathcal{K}$ and $r, l \in [\mathrm{dE}(m)]$, both different from $\omega(m)$, such that $\epsilon^l_m = \epsilon^r_m = 1$. If both P_{σ_1} and P_{σ_2} are corrupted, it means that

$$b'^l_{m,1} \oplus b'^l_{m,2} = \epsilon^l_m \oplus \delta_{\omega(m)}(l) \oplus 1 = 0,$$
$$b'^r_{m,1} \oplus b'^r_{m,2} = \epsilon^r_m \oplus \delta_{\omega(m)}(r) \oplus 1 = 0.$$

If instead there exists only one corrupted party among P_{σ_1} and P_{σ_2},by Claim 3.2, we have

$$b'^l_{m,\theta} = b^l_{m,\theta} = \delta_{\omega(m)}(l) \oplus b^l_{m,\sigma} \oplus 1 = \delta_{\omega(m)}(l) \oplus \epsilon^l_m \oplus b'^l_{m,\sigma} \oplus 1 = b'^l_{m,\sigma},$$
$$b'^r_{m,\theta} = b^r_{m,\theta} = \delta_{\omega(m)}(r) \oplus b^r_{m,\sigma} \oplus 1 = \delta_{\omega(m)}(r) \oplus \epsilon^r_m \oplus b'^r_{m,\sigma} \oplus 1 = b'^r_{m,\sigma}.$$

In both cases, there exist $m \in \mathcal{K}$ and $r \neq l$ such that $b'^r_{m,1} = b'^r_{m,2}$ and $b'^l_{m,1} = b'^l_{m,2}$.

To summarise, either both the protocol and the simulation abort in the first or second check, or, by Claim 3.13, they both abort in the third check. In such situation, ρ is indistinguishable from a random element in $\mathbb{F}$.

■

Claim 3.15. *Consider the protocol. For every $k \in [h+1]$, define the random variable $\widehat{\boldsymbol{v}}_{\boldsymbol{k},\boldsymbol{\omega}}$ as $\widehat{\boldsymbol{v}}_{\boldsymbol{k},\boldsymbol{\eta}}$ for $\eta = \omega$ in $\mathcal{S}_{3\text{-}DPF}$.*

Then, we have

$$\widehat{\boldsymbol{v}}_{\boldsymbol{h},\boldsymbol{\omega}} + \sum_{\substack{i \in [3] \\ \sigma_i \in \mathcal{H}}} \boldsymbol{v_{h,i}} = (\underbrace{0, 0, \ldots, \overbrace{0}^{\omega}}_{N}, \beta, 0, 0, \ldots, 0).$$

Proof of the claim. Observe that for every $\sigma_i \in \mathcal{H}$, $j \neq \omega(k)$ and $b \in \{0,1\}$, the seed $S^{j,\omega}_{k,i,b}$ used to compute $\widehat{\boldsymbol{v}}_{\boldsymbol{h},\boldsymbol{\omega}}$ coincides with $S^{j}_{k,i,b}$. The same occurs for $y^{j,\omega}_{k,i}$ and $y^{j}_{k,i}$.

For every $k \in \mathcal{K}$, we define

$$\widehat{\omega}(k+1) := \omega(-1) + \sum_{i=0}^{k} \omega(i) \cdot \mathrm{dE}(i).$$

Observe that $\widehat{\omega}(h) = \omega$.

We proceed by induction over k. Consider the base case $k = 0$. Recalling what we have just observed about the seeds, it is straightforward to see that for $j \neq \omega(-1)$

$$\widehat{v}^{j}_{0,\omega} + \sum_{\substack{i\in[3]\\ \sigma_i\in\mathcal{H}}} v^{j}_{0,i} = 0.$$

As a matter of fact,

$$v^{j}_{0,i} = CW \cdot \Big(u^0_i \cdot \mathrm{Enc}(S^{j}_{-1,i,0}) + u^1_i \cdot \mathrm{Enc}(S^{j}_{-1,i,1})\Big).$$

Now, observe that

$$\sum_{\substack{i\in[3]\\ \sigma_i\in\mathcal{H}}} \sum_{l=0}^{1} v^{l}_{0,i} = CW \cdot \sum_{\substack{i\in[3]\\ \sigma_i\in\mathcal{H}}} z_i = CW \cdot \Big(\frac{\beta}{CW} - z'\Big) = \beta - CW \cdot z'.$$

Therefore,

$$\widehat{v}^{\omega(-1)}_{0,\omega} + \sum_{\substack{i\in[3]\\ \sigma_i\in\mathcal{H}}} v^{\omega(-1)}_{0,i} = CW \cdot z' - \widehat{v}^{j}_{0,\omega} + \beta - CW \cdot z' - \sum_{\substack{i\in[3]\\ \sigma_i\in\mathcal{H}}} v^{j}_{0,i} = \beta.$$

Suppose that we have proven that for a certain $k \in [h]$

$$\widehat{\boldsymbol{v}}_{\boldsymbol{k},\boldsymbol{\omega}} + \sum_{\substack{i\in[3]\\ \sigma_i\in\mathcal{H}}} \boldsymbol{v}_{\boldsymbol{k},\boldsymbol{i}} = (\overbrace{0,0,\ldots,0}^{\widehat{\omega}(k)},\underbrace{\beta,0,0,\ldots,0}_{\mathrm{dE}(k)}) =: \widehat{\boldsymbol{u}}_{\boldsymbol{k}}.$$

It is immediate to see that for $j \neq \omega(k)$, we have

$$\widehat{\boldsymbol{v}}^{\boldsymbol{j}}_{\boldsymbol{k+1},\boldsymbol{\omega}} + \sum_{\substack{i\in[3]\\ \sigma_i\in\mathcal{H}}} \boldsymbol{v}^{\boldsymbol{j}}_{\boldsymbol{k+1},\boldsymbol{i}} = \boldsymbol{0}.$$

As a matter of fact,

$$\boldsymbol{v}^{\boldsymbol{j}}_{\boldsymbol{k+1},\boldsymbol{i}} = u^0_i \cdot G_k(S^{j}_{k,i,0}) + u^1_i \cdot G_k(S^{j}_{k,i,1}) + y^{j}_{k,i} \cdot \boldsymbol{CW}_{\boldsymbol{k}}.$$

Now, observe that

$$\sum_{\substack{i\in[3]\\ \sigma_i\in\mathcal{H}}}\sum_{j\in[\mathrm{dE}(k)]} \boldsymbol{v}^{\boldsymbol{j}}_{\boldsymbol{k+1,i}} = \sum_{\substack{i\in[3]\\ \sigma_i\in\mathcal{H}}}\Big(\boldsymbol{v_{k,i}} - \boldsymbol{CW_{k,i}} + \sum_{j\in[\mathrm{dE}(k)]} y^j_{k,i}\cdot \boldsymbol{CW_k}\Big) =$$

$$= \widehat{\boldsymbol{u}}_{\boldsymbol{k}} - \widehat{\boldsymbol{v}}_{\boldsymbol{k,\omega}} - \boldsymbol{CW_k} + \boldsymbol{CW'_k} + \sum_{\substack{i\in[3]\\ \sigma_i\in\mathcal{H}}}\sum_j y^j_{k,i}\cdot \boldsymbol{CW_k}$$

By Lemma 2, we know that

$$\sum_{\substack{i\in[3]\\ \sigma_i\in\mathcal{H}}}\sum_j y^j_{k,i} + \sum_{\substack{i\in[3]\\ \sigma_i\in\mathcal{C}}}\sum_j y^{j,\omega}_{k,i} = 1.$$

Hence, we have that

$$\sum_{\substack{i\in[3]\\ \sigma_i\in\mathcal{H}}}\sum_{j\in[\mathrm{dE}(k)]} \boldsymbol{v}^{\boldsymbol{j}}_{\boldsymbol{k+1,i}} = \widehat{\boldsymbol{u}}_{\boldsymbol{k}} - \widehat{\boldsymbol{v}}_{\boldsymbol{k,\omega}} + \boldsymbol{CW'_k} - \sum_{\substack{i\in[3]\\ \sigma_i\in\mathcal{C}}}\sum_j y^{j,\omega}_{k,i}\cdot \boldsymbol{CW_k} =$$

$$= \widehat{\boldsymbol{u}}_{\boldsymbol{k}} - \sum_j \widehat{\boldsymbol{v}}^{\boldsymbol{j}}_{\boldsymbol{k+1,\omega}}$$

We can finally conclude that

$$\widehat{\boldsymbol{v}}^{\boldsymbol{\omega(k)}}_{\boldsymbol{k+1,\omega}} + \sum_{\substack{i\in[3]\\ \sigma_i\in\mathcal{H}}} \boldsymbol{v}^{\boldsymbol{\omega(k)}}_{\boldsymbol{k+1,i}} = \widehat{\boldsymbol{u}}_{\boldsymbol{k}}.$$

It is now immediate to see that

$$\widehat{\boldsymbol{v}}_{\boldsymbol{k+1,\omega}} + \sum_{\substack{i\in[3]\\ \sigma_i\in\mathcal{H}}} \boldsymbol{v}_{\boldsymbol{k+1,i}}$$

is a unit vector with β as non-zero value. Furthermore, the special position becomes $\mathrm{dE}(k)\cdot\omega(k) + \widehat{\omega}(k) = \widehat{\omega}(k+1)$. ■

Claim 3.16. *If P_{σ_0} is corrupted or if there exists no $(k,j)\in\mathcal{T}$ such that $\epsilon^j_k = 1$, the third check is perfectly simulated.*

Proof of the claim. In the protocol, the third check does not abort if and only if $\rho = 0$.

Let $\boldsymbol{U}^{\boldsymbol{\beta}}_{\boldsymbol{\omega}}$ denote the N-dimensional unit vector having special position ω and non-zero element β. Observe that

$$\begin{aligned}
\rho =& \zeta' + \sum_{\substack{i\in[3]\\ \sigma_i\in\mathcal{H}}} \langle \boldsymbol{\chi}, \boldsymbol{v_{h,i}}\rangle - \Big(d' + \sum_{\substack{i=1\\ \sigma_i\in\mathcal{H}}}^{2} \langle \boldsymbol{\chi}, \boldsymbol{y_i}\rangle\Big)\cdot\beta = \\
=& \zeta' + \langle \boldsymbol{\chi}, \boldsymbol{U^{\beta}_{\omega}}\rangle - \langle \boldsymbol{\chi}, \widehat{\boldsymbol{v}}_{\boldsymbol{h,\omega}}\rangle - (d' + \langle \boldsymbol{\chi}, \boldsymbol{y_1} + \boldsymbol{y_2}\rangle - d)\cdot\beta = \\
=& \zeta' + \chi_\omega\cdot\beta - \zeta'_\omega - (d' - d + \chi_\omega)\cdot\beta = \\
=& \zeta' - \zeta'_\omega - (d' - d)\cdot\beta.
\end{aligned}$$

Therefore, if $d = d'$, the check passes if and only if $\zeta' = \zeta'_\omega$. If instead, $d \neq d'$, the check passes if and only if $\beta = (\zeta' - \zeta'_\omega)/(d' - d)$. Exactly the same happens in the simulation. When the check passes $\mathcal{S}_{3\text{-DPF}}$ sends $\rho = 0$ to the adversary. Moreover, when the check fails, $\mathcal{S}_{3\text{-DPF}}$ receives the information necessary to recompute ρ from the functionality, perfectly simulating the check. ■

Claim 3.17. *Consider the simulation, with overwhelming probability there exists no values $\eta, \eta' \in I'''$ such that $\widehat{\boldsymbol{v}}_{\boldsymbol{h},\boldsymbol{\eta}} \neq \widehat{\boldsymbol{v}}_{\boldsymbol{h},\boldsymbol{\eta'}}$.*

Proof of the claim. Suppose that we have $\eta, \eta' \in [N]$ such that $\widehat{\boldsymbol{v}}_{\boldsymbol{h},\boldsymbol{\eta}} \neq \widehat{\boldsymbol{v}}_{\boldsymbol{h},\boldsymbol{\eta'}}$. The two values can simultaneously belong to I''' only if $\langle \boldsymbol{\chi}, \widehat{\boldsymbol{v}}_{\boldsymbol{h},\boldsymbol{\eta}} \rangle = \langle \boldsymbol{\chi}, \widehat{\boldsymbol{v}}_{\boldsymbol{h},\boldsymbol{\eta'}} \rangle$ and, therefore, only if $\langle \boldsymbol{\chi}, \widehat{\boldsymbol{v}}_{\boldsymbol{h},\boldsymbol{\eta}} - \widehat{\boldsymbol{v}}_{\boldsymbol{h},\boldsymbol{\eta'}} \rangle = 0$.

Observe that $\boldsymbol{\chi}$ is random in $\mathbb{F}^N$ and independent of $\widehat{\boldsymbol{v}}_{\boldsymbol{h},\boldsymbol{\eta}}, \widehat{\boldsymbol{v}}_{\boldsymbol{h},\boldsymbol{\eta'}}$. Since $\widehat{\boldsymbol{v}}_{\boldsymbol{h},\boldsymbol{\eta}} - \widehat{\boldsymbol{v}}_{\boldsymbol{h},\boldsymbol{\eta'}} \neq \boldsymbol{0}$, the probability that $\langle \boldsymbol{\chi}, \widehat{\boldsymbol{v}}_{\boldsymbol{h},\boldsymbol{\eta}} - \widehat{\boldsymbol{v}}_{\boldsymbol{h},\boldsymbol{\eta'}} \rangle = 0$ is $|\mathbb{F}|^{-1}$, which is negligible.

Since there exists a polynomial number of pairs $(\eta, \eta') \in [N]^2$, the result is proven by the union bound. ■

Claim 3.18. *If all the parties are honest, the third check never aborts and*

$$\sum_{i \in [3]} \boldsymbol{v}_{\boldsymbol{h},\boldsymbol{i}} = (\overbrace{0, 0, \ldots, 0}^{\omega}, \beta, \underbrace{0, 0, \ldots, 0}_{N}).$$

Proof of the claim. Take $(k, j) \in \mathcal{T}$ with $j \neq \omega(k)$. By Claim 3.7, we know that $S^j_{k,1,1} = S^j_{k,2,1}$. We now prove that $\{S^j_{k,0,0}, S^j_{k,0,1}\} = \{S^j_{k,1,0}, S^j_{k,2,0}\}$.

Observe that $S^j_{k,1,0} \in \{S^j_{k,0,0}, S^j_{k,0,1}\}$. By Claim 3.2, we know that $b^j_{k,2} = b^j_{k,1} \oplus 1$. Therefore,

$$\begin{aligned} S^j_{k,2,0} &= X^j_k[3] = W^j_k \oplus Z^j_k[b^j_{k,2}] = \\ &= W^j_k \oplus W^j_k[w^j_{k,2} \oplus b^j_{k,2}] \oplus X^j_k[w^j_{k,1} \oplus b^j_{k,2}] = \\ &= W^j_k \oplus W^j_k[t^j_{k,2}] \oplus X^j_k[w^j_{k,1} \oplus b^j_{k,1} \oplus 1] = \\ &= X^j_k[t^j_{k,1} \oplus 1]. \end{aligned}$$

It is now immediate to see that for every $j \neq \omega(k)$, $\widehat{\boldsymbol{v}}^{\boldsymbol{j}}_{\boldsymbol{k+1},\boldsymbol{\omega}} = \boldsymbol{0}$. This is consequence of the fact that when a party adds a term obtained from a certain value $S^j_{k,i,b}$, the other party holding $S^j_{k,i,b}$ will subtract the same term.

Since no party is corrupted, $z' = 0$ and $\boldsymbol{CW'_k} = \boldsymbol{0}$. By recursion on k, it is therefore possible to deduce that $\widehat{\boldsymbol{v}}^{\boldsymbol{\omega(k)}}_{\boldsymbol{k+1},\boldsymbol{\omega}} = \boldsymbol{0}$ for every $k \in \mathcal{K}$. That proves that $\widehat{\boldsymbol{v}}_{\boldsymbol{h},\boldsymbol{\omega}} = \boldsymbol{0}$.

Since $\zeta' = 0$, the simulator will never make the functionality abort. Previously, we have also proven that the third check is perfectly simulated, so the same happens in the protocol. Finally, by Claim 3.15, we know that

$$\sum_{i \in [3]} \boldsymbol{v}_{\boldsymbol{h},\boldsymbol{i}} = \widehat{\boldsymbol{v}}_{\boldsymbol{h},\boldsymbol{\omega}} + \sum_{i \in [3]} \boldsymbol{v}_{\boldsymbol{h},\boldsymbol{i}} = (\overbrace{0, 0, \ldots, 0}^{\omega}, \beta, \underbrace{0, 0, \ldots, 0}_{N}).$$

■

Claim 3.19. *If the protocol does not abort, no PPT adversary can distinguish between the outputs of the honest parties in the protocol and in the simulation.*

Proof of the claim. If the protocol does not abort, with overwhelming probability $\epsilon_k^j = 0$ for every $(k, j) \in \mathcal{T}$. Moreover, we know that $\omega \in I'''$. By Claim 3.17, we know that for every $\eta \in I'''$, $\widehat{\boldsymbol{v}}_{\boldsymbol{h},\boldsymbol{\eta}} = \widehat{\boldsymbol{v}}_{\boldsymbol{h},\boldsymbol{\omega}}$ with overwhelming probability. Therefore, in both protocol and simulation

$$\sum_{i\in[3]} \boldsymbol{v}_{\boldsymbol{h},\boldsymbol{i}} = (\overbrace{\underbrace{0,0,\ldots,0}_{N},\beta,0,0,\ldots,0}^{\omega}) - \widehat{\boldsymbol{v}}_{\boldsymbol{h},\boldsymbol{\omega}}.$$

If there exists only one honest party, the claim is proven. If there exist more than one honest party, we must show that the shares $\boldsymbol{v}_{\boldsymbol{h},\boldsymbol{i}}$ of the honest parties look random.

Consider $(k, j) \in \mathcal{T}$ with $j \neq \omega(k)$. Since $\epsilon_k^j = 0$, the seed generation is exactly as in the honest case, hence, by Claim 3.18, we know that $S^j_{k,1,1} = S^j_{k,2,1}$ and $\{S^j_{k,0,0}, S^j_{k,0,1}\} = \{S^j_{k,1,0}, S^j_{k,2,0}\}$. If P_{σ_1} and P_{σ_2} are both honest, by Claim 3.7, the adversary has no information about $S^j_{k,1,1}$ and $S^j_{k,2,1}$.

If P_{σ_0} and P_{σ_2} are both honest, $S^j_{k,2,0}$ is uniformly distributed in $\{0,1\}^\lambda$ from the adversary's perspective. Indeed, $S^j_{k,2,0}$ is independent of $S^j_{k,1,0}$ and the values $Z^j_k[0]$ and $Z^j_k[1]$ leak no information as $W^j_k[0]$ and $W^j_k[1]$ are independent, uniformly distributed in $\{0,1\}^\lambda$ and unknown to the adversary.

If P_{σ_0} and P_{σ_1} are both honest, $S^j_{k,1,0}$ is uniformly distributed in $\{0,1\}^\lambda$ from the adversary's perspective. Indeed, $S^j_{k,1,0}$ is independent of $S^j_{k,2,0}$. Moreover, the values $Z^j_k[0]$ and $Z^j_k[1]$ leak no information leak no information about it. Indeed, one of them is $W^j_k \oplus S^j_{k,2,0}$, the other one is $W^j_k[t^j_{k,2} \oplus 1] \oplus S^j_{k,1,0}$. The former is clearly independent of $S^j_{k,1,0}$, the latter is too as $W^j_k[t^j_{k,2} \oplus 1]$ is unknown to the adversary.

By Claim 3.9, we also know that for every $k \in \mathcal{K}$ and honest party P_{σ_i}, there exists $j \in [\mathrm{dE}(k)]$ and $b \in \{0,1\}$ such that $S^j_{k,i,b}$ is uniformly distributed in $\{0,1\}^\lambda$ from the perspective of every other party. This cannot happen when $j \neq \omega(k)$, indeed, in that case, there always exists another party knowing $S^j_{k,i,b}$. Hence, such j must be $\omega(k)$.

Let θ be the index of a honest party. Observe that for each $(k, j) \in \mathcal{T}$, every honest party P_{σ_i} with $\sigma_i \neq \theta$ holds a value $S^j_{k,i} \in \{S^j_{k,i,0}, S^j_{k,i,1}\}$ uniformly distributed from the perspective of everybody except P_{σ_i} and possibly P_θ.

Consider the base case $k = -1$ and observe that $S^{\omega(-1)}_{-1,\theta}$ protects the privacy of $(S^j_{-1,i})_{i\in\mathcal{H}\setminus\{\theta\}}$ with unconditional security when CW is computed. As a consequence, the elements

$$v^j_{0,i} = CW \cdot \left(u^0_i \cdot \mathrm{Enc}(S^j_{-1,i,0}) + u^1_i \cdot \mathrm{Enc}(S^j_{-1,i,1})\right)$$

are random and mutually independent for every $i \in \mathcal{H} \setminus \{\theta\}$ and $j \in [2]$.

Suppose that we have proven that $(\boldsymbol{v_{k,i}})_{i\in\mathcal{H}\setminus\{\theta\}}$ are random and independent from the adversary's perspective for a certain $k \in [h]$. Due to the randomness of $\boldsymbol{v_{k,i}}$, we know that $\boldsymbol{CW_{k,i}}$ is independent of $G_k(S^j_{k,i})$ for every $j \in [\mathrm{dE}(k)]$ and $i \in \mathcal{H} \setminus \{\theta\}$. As a consequence, since G_k is a PRG and $S^j_{k,i}$ is unknown to every party except P_{σ_i} and possibly P_θ, the vectors $\left\{\boldsymbol{v^j_{k+1,i}} | j \in [\mathrm{dE}(k)], i \in \mathcal{H}, i \neq \theta\right\}$ are random in $\mathbb{F}^{\mathrm{dE}(k)}$ and independent.

By induction, we conclude that the shares $(\boldsymbol{v_{h,i}})_{i\in\mathcal{H}\setminus\{\theta\}}$ are indistinguishable from $|\mathcal{H}| - 1$ random elements in $\mathbb{F}^N$. That ends the proof of the claim and of the theorem. ■

□

E Security Proof of Π_{Offline}

Proof (of Theorem 4).

Consider the simulator $\mathcal{S}_{\text{Offline}}$ described in Figure 17. We prove that no PPT adversary is able to distinguish between the protocol Π_{Offline} and the composition of $\mathcal{F}_{\text{Offline}}$ with $\mathcal{S}_{\text{Offline}}$.

Clearly, the simulation of the initialisation is unconditionally secure. Indeed, the only difference between protocol an simulation is that the values $(\alpha_i)_{i\in\mathcal{H}}$ are uniformly distributed in $\mathbb{F}^\times$ in the former and random in $\mathbb{F}$ in the latter. Since we assume $\mathbb{F}$ has exponential cardinality in the security parameter, the two distributions are indistinguishable. Let α be $\sum_{i\in[n]} \alpha_i$.

It is also immediate to see that the protocol never aborts if all the parties are honest. Observe that in that case, the non-zero elements of the procedures 2-DSPF and 3-DSPF are never equal to 0. Indeed, they are the product of elements in $\mathbb{F}^\times$.

Input Masks. Observe that the simulation of Input Masks is perfect until the outputs are revealed. We recall that P_j is the party to which the masks are addressed.

Claim 4.1. *In the protocol, we have the following relations*

$$\sum_{i\in[n]} x_i = x, \qquad \sum_{i\in[n]} m_{x,i} = \alpha \cdot x.$$

Proof of the claim. By how 2-DSPF is defined, we know that for every $i \neq j$ and $r \in [c]$, we have

$$\widetilde{e}^r_i(X) + \widetilde{e}^{r,1}_i(X) = \sum_{l\in[t]} \mu^r_i[l] \cdot X^{\omega^r[l]} = \sum_{l\in[t]} \alpha_i \cdot \beta^r[l] \cdot X^{\omega^r[l]} = \alpha_i \cdot e^r(X).$$

$\mathcal{S}_{\text{Offline}}$

Initialisation. The simulator waits for $(\alpha_i)_{i\in\mathcal{C}}$ from the adversary and forwards them to the functionality. Moreover, it samples $\alpha_i \xleftarrow{\$} \mathbb{F}^\times$ for every $i \in \mathcal{H}$.

Input Masks. Let P_j be the party to which the masks are addressed. $\mathcal{S}_{\text{Offline}}$ runs the protocol with the adversary simulating the honest parties:

1. If the simulation aborts, the simulator sends (Abort) to the functionality.
2. At the end, the simulator reconstructs the sum of the outputs of the corrupted parties

$$\widehat{x} \leftarrow \sum_{i\in\mathcal{C}} x_i, \qquad \widehat{m}_x \leftarrow \sum_{i\in\mathcal{C}} m_{x,i}.$$

   Observe that $\mathcal{S}_{\text{Offline}}$ can perform this operation. Indeed, in the initialisation, it learnt $(\alpha_i)_{i\in\mathcal{C}}$. Moreover, in every execution of 2-DSPF involving corrupted parties, the adversary sends to the simulator the shares that the corrupted parties selected for the output. Finally, for every honest party P_i and corrupted party P_k, the simulator knows both the seeds $s_{i,k}$ and $s_{k,i}$. Observe that if P_i and P_k are both corrupted, the seeds $s_{k,i}$ and $s_{i,k}$ do not affect the value of $\widehat{x}$ and $\widehat{m}_x$.
3. If P_j is corrupted, the simulator recomputes x and sends $(x, \widehat{x}, \widehat{m}_x)$ to the functionality. The operation can always be performed by $\mathcal{S}_{\text{Offline}}$. Indeed, at the beginning of the procedure, it received $\boldsymbol{\beta}^{\boldsymbol{r}}$ and $\boldsymbol{\omega}^{\boldsymbol{r}}$ for every $r \in [c]$.
4. If P_j is honest the simulator sends $(\widehat{x}, \widehat{m}_x)$ to the functionality.

Triple Generation. $\mathcal{S}_{\text{Offline}}$ runs the protocol with the adversary simulating the honest parties:

1. If the simulation aborts, the simulator sends (Abort) to the functionality.
2. If the adversary tries to guess any non-zero element in any execution of 3-DSPF when any of the three addressed party is honest, the simulator sends (Abort) to the functionality and notifies the abort to the adversary.
3. At the end, the simulator reconstructs the sum of the outputs of the corrupted parties

$$\begin{aligned} \widehat{x} &\leftarrow \sum_{i\in\mathcal{C}} x_i, & \widehat{y} &\leftarrow \sum_{i\in\mathcal{C}} y_i, & \widehat{z} &\leftarrow \sum_{i\in\mathcal{C}} z_i, \\ \widehat{m}_x &\leftarrow \sum_{i\in\mathcal{C}} m_{x,i}, & \widehat{m}_y &\leftarrow \sum_{i\in\mathcal{C}} m_{y,i}, & \widehat{m}_z &\leftarrow \sum_{i\in\mathcal{C}} m_{z,i}. \end{aligned}$$

   Then, it sends $(\widehat{x}, \widehat{m}_x, \widehat{y}, \widehat{m}_y, \widehat{z}, \widehat{m}_z)$ to the functionality. Observe that $\mathcal{S}_{\text{Offline}}$ can perform this operation. Indeed, in the initialisation, it learnt $(\alpha_i)_{i\in\mathcal{C}}$. Moreover, at the beginning of the procedure, it received $\boldsymbol{\beta}_{\boldsymbol{i}}^{\boldsymbol{r}}, \boldsymbol{\gamma}_{\boldsymbol{i}}^{\boldsymbol{r}}, \boldsymbol{\omega}_{\boldsymbol{i}}^{\boldsymbol{r}}, \boldsymbol{\eta}_{\boldsymbol{i}}^{\boldsymbol{r}}$ for every $i \in \mathcal{C}$ and $r \in [c]$. Finally, in every execution of 2-DSPF and 3-DSPF involving corrupted parties, the adversary sends to the simulator the sum of the shares that the corrupted parties selected for the output.

Fig. 17. The simulator $\mathcal{S}_{\text{Offline}}$

Hence, defining $a_{c-1} := 1$, we obtain that

$$\sum_{i\in[n]} m'_{x,i} = \sum_{i\in[n]} \langle \boldsymbol{a}, \widetilde{\boldsymbol{e}}_{\boldsymbol{i}} \rangle = \sum_{i\in[n]} \sum_{r\in[c]} a_r \cdot \widetilde{e}_i^r =$$
$$= \sum_{r\in[c]} a_r \cdot \Big(\sum_{i\neq j} \widetilde{e}_i^r(X) + \sum_{i\neq j} \widetilde{e}_i^{r,1}(X) + \alpha_j \cdot e^r(X) \Big) =$$
$$= \sum_{r\in[c]} a_r \cdot \Big(\sum_{i\in[n]} \alpha_i \cdot e^r(X) \Big) = \alpha \cdot \langle \boldsymbol{a}, \boldsymbol{e} \rangle = \alpha \cdot x.$$

Observe that

$$\sum_{i\in[n]} \sum_{k\neq i} \Big(G(s_{i,k}) - G(s_{k,i}) \Big) = \mathbf{0}.$$

We conclude that

$$\sum_{i\in[n]} x_i = x, \qquad \sum_{i\in[n]} m_{x,i} = \sum_{i\in[n]} m'_{x,i} = \alpha \cdot x.$$

■

Claim 4.2. *Consider the protocol and let P_ι be an honest party. By the PRG security of G, the values $(x_k, m_{x,k})_{k\in\mathcal{H}\setminus\{\iota\}}$ look independent and uniformly distributed in R.*

Proof of the claim. Observe that for every $k \in \mathcal{H}$ with $k \neq \iota$, the seed $s_{k,\iota}$ is random from the adversary perspective. By the PRG security, the adversary cannot realise that we substitute $G(s_{k,\iota})$ with two random elements in R. Observe that these terms are added to obtain x_k and $m_{x,k}$. So, the latter are now uniformly distributed over R. We conclude the proof of the claim by observing that $s_{k,\iota}$ is known only to P_k and P_ι, therefore, the values $(x_k, m_{x,k})_{k\in\mathcal{H}\setminus\{\iota\}}$ are all independent. ■

Observe that by the Claims 4.1 and 4.2, we can conclude that no PPT adversary can distinguish between the real Input Masks and the simulation when P_j is corrupted. The following claim shows that indistinguishability holds even when P_j is honest.

Claim 4.3. *If P_j is honest, by the R^c-LPN$_t$ assumption with static leakage, no PPT adversary can distinguish between the real Input Masks and the simulation.*

Proof of the claim. We can easily convert an adversary $\mathcal{A}$ distinguishing between protocol and simulation into an attacker $\mathcal{A}'$ that wins the module-LPN game with non-negligible advantage.

Upon activation $\mathcal{A}'$ would initialise an internal copy of $\mathcal{A}$ and run the protocol simulating the honest parties. During the generation of input masks addressed to an honest party P_j, $\mathcal{A}'$ would let its challenger select the non-zero values $\boldsymbol{\beta^r}$ and special positions $\boldsymbol{\omega^r}$ for every $r \in [c]$. When $\mathcal{A}$ tries to guess a special position $\omega^r[l]$ for some $r \in [c]$ and $l \in [t]$ by specifying a set $I_r^l \subseteq [N]$ during 2-DSPF, $\mathcal{A}'$

would issue a query (r, l, I_r^l) to its challenger and forward the reply to $\mathcal{A}$. After exchanging the seeds, $\mathcal{A}'$ would wait for $(\boldsymbol{a}, x)$ from its challenger. We recall that x is computed as in the protocol with probability $1/2$. In the other cases, it is uniformly sampled in R. $\mathcal{A}'$ would model $\mathcal{F}_{\text{Rand}}$ by sending $\boldsymbol{a}$ to $\mathcal{A}$. Finally, the attacker would reconstruct the sum of the shares of the corrupted parties

$$\widehat{x} \leftarrow \sum_{i \in \mathcal{C}} x_i, \qquad\qquad \widehat{m}_x \leftarrow \sum_{i \in \mathcal{C}} m_{x,i}$$

as $\mathcal{S}_{\text{Offline}}$ does. Then, for every $i \in \mathcal{H}$, $\mathcal{A}'$ would simulate the output of P_i by generating a random pair $(x_i, m_{x,i}) \in R^2$ subject to

$$\sum_{i \in \mathcal{H}} x_i + \widehat{x} = x, \qquad\qquad \sum_{i \in \mathcal{H}} m_{x,i} + \widehat{m}_x = x \cdot \alpha.$$

Observe that when the challenger replies with random elements in R, the view of $\mathcal{A}$ is indistinguishable from the view in the simulation. Indeed, the only difference is that the challenger samples $\boldsymbol{\beta^r}$ uniformly in $\mathbb{F}^t$, whereas the simulator samples it in $\mathbb{F}^{\times t}$. The statistical distance between the two distributions in however negligible, so they are indistinguishable.

If instead the challenger computes x using sparse polynomials in R, the view of $\mathcal{A}$ is indistinguishable from the view in the protocol. The differences are the distribution of $(x_i, m_{x,i})_{i \in \mathcal{H}}$ and the fact that the challenger samples $\boldsymbol{\beta^r}$ uniformly in $\mathbb{F}^t$, whereas P_j samples it in $\mathbb{F}^{\times t}$. As we argued before, the second discrepancy is not an issue as the distributions are indistinguishable. The first one is not a problem either, due to Claims 4.1 and 4.2.

So, if $\mathcal{A}$ distinguished between protocol and simulation with non-negligible advantage, then $\mathcal{A}'$ would break the R^c-LPN$_t$ hardness. ■

Triple Generation. Observe that the simulation is unconditionally secure until the outputs are revealed. As a matter of fact, the only difference between protocol and simulation until that point, is that in the former, guesses of the non-zero values in 3-DSPF may succeed, whereas in the latter, they always fail. Anyway, observe that the protocol leaks no information about $\boldsymbol{\beta_i^r}$ and $\boldsymbol{\gamma_i^r}$ for every $i \in \mathcal{H}$ and $r \in [c]$. Moreover, the non-zero elements cannot be 0, otherwise, it means that for some $j \in \mathcal{C}$, the adversary chose $\alpha_j = 0$, or $\beta_j^r[l] = 0$ or $\gamma_j^r[l] = 0$ for some $r \in [c]$ and $l \in [t]$. In those cases, the protocol would have aborted during an execution of 2-DSPF. We understand that if at least one of the addressed parties is honest, the non-zero elements are uniformly distributed in $\mathbb{F}^\times$ from the adversary perspective. Hence, any guess fails with overwhelming probability.

Before proceeding with our analysis, we define the random variables $x := \sum_{i \in [n]} x_i$ and $y := \sum_{i \in [n]} y_i$.

Claim 4.4. *In the protocol, we have the following relations*

$$\sum_{i \in [n]} m_{x,i} = \alpha \cdot x, \qquad\qquad \sum_{i \in [n]} m_{y,i} = \alpha \cdot y.$$

Proof of the claim. By how 2-DSPF is defined, we know that for every $i \neq j$ and $r \in [c]$, we have

$$\widetilde{u}_{i,j}^{r,0}(X) + \widetilde{u}_{i,j}^{r,1}(X) = \sum_{l\in[t]} \mu_{i,j}^r[l] \cdot X^{\omega_j^r[l]} = \sum_{l\in[t]} \alpha_i \cdot \beta_j^r[l] \cdot X^{\omega_j^r[l]} = \alpha_i \cdot u_j^r(X),$$
$$\widetilde{v}_{i,j}^{r,0}(X) + \widetilde{v}_{i,j}^{r,1}(X) = \sum_{l\in[t]} \nu_{i,j}^r[l] \cdot X^{\eta_j^r[l]} = \sum_{l\in[t]} \alpha_i \cdot \gamma_j^r[l] \cdot X^{\eta_j^r[l]} = \alpha_i \cdot v_j^r(X).$$

Hence, defining $a_{c-1} := 1$, we obtain that

$$\begin{aligned}
\sum_{i\in[n]} m_{x,i} &= \sum_{i\in[n]} \langle \boldsymbol{a}, \widetilde{\boldsymbol{u}}_{\boldsymbol{i}} \rangle = \sum_{i\in[n]} \sum_{r\in[c]} a_r \cdot \widetilde{u}_i^r(X) = \\
&= \sum_{r\in[c]} a_r \cdot \Bigg(\sum_{i\in[n]} \alpha_i \cdot u_i^r(X) + \sum_{i\neq j} \Big(\widetilde{u}_{i,j}^{r,0}(X) + \widetilde{u}_{j,i}^{r,1}(X) \Big) \Bigg) = \\
&= \sum_{r\in[c]} a_r \cdot \Bigg(\sum_{i\in[n]} \alpha_i \cdot u_i^r(X) + \sum_{i\neq j} \alpha_i \cdot u_j^r(X) \Bigg) = \\
&= \sum_{r\in[c]} a_r \cdot \Big(\alpha \cdot \sum_{i\in[n]} u_i^r(X) \Big) = \alpha \cdot \sum_{i\in[n]} \langle \boldsymbol{a}, \boldsymbol{u}_{\boldsymbol{i}} \rangle = \alpha \cdot x.
\end{aligned}$$

In a totally analogous way, we can prove the second half of the claim. ■

Claim 4.5. *In the protocol, we have that* $\sum_{i\in[n]} z_i = x \cdot y$.

Proof of the claim. By how 2-DSPF is defined, we know that for every $i \neq j$ and $r, s \in [c]$, we have

$$\begin{aligned}
w_{i,j}^{r,s,0}(X) + w_{i,j}^{r,s,1}(X) &= \sum_{l,h\in[t]} \rho_{i,j}^{r,s}[lt+h] \cdot X^{\zeta_{i,j}^{r,s}[lt+h]} = \\
&= \sum_{l,h\in[t]} \beta_i^r[l] \cdot \gamma_j^s[h] \cdot X^{\omega_i^r[l] + \eta_j^s[h]} = \\
&= \Big(\sum_{l\in[t]} \beta_i^r[l] \cdot X^{\omega_i^r[l]} \Big) \cdot \Big(\sum_{h\in[t]} \gamma_j^s[h] \cdot X^{\eta_j^s[h]} \Big) = \\
&= u_i^r(X) \cdot v_j^s(X).
\end{aligned}$$

Hence, defining $a_{c-1} := 1$, we obtain that

$$\begin{aligned}
\sum_{i\in[n]} z_i &= \sum_{i\in[n]} \langle \boldsymbol{a}\otimes\boldsymbol{a}, \boldsymbol{w_i}\rangle = \sum_{i\in[n]}\sum_{r,s\in[c]} a_r\cdot a_s\cdot w_i^{rc+s}(X) = \\
&= \sum_{r,s\in[c]} a_r\cdot a_s\cdot\left(\sum_{i\in[n]} u_i^r(X)\cdot v_i^s(X) + \sum_{i\neq j}\left(w_{i,j}^{r,s,0}(X) + w_{j,i}^{r,s,1}(X)\right)\right) = \\
&= \sum_{r,s\in[c]} a_r\cdot a_s\cdot\left(\sum_{i\in[n]} u_i^r(X)\cdot v_i^s(X) + \sum_{i\neq j} u_i^r(X)\cdot v_j^s(X)\right) = \\
&= \sum_{i,j\in[n]}\sum_{r,s\in[c]} a_r\cdot a_s\cdot u_i^r(X)\cdot v_j^s(X) = \\
&= \sum_{i,j\in[n]}\left(\sum_{r\in[c]} a_r\cdot u_i^r\right)\cdot\left(\sum_{s\in[c]} a_s\cdot v_j^s\right) = \\
&= \sum_{i,j\in[n]} \langle \boldsymbol{a}, \boldsymbol{u_i}\rangle\cdot\langle \boldsymbol{a}, \boldsymbol{v_j}\rangle = \sum_{i,j\in[n]} x_i\cdot y_j = x\cdot y.
\end{aligned}$$

■

Claim 4.6. *In the protocol, we have that* $\sum_{i\in[n]} m_{z,i} = \alpha\cdot x\cdot y$.

Proof of the claim. By how 3-DSPF is defined, we know that for every not-all-equal $i,j,k\in[n]$ and every $r,s\in[c]$, we have

$$\begin{aligned}
\widetilde{w}_{i,j,k}^{r,s,0} + \widetilde{w}_{i,j,k}^{r,s,1} + \widetilde{w}_{i,j,k}^{r,s,2} &= \sum_{l,h\in[t]} \tau_{i,j,k}^{r,s}[lt+h]\cdot X^{\zeta_{j,k}^{r,s}[lt+h]} = \\
&= \sum_{l,h\in[t]} \alpha_i\cdot\beta_j^r[l]\cdot\gamma_k^s[h]\cdot X^{\omega_j^r[l]+\eta_k^s[h]} = \\
&= \alpha_i\cdot\left(\sum_{l\in[t]}\beta_j^r[l]\cdot X^{\omega_j^r[l]}\right)\cdot\left(\sum_{h\in[t]}\gamma_k^s[h]\cdot X^{\eta_k^s[h]}\right) = \\
&= \alpha_i\cdot u_j^r(X)\cdot v_k^s(X).
\end{aligned}$$

Hence, defining $a_{c-1} := 1$, we obtain that

$$\sum_{i\in[n]} m_{z,i} = \sum_{i\in[n]} \langle \boldsymbol{a}\otimes\boldsymbol{a}, \widetilde{\boldsymbol{w}}_{\boldsymbol{i}}\rangle = \sum_{i\in[n]}\sum_{r,s\in[c]} a_r\cdot a_s\cdot \widetilde{w}_i^{rc+s}(X) =$$

$$= \sum_{r,s\in[c]} a_r\cdot a_s\cdot\left(\sum_{i\in[n]}\alpha_i\cdot u_i^r(X)\cdot v_i^s(X) + \sum_{(j,k)\neq(i,i)}\left(\widetilde{w}_{i,j,k}^{r,s,0} + \widetilde{w}_{k,i,j}^{r,s,1} + \widetilde{w}_{j,k,i}^{r,s,2}\right)\right) =$$

$$= \sum_{r,s\in[c]} a_r\cdot a_s\cdot\left(\sum_{i\in[n]}\alpha_i\cdot u_i^r(X)\cdot v_i^s(X) + \sum_{(j,k)\neq(i,i)}\alpha_i\cdot u_j^r(X)\cdot v_k^s(X)\right) =$$

$$= \sum_{i,j,k\in[n]}\sum_{r,s\in[c]} a_r\cdot a_s\cdot\alpha_i\cdot u_j^r(X)\cdot v_k^s(X) =$$

$$= \alpha\cdot\sum_{j,k\in[n]}\left(\sum_{r\in[c]} a_r\cdot u_j^r\right)\cdot\left(\sum_{s\in[c]} a_s\cdot v_k^s\right) =$$

$$= \alpha\cdot\sum_{j,k\in[n]}\langle\boldsymbol{a},\boldsymbol{u_j}\rangle\cdot\langle\boldsymbol{a},\boldsymbol{v_k}\rangle =$$

$$= \alpha\cdot\sum_{j,k\in[n]} x_j\cdot y_k = \alpha\cdot x\cdot y.$$

■

Claim 4.7. *Consider the protocol and let P_ι be an honest party. The values $(m_{x,j}, m_{y,j}, z_j, m_{z,j})_{j\in\mathcal{H}\setminus\{\iota\}}$ are uniformly distributed in R and independent of the remaining outputs and the view of the adversary.*

Proof of the claim. Observe that for every $j\in\mathcal{H}$ with $j\neq\iota$, the polynomials $\widetilde{u}_{j,\iota}^{c-1,0}(X)$, $\widetilde{v}_{j,\iota}^{c-1,0}(X)$, $w_{j,\iota}^{c-1,c-1,0}(X)$ and $\widetilde{w}_{j,\iota,\iota}^{c-1,c-1,0}(X)$ are all random in R and independent of the view of the adversary and of the honest parties $(P_i)_{i\in\mathcal{H}\setminus\{j,\iota\}}$. Here, we are implicitly relying on the fact that the polynomials in $\mathbb{F}[X]$ of degree less than $2N$ form a vector space and the reduction modulo $F(X)$ is a $\mathbb{F}$-linear operation from this space to R. As a consequence, every element in R has the same number of preimages, therefore, the reduction modulo $F(X)$ maps the uniform distribution over the polynomials of degree less than $2N$ into the uniform distribution over R.

Since $\widetilde{u}_{j,\iota}^{c-1,0}(X)$, $\widetilde{v}_{j,\iota}^{c-1,0}(X)$, $w_{j,\iota}^{c-1,c-1,0}(X)$ and $\widetilde{w}_{j,\iota,\iota}^{c-1,c-1,0}(X)$ are terms of $\widetilde{u}_j^{c-1}(X), \widetilde{v}_j^{c-1}(X), w_j^{c^2-1}(X)$ and $\widetilde{w}_j^{c^2-1}(X)$ respectively, we understand that the latter are all random in R and independent of the view of the adversary and of the honest parties $(P_i)_{i\in\mathcal{H}\setminus\{j,\iota\}}$.

To conclude the proof of the claim, observe that $\widetilde{u}_j^{c-1}(X)$ and $\widetilde{v}_j^{c-1}(X)$ are multiplied by $a_{c-1}=1$ when computing $m_{x,j} = \langle\boldsymbol{a},\widetilde{\boldsymbol{u}}_{\boldsymbol{j}}\rangle$ and $m_{y,j} = \langle\boldsymbol{a},\widetilde{\boldsymbol{v}}_{\boldsymbol{j}}\rangle$ respectively. Moreover, $w_j^{c^2-1}(X)$ and $\widetilde{w}_j^{c^2-1}(X)$ are multiplied by $a_{c-1}\cdot a_{c-1}=1$ when computing $z_j = \langle\boldsymbol{a}\otimes\boldsymbol{a},\boldsymbol{w}_{\boldsymbol{j}}\rangle$ and $m_{z,j} = \langle\boldsymbol{a}\otimes\boldsymbol{a},\widetilde{\boldsymbol{w}}_{\boldsymbol{j}}\rangle$. ■

Claim 4.8. *By the R^c-LPN$_t$ assumption with static leakage, no PPT adversary can distinguish between the real* Triple Generation *and the simulation.*

Proof of the claim. Observe that when the second and the third parties P_j and P_k of 3-DSPF are both corrupted, the special positions are already known to the adversary. Indeed, they are $\boldsymbol{\omega}_{\boldsymbol{j}}^{\boldsymbol{r}} \boxplus \boldsymbol{\eta}_{\boldsymbol{k}}^{\boldsymbol{s}}$ for some $r, s \in [c]$.

We proceed by a series of $2|\mathcal{H}|$ hybrids. In the initial stage, we consider the protocol execution in which, for every $i \in \mathcal{H}$, the values $m_{x,i}, m_{y,i}, z_i$ and $m_{z,i}$ are substituted with random elements in R, subject to

$$\sum_{i\in\mathcal{H}} m_{x,i} + \widehat{m}_x = \alpha \cdot x, \qquad \sum_{i\in\mathcal{H}} m_{y,i} + \widehat{m}_y = \alpha \cdot y,$$
$$\sum_{i\in\mathcal{H}} z_i + \widehat{z} = x \cdot y, \qquad \sum_{i\in\mathcal{H}} m_{z,i} + \widehat{m}_z = \alpha \cdot x \cdot y.$$

The variables $\widehat{m}_x$, $\widehat{m}_y$, $\widehat{z}$ and $\widehat{m}_z$ represent the sum of the shares of the corrupted parties and they are formally defined and derived as in $\mathcal{S}_{\text{Offline}}$. In the initial stage, we also make the protocol abort whenever the adversary tries to guess any non-zero element in an execution of 3-DSPF. Observe that by Claims 4.4, 4.5, 4.6 and 4.7, the initial stage is indistinguishable from Π_{Offline}.

Consider now an integer $i \leq |\mathcal{H}|$ and let j be the index of the i-th honest party. In the $2i$-th hybrid, we will substitute the final output x_j with a random element in R, in the $(2i+1)$-th hybrid we will do the same thing with y_j. Observe that in the last stage, the execution is identical to the simulation.

We show that any PPT adversary $\mathcal{A}$ distinguishing between two consequent hybrids can be converted into an efficient attacker $\mathcal{A}'$ against the R^c-LPN$_t$ hardness.

Suppose that $\mathcal{A}$ distinguishes between the $(2i-1)$-th and the $2i$-th hybrid. Let P_j be the i-th honest party. Observe that the only difference between the two stages is that x_j is computed as in the protocol in the former and randomly sampled in the latter. We construct the module-LPN attacker $\mathcal{A}'$ as follows. Upon activation, $\mathcal{A}'$ initialises an internal copy of $\mathcal{A}$ and runs the protocol simulating the honest parties. During the generation of multiplication triples, $\mathcal{A}'$ lets its challenger select the non-zero values $\boldsymbol{\beta}_{\boldsymbol{j}}^{\boldsymbol{r}}$ and special positions $\boldsymbol{\omega}_{\boldsymbol{j}}^{\boldsymbol{r}}$ for every $r \in [c]$. When $\mathcal{A}$ tries to guess a special position $\omega_j^r[l]$ for some $r \in [c]$ and $l \in [t]$ by specifying a set $I \subseteq [N]$ during 2-DSPF, $\mathcal{A}'$ issues a query (r, l, I) to its challenger and forwards the reply to $\mathcal{A}$. Moreover, when $\mathcal{A}$ tries to guess a special position $\zeta_{j,k}^{r,s}[lt+h] = \omega_j^r[l] + \eta_k^s[h]$ for some $k \in [n]$, $r, s \in [c]$ and $l, h \in [t]$ by specifying a set $I' \subseteq [N]$ during 2-DSPF or 3-DSPF, $\mathcal{A}'$ computes the set

$$I'' \leftarrow \left\{\chi - \eta_k^s[h] \mid \chi \in I'\right\},$$

issues the query (r, l, I'') to its challenger and forwards the reply to $\mathcal{A}$. Observe that $\mathcal{A}'$ knows $\eta_k^s[h]$ so I'' can always be computed. When $\mathcal{A}$ tries to guess any non-zero position in 3-DSPF, $\mathcal{A}'$ always simulates an abort of the protocol.

Finally, $\mathcal{A}'$ waits for $(\boldsymbol{a}, x_j)$ from its challenger. We recall that x_j is computed as in the protocol with probability $1/2$. In the other cases, it is uniformly sampled in R. $\mathcal{A}'$ models $\mathcal{F}_{\text{Rand}}$ by sending $\boldsymbol{a}$ to $\mathcal{A}$. At the end, the attacker $\mathcal{A}'$ computes $\widehat{m}_x$, $\widehat{m}_y$, $\widehat{z}$, $\widehat{m}_z$ and the outputs of the honest parties. For every $k \in \mathcal{H}$ with

$k < j$, it substitutes x_k and y_k with random elements in R. Finally, it generates $m_{x,i}, m_{y,i}, z_i$ and $m_{z,i}$ for every $i \in \mathcal{H}$ as in the simulation.

Observe that when the challenger replies with random elements in R, the view of $\mathcal{A}$ is indistinguishable from the view in the $2i$-th hybrid. Indeed, the only difference is that the challenger samples $\boldsymbol{\beta}_{\boldsymbol{j}}^{\boldsymbol{r}}$ uniformly in $\mathbb{F}^t$, whereas the P_j samples it in $\mathbb{F}^{\times t}$. The statistical distance between the two distributions in however negligible, so they are indistinguishable.

If instead the challenger computes x_j using sparse polynomials in R, the view of $\mathcal{A}$ is indistinguishable from the view in the $(2i-1)$-th hybrid. Again, the difference is the fact that the challenger samples $\boldsymbol{\beta}_{\boldsymbol{j}}^{\boldsymbol{r}}$ uniformly in $\mathbb{F}^t$, whereas P_j samples it in $\mathbb{F}^{\times t}$. As we argued before, this is not an issue as the distributions are indistinguishable.

So, if $\mathcal{A}$ distinguished between the $(2i-1)$-th and the $2i$-th hybrid with non-negligible advantage, then $\mathcal{A}'$ would break the R^c-LPN$_t$ hardness. In a totally analogous way, we can prove that the same holds if $\mathcal{A}$ distinguished between the $2i$-th and the $(2i+1)$-th hybrid. ■

□

F Implementation of $\mathcal{F}_{\text{auth-ROT}}$

We now show how the functionality $\mathcal{F}_{\text{auth-ROT}}$, described in Figure 18, can be implemented with low communication. Our protocol, $\Pi_{\text{auth-ROT}}$, will rely on a Correlated-OT functionality $\mathcal{F}_{\text{COT}}$, whose description is available in Figure 19. Note that the latter can be implemented with logarithmic communication [BCG+19a]. We will also use $\mathcal{F}_{\text{Rand}}$.

A formal description of $\Pi_{\text{auth-ROT}}$ can be found in Figure 20. Observe that this is a 2-party protocol between a sender S and a receiver R. When we implement $\mathcal{F}_{\text{auth-ROT}}^{i,j}$, the sender will be P_i whereas the receiver will be P_j. The ideas used in the protocol are rather simple. Essentially, the correlated-OT tuples are converted into Random-OT tuples by means of tweakable correlation-robust hash function H. However, their role is not accomplished, yet. Indeed, they are also used as BeDOZa-style MACs [BDOZ11] to authenticate the choice bits of the receiver and allow verifiable computations.

Theorem 5. *Suppose that $H : \{0,1\}^\lambda \times \{0,1\}^* \longrightarrow \{0,1\}^\lambda$ is a correlation robust tweakable hash function. Then, $\Pi_{auth\text{-}ROT}$ securely implements $\mathcal{F}_{auth\text{-}ROT}$ in the $(\mathcal{F}_{COT}, \mathcal{F}_{Rand})$-hybrid model. Moreover, if both the sender and the receiver are honest, the protocol does not abort.*

Proof. Consider the simulator in Figure 21. We show that no adversary can distinguish between protocol and simulation.

Clearly, when all the parties are corrupted, the simulator can just run the protocol with the adversary and the simulation is perfect. Therefore, we can always assume there exists at least one honest party.

Observe that the simulation of Random is perfect. So we focus our attention on ROT and Linear Combination.

$\mathcal{F}_{\text{auth-ROT}}$

ROT. On input (ROT, a) from both the sender S and the receiver R where a is a fresh element in $\{0,1\}^*$, $\mathcal{F}_{\text{auth-ROT}}$ proceeds as follows:

- If both R and S are honest, it samples a random bit $t \xleftarrow{\$} \{0,1\}$ and two strings $X_0, X_1 \xleftarrow{\$} \{0,1\}^\lambda$. Then, it sends (t, X_t) to R and (X_0, X_1) to S. Finally, it stores t with handle $[[t]]_2 := a$.
- If the receiver R is corrupted and S is honest, it waits for $t \in \{0,1\}$ and $X_t \in \{0,1\}^\lambda$ from the adversary. Then, it samples $X_{1-t} \xleftarrow{\$} \{0,1\}^\lambda$, it stores t with handle $[[t]]_2 := a$ and outputs (X_0, X_1) to S.
- If the sender S is corrupted and R is honest, it waits for $X_0, X_1 \in \{0,1\}^\lambda$ from the adversary. Then, it samples $t \xleftarrow{\$} \{0,1\}$, it stores it with handle $[[t]]_2 := a$ and outputs (t, X_t) to R.

Random. On input (Random, a) from every party where a is a fresh element in $\{0,1\}^*$, $\mathcal{F}_{\text{auth-ROT}}$ proceeds as follows:

- If the receiver R is honest, it outputs $t \xleftarrow{\$} \{0,1\}$ to R and stores the value with handle $[[t]]_2 := a$.
- If R is corrupted, it waits for $t \in \{0,1\}$ and stores it with identity $[[t]]_2 := a$.

Linear Combination. On input $(\mathsf{LinComb}, l, m, V, [[t_0]]_2, [[t_1]]_2, \ldots, [[t_{l-1}]]_2)$ from every party, the functionality checks that $l, m \in \mathbb{N}$ and $V \in \mathbb{F}_2^{m \times l}$. If this is the case, it retrieves the bit t_k with handle $[[t_k]]_2$ for every $k \in [l]$ and sets $t \leftarrow (t_0, t_1, \ldots, t_{l-1})$. Finally, the functionality sends $r \leftarrow V \cdot t$ to S.
Abort. On input (Abort) from the adversary, the functionality aborts.

Fig. 18. The functionality $\mathcal{F}_{\text{auth-ROT}}$

$\mathcal{F}_{\text{COT}}$

Initialise. On input (Init, Δ) from the sender S where $\Delta \in \{0,1\}^\lambda$ and (Init) from the receiver R, store Δ. Ignore subsequent calls to this procedure.
COT. On input (COT) from both R and S proceed as follows

- If both R and S are honest, sample $M \xleftarrow{\$} \{0,1\}^\lambda$ and $t \xleftarrow{\$} \mathbb{F}_2$. Then set $K \leftarrow M \oplus t \cdot \Delta$. Finally, output (M, t) to R and K to S.
- If S is corrupted, wait for $K \in \{0,1\}^\lambda$ from the adversary and sample $t \xleftarrow{\$} \mathbb{F}_2$. Then set $M \leftarrow K \oplus t \cdot \Delta$. Finally, output (M, t) to R.
- If R is corrupted, wait for $t \in \mathbb{F}_2$ and $M \in \{0,1\}^\lambda$ from the adversary. Then set $K \leftarrow M \oplus t \cdot \Delta$. Finally, output K to S.

Abort. On input (Abort) from the adversary, the functionality aborts.

Fig. 19. The functionality $\mathcal{F}_{\text{COT}}$

$\Pi_{\text{auth-ROT}}$

Initialise. The receiver R sends (Init) to $\mathcal{F}_{\text{COT}}$, the sender S samples $\Delta \xleftarrow{\$} \{0,1\}^\lambda$ and sends (Init, Δ) to $\mathcal{F}_{\text{COT}}$.

ROT. On input (ROT, a) where $a \in \{0,1\}^*$ is fresh, the parties perform the following operations

1. R and S call COT in $\mathcal{F}_{\text{COT}}$. Let $(M, t) \in \{0,1\}^\lambda \times \mathbb{F}_2$ the element obtained by R and let $K \in \{0,1\}^\lambda$ be the element obtained by S.
2. R outputs (t, X) where $X \leftarrow H\big(M, a\big)$ and stores (M, t), S outputs (X_0, X_1) where $X_0 \leftarrow H\big(K, a\big)$, $X_1 \leftarrow H\big(K \oplus \Delta, a\big)$ and stores K.

Random. On input (Random, a) where $a \in \{0,1\}^*$ is fresh, the parties perform the following operations

1. S and R call COT in $\mathcal{F}_{\text{COT}}$. Let $(M, t) \in \{0,1\}^\lambda \times \mathbb{F}_2$ the element obtained by R and let $K \in \{0,1\}^\lambda$ be the element obtained by S.
2. R outputs t and stores (M, t), S stores K.

Linear Combination. On input $(\mathsf{LinComb}, l, m, V, a_0, a_1, \ldots, a_{l-1})$ where $l, m \in \mathbb{N}$, $V \in \mathbb{F}_2^{m \times l}$ and a_i is a label of an execution of ROT or Random for every $i \in [l]$, the parties perform the following operations.

1. For every $i \in [l]$, R retrieves the pair $(M_i, t_i) \in \{0,1\}^\lambda \times \mathbb{F}_2$ obtained in the execution of ROT or Random of label a_i. Then, it sets $\boldsymbol{t} \leftarrow (t_0, t_1, \ldots, t_{l-1})$ and $\boldsymbol{M} \leftarrow (M_0, M_1, \ldots, M_{l-1})$.
2. R sends $\boldsymbol{r} \leftarrow V \cdot \boldsymbol{t}$ to S
3. S and R call $\mathcal{F}_{\text{Rand}}$ to obtain $\boldsymbol{\chi} \in \mathbb{F}_{2^\lambda}^m$.
4. R sends $T \leftarrow \langle \boldsymbol{\chi}, V \cdot \boldsymbol{M} \rangle$ to S.
5. For ever $i \in [l]$, let K_i be the value stored by S in the execution of ROT or Random of label a_i. S sets $\boldsymbol{K} \leftarrow (K_0, K_1, \ldots, K_{l-1})$ and

$$T' \leftarrow \langle \boldsymbol{\chi}, V \cdot \boldsymbol{K} \oplus \boldsymbol{r} \cdot \Delta \rangle.$$

   If $T = T'$, S outputs $\boldsymbol{r}$, otherwise, it makes the protocol abort.

Fig. 20. The protocol $\Pi_{\text{auth-ROT}}$

$\mathcal{S}_{\text{auth-ROT}}$

Initialise. If S is corrupted, upon receiving (Init, Δ) from the adversary, the simulator stores Δ.

ROT. The simulator performs the following operations

- If R is corrupted and S is honest, the simulator waits for $(M, t) \in \{0,1\}^\lambda \times \mathbb{F}_2$ from the adversary. Then, it sets $X \leftarrow H\big(M, a\big)$ and sends (X, t) to the functionality.
- If S is corrupted and R is honest, the simulator waits for $K \in \{0,1\}^\lambda$ from the adversary. Then, it sets $X_0 \leftarrow H\big(K, a\big)$ and $X_1 \leftarrow H\big(K \oplus \Delta, a\big)$. Then, it sends (X_0, X_1) to the functionality.

Random. The simulator performs the following operations

- If R is corrupted, it waits for $(M, t) \in \{0,1\}^\lambda \times \mathbb{F}_2$ from the adversary. The simulator sends t to the functionality.
- If S is corrupted, the simulator waits for $K \in \{0,1\}^\lambda$ from the adversary.

Linear Combination. On input $(\mathsf{LinComb}, l, m, V, a_0, a_1, \ldots, a_{l-1})$ where $l, m \in \mathbb{N}$, $V \in \mathbb{F}_2^{m \times l}$ and a_i is a label of an execution of ROT or Random for every $i \in [l]$, the simulator performs the following operations

- If S is the only corrupted party:
  1. the simulator waits for $\boldsymbol{r} \in \mathbb{F}_2^m$ from the functionality, then, it sends it to the adversary on behalf of R.
  2. The simulator sends $\boldsymbol{\chi} \xleftarrow{\$} \mathbb{F}_{2^\lambda}^m$ to the adversary.
  3. For every $i \in [l]$, let K_i be the element in $\{0,1\}^\lambda$ obtained by the simulator in the execution of ROT or Random of label a_i. The simulator sets $\boldsymbol{K} \leftarrow (K_0, K_1, \ldots, K_{l-1})$ and $T \leftarrow \langle \boldsymbol{\chi}, V \cdot \boldsymbol{K} \oplus \boldsymbol{r} \cdot \Delta \rangle$.
  4. The simulator sends $\boldsymbol{T}$ to the adversary. If the latter replies with ABORT, the simulator sends Abort to the functionality.
- If R is the only corrupted party:
  1. The simulator waits for $\boldsymbol{r} \in \mathbb{F}_2^m$ from the adversary.
  2. The simulator sends $\boldsymbol{\chi} \xleftarrow{\$} \mathbb{F}_{2^\lambda}^m$ to the adversary and waits for $T \in \{0,1\}^\lambda$ as a reply.
  3. For ever $i \in [l]$, let (M_i, t_i) be the COT tuple used by the simulator in the execution of ROT or Random of label a_i. Let $\boldsymbol{t} := (t_0, t_1, \ldots, t_{l-1})$ and $\boldsymbol{M} := (M_0, M_1, \ldots, M_{l-1})$. The simulator sends Abort to the functionality if and only if $V \cdot \boldsymbol{t} \neq \boldsymbol{r}$ or if $\langle \boldsymbol{\chi}, V \cdot \boldsymbol{M} \rangle \neq T$. Otherwise, the simulator forwards $(\mathsf{LinComb}, l, m, V, a_0, a_1, \ldots, a_{l-1})$ to the functionality.

Fig. 21. The simulator $\mathcal{S}_{\text{auth-ROT}}$

Claim 5.1. *If S is corrupted and R is honest, the protocol is perfectly simulated.*

Proof of the claim. Consider ROT and observe that the distribution of the value t output by the receiver is the same as in the protocol. Actually, the same happens for X. Indeed, in the protocol we have

$$X = H(M, a) = H(K \oplus t \cdot \Delta, a) = \begin{cases} X_0 & \text{if } t = 0, \\ X_1 & \text{if } t = 1. \end{cases}$$

Consider now $\mathsf{Linear\ Combination}$. Observe that $\boldsymbol{r}$ and $\boldsymbol{\chi}$ are perfectly simulated. Actually, the same holds for T. Indeed, in the protocol, we have

$$T = \langle \boldsymbol{\chi}, V \cdot \boldsymbol{M} \rangle = \langle \boldsymbol{\chi}, V \cdot (\boldsymbol{K} \oplus \boldsymbol{t} \cdot \Delta) \rangle = \langle \boldsymbol{\chi}, V \cdot \boldsymbol{K} \oplus V \cdot \boldsymbol{t} \cdot \Delta \rangle = \langle \boldsymbol{\chi}, V \cdot \boldsymbol{K} \oplus \boldsymbol{r} \cdot \Delta \rangle.$$

The right-hand side is the value sent by the simulator to the adversary. ■

Claim 5.2. *Consider the protocol and suppose that S is honest. Consider the procedure $\mathsf{Linear\ Combination}$ and define $\boldsymbol{\epsilon} := \boldsymbol{r} \oplus V \cdot \boldsymbol{t}$. The procedure $\mathsf{Linear\ Combination}$ succeeds if and only if $\langle \boldsymbol{\chi}, \boldsymbol{\epsilon} \rangle = 0$ and $T = \langle \boldsymbol{\chi}, V \cdot \boldsymbol{M} \rangle$, or*

$$\Delta = \frac{T - \langle \boldsymbol{\chi}, V \cdot \boldsymbol{M} \rangle}{\langle \boldsymbol{\chi}, \boldsymbol{\epsilon} \rangle}$$

Proof of the claim. Consider the procedure $\mathsf{Linear\ Combination}$ in the protocol and define $\boldsymbol{\epsilon} := \boldsymbol{r} \oplus V \cdot \boldsymbol{t}$. The value T' computed by the honest sender is

$$\begin{aligned} T' &= \langle \boldsymbol{\chi}, V \cdot \boldsymbol{K} \oplus \boldsymbol{r} \cdot \Delta \rangle = \langle \boldsymbol{\chi}, V \cdot \boldsymbol{K} \oplus V \cdot \boldsymbol{t} \cdot \Delta \oplus \boldsymbol{\epsilon} \cdot \Delta \rangle = \\ &= \langle \boldsymbol{\chi}, V \cdot (\boldsymbol{K} \oplus \boldsymbol{t} \cdot \Delta) \rangle \oplus \langle \boldsymbol{\chi}, \boldsymbol{\epsilon} \cdot \Delta \rangle = \langle \boldsymbol{\chi}, V \cdot \boldsymbol{M} \rangle \oplus \Delta \cdot \langle \boldsymbol{\chi}, \boldsymbol{\epsilon} \rangle. \end{aligned}$$

The procedure succeeds if and only if $T = T'$, therefore, if and only if $\langle \boldsymbol{\chi}, \boldsymbol{\epsilon} \rangle = 0$ and $T = \langle \boldsymbol{\chi}, V \cdot \boldsymbol{M} \rangle$, or

$$\Delta = \frac{T \oplus \langle \boldsymbol{\chi}, V \cdot \boldsymbol{M} \rangle}{\langle \boldsymbol{\chi}, \boldsymbol{\epsilon} \rangle}$$

■

Claim 5.3. *If S is honest and R is corrupted, by the correlation robustness of H, no PPT adversary can distinguish between protocol and simulation.*

Proof of the claim. Observe that we cannot directly reduce the security of the protocol to the correlation robustness of H. Indeed, the value Δ is used in $\mathsf{Linear\ Combination}$ too and both the procedures have some leakage. We proceed by means of two hybrids.

Hybrid 1. We start by considering a slightly modified version of the protocol, in which, during the initialisation, the sender samples $\widehat{M} \xleftarrow{\$} \{0,1\}^\lambda$. During

Linear Combination, the sender always makes the check succeed if $\langle \boldsymbol{\chi}, \boldsymbol{\epsilon} \rangle = 0$ and $T = \langle \boldsymbol{\chi}, V \cdot \boldsymbol{M} \rangle$. When that does not happen, it computes

$$\Delta' \leftarrow \frac{T \oplus \langle \boldsymbol{\chi}, V \cdot \boldsymbol{M} \rangle}{\langle \boldsymbol{\chi}, \boldsymbol{\epsilon} \rangle}$$

and makes the protocol abort if and only if $H(\widehat{M} \oplus \Delta', a) \neq H(\widehat{M} \oplus \Delta, a)$ or if Δ' cannot be computed due to a zero denominator.

Observe that the only difference between *Hybrid 1* and $\Pi_{\text{auth-ROT}}$ is that, in the latter, Linear Combination succeeds if and only if $\Delta' = \Delta$, whereas in the former the procedure may succeeds even if $\Delta' \neq \Delta$. Observe that $\widehat{M}$ is uniformly distributed in $\{0,1\}^\lambda$ from the adversary's perspective. So, by the correlation robustness of H, we can substitute $H(\widehat{M} \oplus \Delta', a)$ and $H(\widehat{M} \oplus \Delta, a)$ with the responses of a random oracle. In other words, the probability that the check succeeds when $\Delta' \neq \Delta$ is negligible. Hence, *Hybrid 1* and $\Pi_{\text{auth-ROT}}$ are indistinguishable.

Hybrid 2. Consider now a modified version of the simulator, which, during the initialisation, samples $\widehat{M} \xleftarrow{\$} \{0,1\}^\lambda$. In Linear Combination, the simulator always makes the check succeed if $\langle \boldsymbol{\chi}, \boldsymbol{\epsilon} \rangle = 0$ and $T = \langle \boldsymbol{\chi}, V \cdot \boldsymbol{M} \rangle$. When that does not happen, it computes

$$\Delta' \leftarrow \frac{T \oplus \langle \boldsymbol{\chi}, V \cdot \boldsymbol{M} \rangle}{\langle \boldsymbol{\chi}, \boldsymbol{\epsilon} \rangle}$$

it samples $C \xleftarrow{\$} \{0,1\}^\lambda$ and sends Abort to the functionality, if and only if $H(\widehat{M} \oplus \Delta', a) \neq C$ or if Δ' cannot be computed due to a zero denominator.

Observe that Δ is uniformly distributed in $\{0,1\}^\lambda$ from the adversary perspective. So, in *Hybrid 1*, by the correlation robustness of H, we can substitute X_{1-t} (in ROT) and $H(\widehat{M} \oplus \Delta, a)$ with random elements in $\{0,1\}^\lambda$. Indeed,

$$X_{1-t} = H(K \oplus (t \oplus 1) \cdot \Delta, a) = H(M \oplus \Delta, a).$$

We conclude that *Hybrid 1* and *Hybrid 2* are indistinguishable.

It is rather simple to see that Hybrid 2 and the composition of $\mathcal{S}_{\text{auth-ROT}}$ with $\mathcal{F}_{\text{auth-ROT}}$ are indistinguishable with unconditional security. The only differences are in Linear Combination. Observe that, in Hybrid 2, the check fails with overwhelming probability if $\boldsymbol{r} \neq V \cdot \boldsymbol{t}$. Indeed, in that case $\boldsymbol{\epsilon} \neq \boldsymbol{0}$. Since $\boldsymbol{\chi}$ is random and independent of $\boldsymbol{\epsilon}$, the probability that $\langle \boldsymbol{\chi}, \boldsymbol{\epsilon} \rangle = 0$ is negligible. The same holds for the probability that $H(\widehat{M} \oplus \Delta', a) = C$. When instead, $\boldsymbol{r} = V \cdot \boldsymbol{t}$ but $T \neq \langle \boldsymbol{\chi}, V \cdot \boldsymbol{M} \rangle$, both cases abort with probability 1.

Claim 5.4. *If both the sender and the receiver are honest, by the correlation robustness of H, no PPT adversary can distinguish between protocol and simulation. Moreover, the protocol never aborts.*

Proof of the claim. By Claim 5.2, it is immediate to observe that protocol never aborts. Indeed, $\boldsymbol{\epsilon}$ is always $\boldsymbol{0}$. That also means that Linear Combination is perfectly simulated. We focus our attention on ROT.

Observe that the distribution of t is the same in both protocol and simulation. Moreover, in $\Pi_{\text{auth-ROT}}$, we have $X = H(M, a) = H(K \oplus t \cdot \Delta, a) = X_t$. The equality $X = X_t$ holds in the simulation too.

Since K is uniformly distributed in $\{0,1\}^\lambda$ and independent of t and Δ, by the correlation robustness of H, it is secure to substitute X and X_t with a random element in $\{0,1\}^\lambda$. Finally, since Δ is random and independent of M, again, by the correlation robustness of H, we can substitute

$$X_{1-t} = H(K \oplus (1 \oplus t) \cdot \Delta, a) = H(M \oplus \Delta, a)$$

with a random element in $\{0,1\}^\lambda$. ■

□