Better Preprocessing for Secure Multiparty Computation

Carsten Baum[1], Ivan Damgård[1], Tomas Toft[2], and Rasmus Zakarias[3]*

[1] Department of Computer Science, Aarhus University
[2] Danske Bank
[3] Uber

Abstract. We present techniques and protocols for the preprocessing of secure multiparty computation (MPC), focusing on the so-called SPDZ MPC scheme and its derivatives. These MPC schemes consist of a so-called preprocessing or offline phase, where correlated randomness is generated that is independent of the inputs and the evaluated function, and an online phase, where such correlated randomness is consumed to securely and efficiently evaluate circuits. In the recent years, it has been shown that such protocols turn out to be very efficient in practice.
While much research has been conducted towards optimizing the online phase of the MPC protocols, there seems to have been less focus on the offline phase of such protocols. With this work, we want to close this gap and give a toolbox of techniques that aim at optimizing the preprocessing. We support both instantiations over small fields and large rings using somewhat homomorphic encryption and the Paillier cryptosystem, respectively. In the case of small fields, we show how the preprocessing overhead can basically be made independent of the field characteristic and present a more efficient (amortized) zero-knowledge proof of plaintext knowledge. In the case of large rings, we present a protocol based on the Paillier cryptosystem which has a lower message complexity than previous protocols and employs more efficient zero-knowledge proofs and decryption which, to the best of our knowledge, were not presented in previous work.

Keywords: Efficient Multiparty Computation, Preprocessing, Homomorphic Encryption, Paillier Encryption

1 Introduction

During the recent years, secure two- and multiparty computation [24,35] has evolved from a merely academic research topic into a practical technique for secure function evaluation (see e.g. [6] as an example). Multiparty computation (MPC) aims at solving the following problem: How can a set of parties $\mathtt{P}_1, ..., \mathtt{P}_n$, where each party $\mathtt{P}_i$ has a secret input value x_i, compute a function $y = f(x_1, ..., x_n)$ on their values while not revealing any other information than the output y? Such function could e.g. perform a statistical analysis on the inputs or an online auction or election. Ideally, all these parties would give their secret to a trusted third party (which is incorruptible), that evaluates the function f and reveals the result y to each participant. Such a solution in particular guarantees two properties:

Privacy: Even if malicious parties collude, as long as they cannot corrupt the trusted third party they cannot gain any information except y and what they can derive from it using their inputs.

* This is the full version of the paper that appeared under the same title at the 14th International Conference on Applied Cryptography and Network Security (ACNS) 2016 held in Guildford, United Kingdom. The authors acknowledge support from the Danish National Research Foundation and The National Science Foundation of China (under the grant 61061130540) for the Sino-Danish Center for the Theory of Interactive Computation, within which part of this work was performed; and also from the CFEM research center (supported by the Danish Strategic Research Council) within which part of this work was performed. Work by Carsten Baum was done in part while visiting IDC Herzliya and supported by the ERC under the EU's Seventh Framework Programme (FP/2007-2013) ERC Grant Agreement n. 307952

Correctness: After each party sent their input, there is no way how malicious parties can interfere with the computation of the trusted third party in such a way as to force it to output a specific result y' to the parties that are honest.

MPC replaces such a trusted third party with an interactive protocol among the n parties, while still guaranteeing the above properties. In recent years, it has been shown that even if $n-1$ of the n parties can be corrupted, the efficiency of secure computation can be dramatically improved by splitting the protocol into different phases: During a *preprocessing* or *offline* phase, correlated randomness is generated. This computation is both independent of f and the inputs x_i and can therefore be carried out any time before the actual function evaluation takes place. This way, a lot of the *heavy* computation that relies e.g. on public-key primitives (which we need to handle dishonest majority) will be done beforehand and need not be performed in the later *online* phase, where one can rely on *cheap*, information-theoretic primitives.

In the past years, this approach led to a number of very efficient MPC protocols such as [31,19,16,18,27] to just name a few. In this work, we will primarily focus on variants of the so-called SPDZ protocol [19,16] and their preprocessing phases. They are secure against up to $n-1$ static corruptions, which will also be our adversarial model. For the preprocessing, they rely on very efficient lattice-based homomorphic cryptosystems that allow to perform both additions and multiplications on the encrypted ciphertexts and can pack a large vector of plaintexts into one ciphertext[4]. Unfortunately, the current implementations of the preprocessing has several (non-obvious) drawbacks in terms of efficiency which we try to address in this work:

Incorrect decryption. The complexity of the preprocessing phase naturally depends upon the size of the ring R over which the computation takes place: in the case where $R = \mathbb{Z}/p\mathbb{Z}$ the overhead is inversely proportional to p. The main reason is that implementing the functionality $\mathcal{F}_{\mathsf{KeyGenDec}}^{\text{FAULTY}}$ efficiently without the errors is difficult so that the version with faulty distributed decryption is used in practice. Since the output from the preprocessing depends in part on decryption results, it must be checked for correctness. This is done by sacrificing some part of the computed data to check the remainder. Such an approach only guarantees correctness with probability $1/p$. Hence, especially for small fields, one has to repeat that procedure multiple times which introduces noticeable overhead.

Zero-Knowledge proofs. For each ciphertext, one has to prove plaintext knowledge in order ensure that the ciphertext is *freshly generated*. In implementations it has turned out that the overhead coming from these proofs dominates the message complexity and runtime.

Computation over the integers. If the goal in the end is to do secure computation over the integers, R must be chosen very large to avoid overflow. Here, one could instead investigate the case where $R = \mathbb{Z}/N\mathbb{Z}$ with a preprocessing scheme using Paillier encryption (where N is an RSA number).

1.1 Results and techniques

In this work, the following results will be shown:

(1) We present a novel way of checking the correctness of shared multiplication triples for SHE schemes. In particular, we need to sacrifice only a constant fraction of the data to do the checking, where existing methods need to sacrifice a fraction $\Theta(1 - 1/\kappa)$ for error probability $2^{-\kappa}$.
(2) We redesign the zero-knowledge proof used in [16] and show that one can reduce the number of auxiliary ciphertexts generated for each ciphertext we want to prove knowledge of.
(3) We show how the linearly homomorphic encryption scheme due to Paillier [34,15] can be used more efficiently to produce multiplication triples by representing the data as polynomials and thereby reducing the amount of complex zero-knowledge proofs. Moreover, we also present more efficient zero-knowledge proofs for, e.g., plaintext knowledge, that only require players to work modulo N even if the ciphertexts are defined modulo N^2. Though the technique may already be known, it does not seem to appear in

[4] For more details, see Section 2.2.

previous published work. Additionally, we present a simpler and more efficient distributed decryption method for the Paillier instantiation that is used in our preprocessing.

Before we start to formally present our work, let us have a look at the techniques which we use to achieve our results.

Verifying multiplicative relations. Our goal is (intuitively) to produce encrypted vectors $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}$ such that $\boldsymbol{x} \odot \boldsymbol{y} = \boldsymbol{z}$. Unfortunately, the SPDZ preprocessing can only guarantee that $\boldsymbol{x} \odot \boldsymbol{y} = \boldsymbol{z} + \boldsymbol{\Delta}$ where $\boldsymbol{\Delta}$ can be chosen by the adversary. To counter this, we encode the plaintexts in such a way that we can check the result later: we will let $\boldsymbol{x}, \boldsymbol{y}$ be codewords of a linear code C. This works particularly well because the SPDZ preprocessing encrypts vectors i.e. it has a plaintext space $R = \mathbb{Z}_p^\ell$ where ℓ is the number of instances that each ciphertext contains. We multiply $\boldsymbol{x}$ and $\boldsymbol{y}$ coordinate-wise, so this yields a codeword in a related code (namely its so-called Schur transform C^* of C). Assume that we can, as explained above, efficiently multiply and decrypt, but decryption may be faulty and each party obtains potentially faulty shares of the product. Now we check if $\boldsymbol{z}$ is indeed codeword in the Schur transform. This can be done almost exclusively using linear operations, which are performed without interaction.

Checking whether the result is a codeword is not sufficient, but if $\boldsymbol{z}$ is in the code and not equal to the codeword $\boldsymbol{x} \odot \boldsymbol{y}$, then an adversary would have to have cheated in a large number of positions (the minimum distance of the code). Thus, given the resulting vector $\boldsymbol{z}$ is a codeword, one checks a small number of random positions of the vector to see if it contains the product of corresponding positions in $\boldsymbol{x}$ and $\boldsymbol{y}$. During each check we have a constant probability of catching the adversary, and this quickly amplifies to our desired security levels. The only assumption that we have to make on the underlying field is that appropriate codes with good distance can be defined.

More efficient proofs of plaintext knowledge. To explain our contribution, we need to be more specific about the zero-knowledge proofs of plaintext knowledge (ZKPoPK) suggested in earlier work. First, these are designed for lattice-type cryptosystems that are homomorphic over a finite field, say the field $\mathbb{F}_p$ with p elements for a prime p. Now, the basic step in the protocol is that a player `Pr` chooses a random message m, encrypts it and gives a ZKPoPK to convince the other players that the ciphertext is well-formed and that he knows the plaintext and randomness. If `Pr` is honest, then he will choose m randomly in an interval of size p centered around 0. However, in all known ZKPoPKs it will be the case that if `Pr` is corrupt, we cannot guarantee that the message m known to `Pr` will be in this interval. The best we can do is to force `Pr` to choose m in a somewhat bigger interval. If this interval has size $p \cdot s$ we will say that the ZKPoPK has *soundness slack* s. One would like the soundness slack to be as close to 1 as possible, since this allows us to choose smaller parameters for the underlying cryptosystem and hence improve efficiency.

Protocols are usually designed to prove plaintext knowledge for several input ciphertexts at once and if the prover needs to generate T auxiliary ciphertexts for t input ciphertexts we say that the *ciphertext overhead* of the protocol is T/t. One would of course like this overhead to be as small as possible. We achieve an improvement by extending the proof technique of [16,32]:

(1) We first run a very simple ZKPoPK on each ciphertext with constant soundness error probability. While this will not ensure that all ciphertexts are honestly generated, one can show that almost all of them are good, except with negligible probability.
(2) In a second step, we randomly assign all ciphertexts into squares of width $\sqrt{t}$, compute the row and column sums and prove plaintext knowledge of each such row and column sum. From the previous step, we are guaranteed that most of the ciphertexts are well formed, and can now *explain* the remaining plaintexts as the difference of the row or column sum and the plaintexts of the *good* ciphertexts.

This allows us to reduce the fraction T/t by a factor of 2 for realistic instances (in comparison to the proof technique of [16]) while keeping the soundness slack s as small as in [16]. Our technique might be of independent interest.

Paillier-based preprocessing for SPDZ. Paillier's encryption scheme is only linearly homomorphic, which means that it does not allow to perform multiplications of the plaintexts of two or more ciphertexts directly - it only supports additions but not multiplications. On the other hand, it has an efficient and reliable decryption routine which is what we will make use of. Computing products of encryptions using linearly homomorphic encryption schemes is a well-known technique and can be done using the following protocol Π (we denote an encrypted value x as $\langle\langle x \rangle\rangle$):

(1) $\mathtt{P}_1$ sends some encryption $\langle\langle x_1 \rangle\rangle, \langle\langle y_1 \rangle\rangle$ to $\mathtt{P}_2$ while $\mathtt{P}_2$ sends $\langle\langle x_2 \rangle\rangle, \langle\langle y_2 \rangle\rangle$ to $\mathtt{P}_1$. They want to compute uniformly random values z_1, z_2 where $\mathtt{P}_1$ holds z_1 and $\mathtt{P}_2$ z_2 with the constraint that $(x_1+x_2)\cdot(y_1+y_2) = z_1 + z_2$.
(2) $\mathtt{P}_2$ sends an encryption $\langle\langle z_2' \rangle\rangle = y_2' \cdot \langle\langle x_1 + x_2 \rangle\rangle + \langle\langle -a_2 \rangle\rangle$ to $\mathtt{P}_1$ and proves (among other things) that this y_2' is the same as the plaintext inside $\langle\langle y_2 \rangle\rangle$ (where $\langle\langle a_2 \rangle\rangle$ is an auxiliary, uniformly random value).
(3) $\mathtt{P}_1$ sends $\langle\langle z_1' \rangle\rangle = y_1' \cdot \langle\langle x_1 + x_2 \rangle\rangle + \langle\langle -a_1 \rangle\rangle$ to $\mathtt{P}_2$ and proves a similar statement.
(4) Both use the distributed decryption to safely decrypt the value $z_1' + z_2'$, which does not reveal any information about the product because a_1, a_2 were appropriately uniformly at random.
(5) $\mathtt{P}_1$ sets $z_1 = z_1' + z_2' + a_1$ as her share, while $\mathtt{P}_2$ chooses $z_2 = a_2$.

We consider the setting where many of these multiplications must be performed. Here our approach is, instead of sampling all x_i, y_i independently, to let these be evaluations of a polynomial (that is implicitly defined), and then multiply the factors *unreliably*: instead of giving a zero-knowledge proof that $y_2' = y_2$, we only need to prove that $\mathtt{P}_2$ knows some y_2', a_2, which reduces the complexity of the proof. This means that the result is only correct if all parties honestly follow the multiplication protocol.

The products computed using unreliable multiplication now all lie on a polynomial as well, and using Lagrange interpolation one can evaluate the polynomial in arbitrary points. This can be used to efficiently (and almost locally) check if all products are correct. We want to remark that this approach is asymptotically as efficient as existing techniques, but relies on zero-knowledge proofs with lower message complexity. Moreover, decrypting random values allows for a more efficient decryption routine. It is an interesting open question how these approaches compare in practice.

1.2 Related work

In independent work, Frederiksen et al. showed how to preprocess data for the SPDZ MPC scheme using oblivious transfer [22]. Their approach can make use of efficient OT-extension, but does only allow fields of characteristic 2. While this has some practical applications, it does not generalize (efficiently) to arbitrary fields. On the contrary, our techniques are particularly efficient for other use-cases when binary fields cannot be used to compute the desired function efficiently. Therefore, both results complement each other.

Our technique for checking multiplicative relations is related to the work in [3] for secret shared values in honest majority protocols and in [13] for committed values in 2-party protocols. To the best of our knowledge, this type of technique has not been used before for dishonest majority MPC.

Paillier encryption. The Paillier encryption scheme has been used in MPC preprocessing before such as in [4]. Moreover it was also employed in various MPC schemes such as [12,6,17] to just name a few. The instantiation of the scheme that we use is from [15], and we modify it to use a certain generator which allows for our efficient proofs and decryption.

Proofs for lattice-based encryption schemes. In its simplest case, zero-knowledge proofs for lattice-based encryption schemes can be constructed using Σ-protocols. Such proofs do only achieve soundness error $1/2$ and must therefore be repeated κ times (for κ being the statistical security parameter) to achieve soundness error negligible in κ. Additional care must also be taken to prevent leakage of the plaintext, which normally incurs a 2^κ factor in the soundness slack. Lyubashevsky [30,29] introduced the use of the Fiat-Shamir heuristic together with rejection sampling in the context of such encryption schemes, which allows to drastically reduce the bounds on the plaintexts.

The work by Damgård et al. [19] allows to further reduce the *amortized* cost of ZK proofs, where the authors show how to prove knowledge of κ plaintexts in parallel using $O(\kappa)$ auxiliary ciphertexts. As a drawback, the technique introduces an additional $2^{O(\kappa)}$ overhead on the proven bounds. In contrast, subsequent work [16] allowed much tighter bounds at the expense of a larger ciphertext overhead of the protocol. In comparison to all of the above works (which do also apply to arbitrary LWE encryption schemes), Benhamouda et al. [5] introduced a new technique in the context of Ring-LWE encryption schemes. They expand the size of the challenge space from 2 to $2 \cdot W$ where W is the ring dimension, and show that for their fixed set of challenge polynomials an inverse of small norm always exists. This allows them to obtain proofs with soundness $1/(2 \cdot W)$ while proving plaintext bounds $\tilde{O}(W^2\sigma)$ where σ is the standard deviation of the noise. On the downside, they do not actually achieve a proof of plaintext knowledge, but only of a value related to the plaintext.[5] A different approach was taken by Ling et al. in [28]. They use a technique due to Stern and achieve knowledge error 2/3.

2 Preliminaries

Throughout this work, we assume that a secure point-to-point channels between the parties exist and that a broadcast channel is available. We make commitments abstractly available using the functionality $\mathcal{F}_{\mathsf{Commit}}$ and assume the existence of a random oracle, which will be accessible using the coin-flipping functionality $\mathcal{F}_{\mathsf{Rand}}$.

Functionality $\mathcal{F}_{\mathsf{Rand}}$

Let R be a ring such that there exists a PPT TM to efficiently sample values $r \in R$ uniformly at random.

Random sample: Upon receiving (rand, R) from all parties, it samples a uniform $r \in R$ and outputs (rand, r) to all parties.

Fig. 1. $\mathcal{F}_{\mathsf{Rand}}$: Functionality to sample randomness.

We use $\odot$ for the coordinate-wise multiplication of vectors, $(g, h) = d$ to denote that d is the greatest common divisor of g, h and let $[r]$ be defined as the set $[r] := \{1, \ldots, r\}$. We will denote vectors in bold lower-case letters such as $\boldsymbol{b}$ whereas matrices are bold upper-case letters like $\boldsymbol{M}$. $[\![m]\!], \langle\langle m \rangle\rangle$ denote encryptions of a message m (for the SHE and Paillier encryption, respectively) where the randomness is left implicit. To ease readability, we provide an overview of the most important variables and notation in Table 1.

Functionality $\mathcal{F}_{\mathsf{Commit}}$

Commit: On input $(\mathsf{commit}, v, r, i, j, id_v)$ by $\mathtt{P}_i$, where both v and r are either in R or $\perp$, and id_v is a unique identifier, it stores (v, r, i, j, id_v) on a list and outputs (i, id_v) to $\mathtt{P}_j$.

Open: On input $(\mathsf{open}, i, j, id_v)$ by $\mathtt{P}_i$, the functionality outputs (v, r, i, j, id_v) to $\mathtt{P}_j$. If $(\mathsf{no\_open}, i, id_v)$ is given by the adversary, and $\mathtt{P}_i \in \widehat{\mathtt{P}}$, the functionality outputs $(\perp, \perp, i, j, id_v)$ to $\mathtt{P}_j$.

Fig. 2. $\mathcal{F}_{\mathsf{Commit}}$: Ideal functionality for commitments.

2.1 The SPDZ Multiparty Computation Protocol

We start out with a short primer on the MPC protocol from [19], which we will mostly refer to as SPDZ. This we use not just as motivation for our results, but also to make the reader familiar with the notation.

[5] In LWE, one would (generally) like to prove existence of a short vector $\boldsymbol{s}$ such that $\boldsymbol{As} = \boldsymbol{c}$ where $\boldsymbol{c}$ is the ciphertext and $\boldsymbol{A}$ is some public matrix. What [5] roughly achieve to prove is that there exists an $\boldsymbol{s'}$ such that $2\boldsymbol{c} = 2\boldsymbol{As'} \bmod q$. The vector $\boldsymbol{s'}$ is not guaranteed to be divisible by 2 over $\mathbb{Z}$.

Notation	What it means	Notation	What it means
W	Ring dimension of SHE scheme	$\mathtt{P}_i$	Player i
d	Dimension of noise vector in SHE	$\mathtt{A}$	Adversary
ρ	Bound on honestly generated noise	$\mathtt{Z}$	Environment
τ	Bound on plaintext vector size	n	Number of parties
p	Plaintext modulus	$\widehat{\mathtt{P}}$	Set of bad parties
B_P, B_R	Bounds proven by ZK Proofs	$\mathtt{Pr}$	Prover
$\mathsf{Enc}, \mathsf{Dec}$	Encryption/Decryption	$\mathtt{C}$	Challenger
λ, κ	Computational/statistical security parameter	$\mathtt{Ve}$	Verifier
N	Paillier modulus	α	MAC key for $\langle\cdot\rangle$-representations
$\langle\cdot\rangle$	Value secret shared as in SPDZ	$[\![m]\!], \langle\!\langle m \rangle\!\rangle$	Encryption of m using SHE/Paillier

Table 1. Notation as used in this work.

SPDZ evaluates an arithmetic circuit C over a ring R on a gate-level, where there are addition and multiplication gates. Each value $c \in R$ of the computation (which is assigned to a wire in the process of the evaluation) is MACed using a uniformly random MAC secret MAC key α as $\alpha \cdot c$ and both of these values are then sum-shared among all parties. This MAC key α is fixed for all such shared values, and α is additionally sum-shared among the parties, where party $\mathtt{P}_i$ holds share α_i such that $\alpha = \sum_{i=1}^{n} \alpha_i$.

Definition 1. *Let $r, s, e \in R$, then the $\langle r \rangle$-representation of r is defined as*

$$\langle r \rangle := \big((r_1, \ldots, r_n), (\gamma(r)_1, \ldots, \gamma(r)_n)\big)$$

where $r = \sum_{i=1}^{n} r_i$ and $\alpha \cdot r = \sum_{i=1}^{n} \gamma(r)_i$. Each player $\mathtt{P}_i$ will hold his shares $r_i, \gamma(r)_i$ of such a representation. We define

$$\begin{aligned}
\langle r \rangle + \langle s \rangle &:= \big((r_1 + s_1, \ldots, r_n + s_n), (\gamma(r)_1 + \gamma(s)_1, \ldots, \gamma(r)_n + \gamma(s)_n)\big) \\
e \cdot \langle r \rangle &:= \big((e \cdot r_1, \ldots, e \cdot r_n), (e \cdot \gamma(r)_1, \ldots, e \cdot \gamma(r)_n)\big) \\
e + \langle r \rangle &:= \big((r_1 + e, r_2, \ldots, r_n), (\gamma(r)_1 + e \cdot \alpha_1, \ldots, \gamma(r)_n + e \cdot \alpha_n) \big)
\end{aligned}$$

This representation is closed under linear operations:

Remark 1. Let $r, s, e \in R$. We say that $\langle r \rangle \hat{=} \langle s \rangle$ if both $\langle r \rangle, \langle s \rangle$ reconstruct to the same value. Then it holds that

$$\langle r \rangle + \langle s \rangle \hat{=} \langle r + s \rangle \quad \text{and} \quad e \cdot \langle r \rangle \hat{=} \langle e \cdot r \rangle \quad \text{and} \quad e + \langle r \rangle \hat{=} \langle e + r \rangle$$

In order to multiply two representations, we rely on a technique due to Beaver [2]: Let $\langle r \rangle, \langle s \rangle$ be two values where we want to calculate a representation $\langle t \rangle$ such that $t = r \cdot s$. Assume the availability of a triple[6] $(\langle a \rangle, \langle b \rangle, \langle c \rangle)$ such that a, b are uniformly random and $c = a \cdot b$. To obtain $\langle t \rangle$, one can use the procedure as depicted in Fig. 3. Correctness and privacy of this procedure were established before, e.g. in [19]. This

Procedure Mult

Multiply$(\langle r \rangle, \langle s \rangle, \langle a \rangle, \langle b \rangle, \langle c \rangle)$**:**

(1) The players calculate $\langle \gamma \rangle = \langle r \rangle - \langle a \rangle, \langle \delta \rangle = \langle s \rangle - \langle b \rangle$.
(2) The players publicly reconstruct γ, δ.
(3) Each player locally calculates $\langle t \rangle = \langle c \rangle + \delta \langle a \rangle + \gamma \langle b \rangle + \gamma\delta$.
(4) Return $\langle t \rangle$ as the representation of the product.

Fig. 3. Mult: Procedure to generate the product of two $\langle\cdot\rangle$-shared values.

[6] We will also refer to those triples as *multiplication triples* throughout this paper.

already allows to compute on shared values, and inputting information into such a computation can also easily be achieved using standard techniques[7]. Checking that a value was indeed reconstructed correctly will be done using Π_{MacCheck} which allows to check the MAC of the opened value without revealing the key α.

Protocol Π_{MacCheck}

CheckOutput$(v_1, \ldots, v_t, R)$**:**

(1) The parties sample a vector $\boldsymbol{r} \in R^t$ using $\mathcal{F}_{\mathsf{Rand}}$.
(2) Each $\mathtt{P}_i$ computes $v = \sum_{i=1}^{t} \boldsymbol{r}[i] \cdot v_i$ and $\gamma_i = \sum_{j=1}^{t} \boldsymbol{r}[j] \cdot \gamma(v_j)$.
(3) Each $\mathtt{P}_i$ computes $\sigma_i = \gamma_i - \alpha_i \cdot v$ and commits to σ_i using $\mathcal{F}_{\mathsf{Commit}}$ as c_i'.
(4) Each c_i' is opened towards all players using $\mathcal{F}_{\mathsf{Commit}}$.
(5) Output $\sigma = \sum_{i=1}^{n} \sigma_i$.

Fig. 4. Π_{MacCheck}: Protocol to check validity of MACs.

This checking procedure will fail to detect an incorrect reconstruction with probability at most $2/p$ over fields of characteristic p, and similarly with probability $2/q$ over rings $\mathbb{Z}/N\mathbb{Z}$ where q is the smallest prime factor of N. This is captured by the following lemma which we will need at multiple points in this work:

Lemma 1. *Assume that Π_{MacCheck} is executed over the field $\mathbb{Z}/p\mathbb{Z}$. The protocol Π_{MacCheck} is correct and sound: It returns $\sigma = 0$ if all the values v_i and their corresponding MACs $\gamma(v_i)$ are correctly computed and outputs $\sigma \neq 0$ except with probability $2/p$ in the case where at least one value or MAC is not correctly computed.*

Proof. See e.g. [16].

The online phase of the SPDZ protocol can be realized from the linear properties of $\langle \cdot \rangle$ as well as $\Pi_{\mathsf{MacCheck}}, \mathsf{Mult}$. In order to run the protocol, it is necessary to have both random values $\langle r \rangle$ for the inputs as well as triples $(\langle a \rangle, \langle b \rangle, \langle c \rangle)$ for Mult. Both of these, as well as the sharing of the MAC key α, come from a preprocessing phase as depicted in Fig. 5.

2.2 Somewhat Homomorphic Encryption

In [19,16] the preprocessing functionality, i.e. $\mathcal{F}_{\mathsf{Offline}}$, is realized using a protocol based on *Somewhat Homomorphic Encryption*(SHE). In the following, we will give a quick definition of what we mean by an SHE scheme.

Let $\mathcal{M} = \mathbb{F}^\ell = \mathbb{Z}_p^\ell$ be the direct product of ℓ $\mathbb{F}$-instances, where '+' and '$\cdot$' are the ring operations in $\mathcal{M}$ implied by the direct product. Moreover, consider $\mathcal{A} \approx \mathbb{Z}^W$ for some integer $W \in \mathbb{N}^+$ as an intermediate space. For $\mathcal{A}$, we define the $||\cdot||_\infty$-norm in the usual way. Encryption will work as a map from $\mathcal{A}$ to some additive abelian group $\mathcal{B}$ that has the additive operation $\boxplus$ and an operation $\boxtimes$ that is not necessarily closed, but commutative and distributive. The operations of $\mathcal{A}$ will also be denoted as '+','$\cdot$'. Addition will be component-wise, whereas there is no restriction on how the multiplication is realized.

In order to map $\boldsymbol{m} \in \mathcal{M}$ to an element $\boldsymbol{a} \in \mathcal{A}$ and back, there exist the two functions

$$\iota : \mathcal{M} \to \mathcal{A} \text{ and } \eta : \mathcal{A} \to \mathcal{M}$$

where ι is injective. The ring operations from $\mathcal{M}$ must carry over to a certain degree:

(1) $\forall \boldsymbol{m} \in \mathcal{M} : \eta(\iota(\boldsymbol{m})) = \boldsymbol{m}$
(2) $\forall \boldsymbol{m}_1, \boldsymbol{m}_2 \in \mathcal{M} : \eta(\iota(\boldsymbol{m}_1) + \iota(\boldsymbol{m}_2)) = \boldsymbol{m}_1 + \boldsymbol{m}_2$
(3) $\forall \boldsymbol{m}_1, \boldsymbol{m}_2 \in \mathcal{M} : \eta(\iota(\boldsymbol{m}_1) \cdot \iota(\boldsymbol{m}_2)) = \boldsymbol{m}_1 \cdot \boldsymbol{m}_2$
(4) $\forall \boldsymbol{a} \in \mathcal{A} : \eta(\boldsymbol{a}) = \eta(\boldsymbol{a} \bmod p)$
(5) $\forall \boldsymbol{m} \in \mathcal{M} : ||\iota(\boldsymbol{m})||_\infty \leq \tau$ with $\tau = p/2$

[7] Open a random value $\langle r \rangle$ to a party that wants to input x. That party then broadcasts $x - r$ and the parties jointly compute $(x - r) + \langle r \rangle = \langle x \rangle$.

Functionality $\mathcal{F}_{\mathsf{Offline}}$

This functionality generates a shared MAC key α and $\langle\cdot\rangle$-representations and multiplication triples.

Initialize: On input (Init, R) from all players, the functionality stores a description of the ring R. $\mathtt{A}$ chooses the set of parties $\widehat{\mathtt{P}} \subset \mathtt{P}$ he corrupts.

(1) For all $\mathtt{P}_i \in \widehat{\mathtt{P}}$, $\mathtt{A}$ inputs $\alpha_i \in R$, while for all $\mathtt{P}_i \notin \widehat{\mathtt{P}}$, the functionality chooses $\alpha_i \xleftarrow{\$} R$.
(2) Set the key $\alpha = \sum_{i=1}^{n} \alpha_i$ and send α_i to $\mathtt{P}_i \notin \widehat{\mathtt{P}}$.

$\mathsf{SpdzRep}(\boldsymbol{r}_1, \ldots, \boldsymbol{r}_n, \alpha, \boldsymbol{\Delta}_{\gamma,\boldsymbol{r}}, m)$**:** This macro will be run to create $\langle\cdot\rangle$-representations.

(1) Define $\boldsymbol{r} = \sum_{i=1}^{n} \boldsymbol{r}_i$.
(2) For $\mathtt{P}_i \in \widehat{\mathtt{P}}$, $\mathtt{A}$ inputs $\gamma(\boldsymbol{r})_i \in R^m$, and for $\mathtt{P}_i \notin \widehat{\mathtt{P}}$, the functionality chooses $\gamma(\boldsymbol{r})_i \xleftarrow{\$} R^m$ except for $\gamma(\boldsymbol{r})_j$, with j being the smallest index not in $\widehat{\mathtt{P}}$.
(3) Set $\gamma(\boldsymbol{r}) = \alpha \cdot \boldsymbol{r} + \boldsymbol{\Delta}_{\gamma,\boldsymbol{r}}$ and $\gamma(\boldsymbol{r})_j = \gamma(\boldsymbol{r}) - \sum_{j \neq i=1}^{n} \gamma(\boldsymbol{r})_i$. For every honest party $\mathtt{P}_i$, send $\gamma(\boldsymbol{r})_i$ to $\mathtt{P}_i$.
(4) Define $\langle \boldsymbol{r} \rangle = (\boldsymbol{r}_1, \ldots, \boldsymbol{r}_n, \gamma(\boldsymbol{r})_1, \ldots, \gamma(\boldsymbol{r})_n)$. Return $\langle \boldsymbol{r} \rangle$.

Input: On input (Input, n_I) from all parties, the functionality does the following:

(1) For $\mathtt{P}_i \notin \widehat{\mathtt{P}}$, the functionality samples $\boldsymbol{r}_i \xleftarrow{\$} R^{n_I}$.
(2) For $\mathtt{P}_i \in \widehat{\mathtt{P}}$, $\mathtt{A}$ inputs $\boldsymbol{r}_i, \boldsymbol{\Delta}_{\gamma,\boldsymbol{r}} \in R^{n_I}$.
(3) Define $\boldsymbol{r} = \sum_{j=1}^{n} \boldsymbol{r}_j$. Run the macro $\langle \boldsymbol{r} \rangle \leftarrow \mathsf{SpdzRep}(\boldsymbol{r}_1, \ldots, \boldsymbol{r}_n, \alpha, \boldsymbol{\Delta}_{\gamma,\boldsymbol{r}}, n_I)$.
(4) Return $\langle \boldsymbol{r} \rangle$.

Triples: On input (Triple, n_M) from all parties, the functionality does the following:

(1) For $\mathtt{P}_i \in \widehat{\mathtt{P}}$, $\mathtt{A}$ inputs $\boldsymbol{a}_i, \boldsymbol{b}_i, \boldsymbol{c}_i, \boldsymbol{\Delta}_{\gamma,\boldsymbol{a}}, \boldsymbol{\Delta}_{\gamma,\boldsymbol{b}}, \boldsymbol{\Delta}_{\gamma,\boldsymbol{c}} \in R^{n_M}$. For $\mathtt{P}_i \notin \widehat{\mathtt{P}}$, the functionality samples $\boldsymbol{a}_i, \boldsymbol{b}_i \xleftarrow{\$} R^{n_M}$.
(2) Define $\boldsymbol{a} = \sum_{j=1}^{n} \boldsymbol{a}_j, \boldsymbol{b} = \sum_{j=1}^{n} \boldsymbol{b}_j$.
(3) Let $\mathtt{P}_j \notin \widehat{\mathtt{P}}$ be the smallest index of an honest player. For all $\mathtt{P}_i \notin \widehat{\mathtt{P}}, i \neq j$ choose $\boldsymbol{c}_i \xleftarrow{\$} R^{n_M}$. For $\mathtt{P}_j$ let $\boldsymbol{c}_j = \boldsymbol{a} \odot \boldsymbol{b} - \sum_{i \in [n], i \neq j} \boldsymbol{c}_i$. Send $\boldsymbol{a}_i, \boldsymbol{b}_i, \boldsymbol{c}_i$ to each honest $\mathtt{P}_i$.
(4) Run the macros

$$\langle \boldsymbol{a} \rangle \leftarrow \mathsf{SpdzRep}(\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n, \alpha, \boldsymbol{\Delta}_{\gamma,\boldsymbol{a}}, n_M),$$
$$\langle \boldsymbol{b} \rangle \leftarrow \mathsf{SpdzRep}(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n, \alpha, \boldsymbol{\Delta}_{\gamma,\boldsymbol{b}}, n_M),$$
$$\langle \boldsymbol{c} \rangle \leftarrow \mathsf{SpdzRep}(\boldsymbol{c}_1, \ldots, \boldsymbol{c}_n, \alpha, \boldsymbol{\Delta}_{\gamma,\boldsymbol{c}}, n_M).$$

(5) Return $(\langle \boldsymbol{a} \rangle, \langle \boldsymbol{b} \rangle, \langle \boldsymbol{c} \rangle)$.

Fig. 5. $\mathcal{F}_{\mathsf{Offline}}$: Functionality $\mathcal{F}_{\mathsf{Offline}}$ for the preprocessing of SPDZ.

Algorithms. Assume that $\mathcal{M}, \mathcal{A}, \mathcal{B}$ are defined as above for a fixed parameter set. To sample *noise* for the ciphertexts, we define the efficient polynomial time algorithm D_ρ^d, which outputs vectors $\boldsymbol{r} \in \mathbb{Z}^d$ such that $\Pr[||\boldsymbol{r}||_\infty \geq \rho \mid \boldsymbol{r} \leftarrow \mathrm{D}_\rho^d] < \mathsf{negl}(\lambda)$. We let C be the set of arithmetic *Single Instruction Multiple Data* (SIMD) circuits over $\mathbb{F}^\ell$. The SIMD property implies that there exists a function $f \in \mathbb{F}[X_1, \ldots, X_{n(f)}]$ such that $\widehat{f} \in C$ evaluates the function f ℓ times on inputs in $\mathbb{F}^{n(f)}$ in parallel.

For the specified algebraic structures, define the probabilistic polynomial-time algorithms $\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec}$ that represent the cryptosystem H as follows:

KG : This algorithm samples a public-key/private-key pair $(\mathsf{pk}, \mathsf{sk})$.

$\mathsf{Enc}_{\mathsf{pk}}(\boldsymbol{x}, \boldsymbol{r})$: Let $\boldsymbol{x} \in \mathcal{A}$ and $\boldsymbol{r} \in \mathbb{Z}^d$ then this algorithm creates a $g \in \mathcal{B}$ deterministically. Enc must be additively homomorphic for at least a *small* number t of correctly formed ciphertexts on both plaintexts and randomness: Let $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_t \in \mathit{image}(\iota)$, $\boldsymbol{r}_1, \ldots, \boldsymbol{r}_t \leftarrow \mathrm{D}_\rho^d$. Then it holds that

$$\mathsf{Enc}_{\mathsf{pk}}(\boldsymbol{x}_1 + \cdots + \boldsymbol{x}_t, \boldsymbol{r}_1 + \cdots + \boldsymbol{r}_t) = \mathsf{Enc}_{\mathsf{pk}}(\boldsymbol{x}_1, \boldsymbol{r}_1) \boxplus \ldots \boxplus \mathsf{Enc}_{\mathsf{pk}}(\boldsymbol{x}_t, \boldsymbol{r}_t)$$

$\mathsf{Dec}_{\mathsf{sk}}(g)$: For $g \in \mathcal{B}$ this algorithm will return an $\boldsymbol{m} \in \mathcal{M} \cup \{\perp\}$.

Correctness. Let $n(f), f \in C$ be the number of input values of f and let $\widehat{f}$ be the embedding of f into $\mathcal{B}$ where '+' is replaced by $\boxplus$, '$\cdot$' by $\boxtimes$ and the constant $c \in \mathbb{F}$ by $\mathsf{Enc}_{\mathsf{pk}}(\iota(c \cdot \mathbf{1}), \mathbf{0})$. For data vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{n(f)}$, let $f(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{n(f)})$ be the SIMD application of f to this data.

Definition 2 (Correctness). $H = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ *is called* (B_P, B_R, C)*-correct if*

$$\Pr\left[\mathsf{Dec}_{\mathsf{sk}}(\boldsymbol{c}) \neq f\left(\eta(\boldsymbol{x}_1), \ldots, \eta(\boldsymbol{x}_{n(f)})\right) \middle| \begin{array}{l}(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KG}(1^\lambda) \wedge f \in C \wedge \boldsymbol{c} \leftarrow \widehat{f}(\boldsymbol{c}_1, \ldots, \boldsymbol{c}_{n(f)}) \wedge \\ \left[(\boldsymbol{x}_i, \boldsymbol{r}_i) \in \mathcal{A} \times (\mathbb{Z}^d) \wedge \boldsymbol{c}_i \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\boldsymbol{x}_i, \boldsymbol{r}_i) \wedge\right]_{i \in [n(f)]} \\ \eta(\boldsymbol{x}_i) \in \mathcal{M} \wedge ||\boldsymbol{x}_i||_\infty \leq B_P \wedge ||\boldsymbol{r}_i||_\infty \leq B_R\end{array}\right] \leq \mathsf{negl}(\lambda)$$

For SPDZ one would additionally require that the encryption scheme has a *distributed decryption and key generation* procedure, but we do not specify these here since we will not make use of these functionalities. With this at hand, we can define our cryptosystem as follows:

Definition 3 (Somewhat Homomorphic Cryptosystem). *Let C contain formulas of the form* $\left(\sum_{i=1}^n x_i\right) \cdot \left(\sum_{i=1}^n y_i\right) + \sum_{i=1}^n z_i$ *for $n \in \mathbb{N}$. Let $H = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ be an IND-CPA secure cryptosystem. H is called somewhat homomorphic if it is (B_P, B_R, C)-correct for fixed B_P, B_R.*

One can easily see that e.g. the Ring-LWE-based BGV scheme [8] or the BGH extension of LWE-based BGV [7] have the required features. The more recent matrix-based cryptosystems like the GSW scheme [23,9] do unfortunately have no SIMD property and are in practice outperformed by the above schemes.

2.3 Zero-Knowledge Proofs of Plaintext Knowledge for SHE

Two different flavors of amortized zero-knowledge proofs were used before in the context of the above cryptosystem H and preprocessing for MPC. The first technique was described in [19] and follows an idea due to [11,4]. Another approach was introduced in [16] and uses a LEGO-like [32] argument.

The proofs will have statistical security parameter κ. We prove plaintext knowledge of the set $\mathcal{S}_P = \{\boldsymbol{c}_1, \ldots, \boldsymbol{c}_t\}$ of ciphertexts. More formally, one shows that the following relation holds:

$$R^H_{\text{PoPK}} = \left\{ (\boldsymbol{a}, \boldsymbol{w}) \middle| \begin{array}{c}\boldsymbol{a} = (\boldsymbol{c}_1, \ldots, \boldsymbol{c}_t, \mathsf{pk}) \wedge \boldsymbol{w} = (\boldsymbol{x}_1, \boldsymbol{r}_1, \ldots, \boldsymbol{x}_t, \boldsymbol{r}_t) \wedge \\ \left[\begin{array}{c}\boldsymbol{c}_i = \mathsf{Enc}_{\mathsf{pk}}(\boldsymbol{x}_i, \boldsymbol{r}_i) \wedge \eta(\boldsymbol{x}_i) \in \mathcal{M} \wedge \\ ||\boldsymbol{x}_i||_\infty \leq B_P \wedge ||\boldsymbol{r}_i||_\infty \leq B_R\end{array}\right]_{i \in [t]}\end{array} \right\}$$

In this work, we will use an approach that is an optimization of the proof technique from [16]. The protocol uses a larger number of auxiliary ciphertexts, but the gap between (τ, ρ) and B_P, B_R is polynomial (whereas [19] has a soundness slack that is exponential in κ). In order to prove plaintext knowledge, one uses a set $\mathcal{S}_A$ of T auxiliary ciphertexts where t divides T. This set is obtained using the procedure $\mathsf{CutAndChoose}$, where first $2T$ auxiliary ciphertexts are generated by $\mathtt{Pr}$, among which T are opened. These T opened ciphertexts are chosen uniformly at random and $\mathtt{Ve}$ checks that they are all formed correctly. The remaining, T unopened auxiliary ciphertexts are now the set $\mathcal{S}_A$ and only a small subset of the remaining ciphertexts may not be well formed. $\mathtt{Ve}$ then randomly assigns the T auxiliary ciphertexts into t buckets, and by a standard argument the probability that all ciphertexts in each bucket are not generated correctly is negligible. $\mathtt{Pr}$ in turn, for bucket i and $\boldsymbol{c}_i \in \mathcal{S}_P$, opens the sum $\boldsymbol{c}_i + \boldsymbol{a}_j$ for all j in bucket i and $\mathtt{Ve}$ checks correctness.

This proof requires to generate $2T/t$ auxiliary ciphertext per proven plaintext. In particular for a *practical* number of ciphertexts (think $t \approx 40$) this makes implementations quite slow (due to the demand in RAM).

2.4 (Reed-Solomon) codes

Let $p, k, m \in \mathbb{N}^+, m > k$ and p be a prime. Consider the two vector spaces $\mathbb{F}_p^k, \mathbb{F}_p^m$ and a monomorphism $C : \mathbb{F}_p^k \to \mathbb{F}_p^m$ together as a *code*, i.e. $\boldsymbol{c} = C(\boldsymbol{x})$ as an encoding of $\boldsymbol{x}$ in $\mathbb{F}_p^m$. We assume that it is efficiently decidable whether $\boldsymbol{c}' \in C$ (error detection), where

$$\boldsymbol{c}' \in C \Leftrightarrow \exists \boldsymbol{x}' \in \mathbb{F}_p^k : \; C(\boldsymbol{x}') = \boldsymbol{c}',$$

Procedure CutAndChoose

CutAndChoose(T, B_P, B_R):

(1) $\texttt{Pr}$ calls the RO with seeds $f_1, \ldots, f_{2T}$. From the output, it generates $\boldsymbol{s}_i, \boldsymbol{y}_i$ deterministically as follows:
  (1.1) Choose $\boldsymbol{s}_i$ uniformly at random with $||\boldsymbol{s}_i||_\infty \leq B_R$.
  (1.2) Let $\boldsymbol{m}_i \in \mathcal{M}$ be a random element and set $\boldsymbol{y}_i = \iota(\boldsymbol{m}_i) + \boldsymbol{u}_i$ where $\boldsymbol{u}_i$ is generated such that each entry is a uniformly random multiple of p subject to the constraint that $||\boldsymbol{y}_i||_\infty \leq B_P$.
(2) For $i \in [2T]$ $\texttt{Pr}$ computes $\boldsymbol{a}_i = \mathsf{Enc}_{\mathsf{pk}}(\boldsymbol{y}_i, \boldsymbol{s}_i)$ and sends $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_{2T}$ to $\texttt{Ve}$.
(3) $\texttt{Pr}, \texttt{Ve}$ sample a set $V \subset [2T]$ of size T using $\mathcal{F}_{\mathsf{Rand}}$.
(4) For all $i \in V$, $\texttt{Pr}$ sends f_i to $\texttt{Ve}$ who verifies that they induce the ciphertexts $\boldsymbol{a}_i$ as generated in Step (2) and that $||\boldsymbol{y}_i||_\infty \leq B_P$ and $||\boldsymbol{s}_i||_\infty \leq B_R$. If one of the checks does not hold, then $\texttt{Ve}$ aborts.
(5) Let $[2T] \setminus V = \{i_1, \ldots, i_T\}$. Output $(\boldsymbol{a}_{i_1}, \boldsymbol{y}_{i_1}, \boldsymbol{s}_{i_1}, \ldots, \boldsymbol{a}_{i_T}, \boldsymbol{y}_{i_T}, \boldsymbol{s}_{i_T})$.

Fig. 6. CutAndChoose: Procedure for generating auxiliary ciphertexts using cut and choose.

and the Hamming distance d of two codewords $\boldsymbol{x}, \boldsymbol{y} \in C$ should be large (meaning that the difference of any two distinct codewords should be non-zero in as many positions as possible). Such a code is called an $[m, k, d]$ code.

If, for every message $\boldsymbol{x} \in \mathbb{F}_p^k$ the message $\boldsymbol{x}$ reappears directly in $C(\boldsymbol{x})$ then the code is called systematic. Without loss of generality, one can assume that the first m positions of a codeword are equal to the encoded message in that case. The mapping of C can be represented as multiplication with a matrix $\boldsymbol{G}$ (called the *generator matrix*), and one can write the encoding procedure as $C : \boldsymbol{x} \mapsto \boldsymbol{G}\boldsymbol{x}$ where $\boldsymbol{G} \in \mathbb{F}_p^{m \times k}$. Similarly, we assume the existence of a *check matrix* $\boldsymbol{H} \in \mathbb{F}_p^{(m-k) \times m}$ where $\boldsymbol{H}\boldsymbol{x} = \boldsymbol{0} \Leftrightarrow \boldsymbol{x} \in C$.

For a $[m, k, d]$ code C, define the *Schur transform* (as in [18]) as $C^* = span(\{\boldsymbol{x} \odot \boldsymbol{y} \mid \boldsymbol{x}, \boldsymbol{y} \in C\})$. C^* is itself a code where the message length k' cannot be smaller than k. On the contrary, C^* has a smaller minimum distance $d' \leq d$. The actual values k', d' depend on the properties of the code C.

An example for a code with small loss $d - d'$ with respect to the Schur transform (as we shall see later) is the *Reed-Solomon* code, where the encoding C works as follows: for $a, b \in \mathbb{N}^+$ define the matrix

$$V_a(z_1, \ldots, z_b) := \begin{pmatrix} 1 & z_1 & z_1^2 & \ldots & z_1^a \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & z_b & z_b^2 & \ldots & z_b^a \end{pmatrix}.$$

Fix pairwise distinct and non-zero $z_1, \ldots, z_k \in \mathbb{F}_p$ and define the matrices $\boldsymbol{A_1} = V_k(z_1, \ldots, z_k)^{-1}$ and $\boldsymbol{A_2} = V_m(z_1, \ldots, z_k)$. Define the encoding as

$$\begin{aligned} C : \mathbb{F}_p^k &\to \mathbb{F}_p^m \\ \boldsymbol{x} &\mapsto \boldsymbol{A_2}\boldsymbol{A_1}\boldsymbol{x}. \end{aligned}$$

This encoding can be made efficient since the matrices are decomposable for certain values $z_1, \ldots, z_k$ using the *Fast Fourier Transform* (FFT). Decoding can be done in a similar fashion, be we do not discuss it here.

The intuition behind the encoding procedure is as follows: the k values uniquely define a polynomial f of degree at most $k - 1$, whose coefficients can be computed using $\boldsymbol{A_1}$ (as an inverse FFT). One evaluates the polynomial in the remaining $m - k$ positions using $\boldsymbol{A_2}$. The minimum distance d is exactly $m - k + 1$, since two polynomials of degree at most $k - 1$ are equal if they agree in at least k positions. Now, by letting $\boldsymbol{A_2}$ be another FFT matrix, the point-wise multiplication of codewords from C yields a codeword in C^* which is a polynomial of degree at most $2(k-1)$ and the code C^* therefore has minimum distance $d' = m - 2k + 1$.

2.5 The Paillier cryptosystem

We use the Paillier encryption scheme $P = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ as defined in [34,15] (with some practical restrictions). Let $N = p \cdot q$ be the product of two odd, τ-bit safe primes with $(N, \phi(N)) = 1$ (we choose τ such

that the scheme has λ bit security). We define Paillier encryption of a message $x \in \mathbb{Z}/N\mathbb{Z}$ with randomness $r \in \mathbb{Z}/N\mathbb{Z}^*$ is defined as

$$\langle\langle x \rangle\rangle := P.\mathsf{Enc}_{\mathsf{pk}}(x, r) = r^N \cdot (N+1)^x \bmod N^2.$$

Knowing the factorization of N allows decryption of ciphertext $c \in \mathbb{Z}/N^2\mathbb{Z}^*$, e.g., by determining the randomness used,

$$r = c^{N^{-1} \bmod \phi(N)} \bmod N.$$

The decryption $P.\mathsf{Dec}$ then proceeds as

$$x = \left(\left(c \cdot r^{-N} \bmod N^2\right) - 1\right)/N \bmod N.$$

The key generation algorithm $P.\mathsf{KG}$ samples an RSA modulus $N = p \cdot q$, lets the public key be $\mathsf{pk} = (N)$ and the secret key be $\mathsf{sk} = (p, q, f = N^{-1} \bmod \varphi(N))$. The encryption scheme is additively homomorphic and IND-CPA secure assuming the Composite Residuosity problem $CR[N]$ is hard (see [34] for details).

Functionality $\mathcal{F}^{\mathrm{P}}_{\mathsf{KeyGenDec}}$

Key Generation:

(1) On input $(\mathsf{Keygen}, \tau, \kappa)$ by all parties, compute $(N, \mathsf{sk}) \leftarrow P.\mathsf{KG}(1^\lambda)$.
(2) Sample $\mathsf{sk}_2, \ldots, \mathsf{sk}_n \xleftarrow{\$} \mathbb{Z}/2^\kappa N\mathbb{Z}$ and choose $\mathsf{sk}_1 \in \mathbb{Z}/2^\kappa N\mathbb{Z}$ such that $\mathsf{sk} = \sum_i \mathsf{sk}_i \bmod \varphi(N)$.
(3) Output (N, sk_i) to party $\mathtt{P}_i$.

Distributed Decryption:

(1) When receiving $(\mathsf{Decrypt}, c)$ from all players, check whether there exists a shared key pair (N, sk). If not, return $\perp$.
(2) We now let the parties decide how to decrypt:
  – Upon receiving (Rand) from all players, send $(x, r) \leftarrow P.\mathsf{Dec}_{\mathsf{sk}}(c)$ to $\mathtt{A}$. Upon receiving $x^* \in \{(x, r), \perp\}$ from $\mathtt{A}$, send (Result, x^*) to all players.
  – Upon receiving (NoRand) from all players, send $(x, ?) \leftarrow P.\mathsf{Dec}_{\mathsf{sk}}(c)$ to $\mathtt{A}$. Upon receiving $x^* \in \{x, \perp\}$ from $\mathtt{A}$, send (Result, x^*) to all players.

Fig. 7. $\mathcal{F}^{\mathrm{P}}_{\mathsf{KeyGenDec}}$: Functionality that provides shared keys and decrypts ciphertexts for Paillier encryption.

For the purpose of using this encryption scheme in an interactive protocol, assume the existence of a functionality $\mathcal{F}^{\mathrm{P}}_{\mathsf{KeyGenDec}}$ as in Fig. 7. As mentioned in the introduction, we later implement the **Distributed Decryption** based on a functionality $\mathcal{F}^{\mathrm{P}}_{\mathsf{KeyGen}}$ that only contains the **Key Generation** part of $\mathcal{F}^{\mathrm{P}}_{\mathsf{KeyGenDec}}$ [8]. Our **Distributed Decryption** follows directly from the decryption formula given above where one does completely recover the randomness used during encryption. This is not always desirable and the functionality therefore provides an option whether this should be done or not. Observe that in this work, we will only use decryption that reveals the randomness and provide the other option only for completeness.

3 More efficient Preprocessing from Somewhat Homomorphic Encryption

In this section, we present an improved preprocessing protocol for SPDZ over fields. Towards achieving this, we overhaul the triple generation in a way that allows more efficient checks of correctness. This check uses the original SPDZ preprocessing as a black box (see Fig. 8) plus the aforementioned codes which we will describe in more detail. The triple check introduces some computational overhead on top of $\mathcal{F}^{\mathrm{FAULTY}}_{\mathsf{Offline}}$, but we show how this overhead can be reduced.

[8] A protocol for such a key sampling can be implemented using an arbitrary MPC scheme.

3.1 The offline phase of SPDZ

We already mentioned in Section 2.2 that the preprocessing of [19] uses a SHE scheme for the preprocessing. Unfortunately, this scheme H only has a *potentially faulty* distributed decryption procedure. Therefore, the actual output of the preprocessing rather looks like that depicted in Fig. 8.

Functionality $\mathcal{F}_{\mathsf{Offline}}^{\text{FAULTY}}$

This functionality generates a shared MAC key α and (potentially faulty) $\langle\cdot\rangle$-representations.

Initialize: On input $(\mathsf{Init}, \mathbb{F}, C)$ from all players, store a description of $\mathbb{F}$ and of the code C.

(1) For all $\mathtt{P}_i \in \widehat{\mathtt{P}}$, $\mathtt{A}$ inputs $\alpha_i \in \mathbb{F}$, while for all $i \notin \widehat{\mathtt{P}}$, choose $\alpha_i \xleftarrow{\$} \mathbb{F}$.
(2) Set the key $\alpha = \sum_{i=1}^{n} \alpha_i$ and send α_i to $\mathtt{P}_i \notin \widehat{\mathtt{P}}$.

Triples: On input (Triple) from all parties, the functionality does the following to generate triples:

(1) For $\mathtt{P}_i \notin \widehat{\mathtt{P}}$, sample $\boldsymbol{a}_i, \boldsymbol{b}_i \xleftarrow{\$} C$.
(2) For $\mathtt{P}_i \in \widehat{\mathtt{P}}$, $\mathtt{A}$ inputs $\boldsymbol{a}_i, \boldsymbol{b}_i, \boldsymbol{c}_i, \boldsymbol{\delta}, \boldsymbol{\Delta}_{\gamma,\boldsymbol{a}}, \boldsymbol{\Delta}_{\gamma,\boldsymbol{b}}, \boldsymbol{\Delta}_{\gamma,\boldsymbol{c}} \in \mathbb{F}^m$. If $\boldsymbol{a}_i, \boldsymbol{b}_i \notin C$ then stop.
(3) Define $\boldsymbol{a} = \sum_{j=1}^{n} \boldsymbol{a}_j, \boldsymbol{b} = \sum_{j=1}^{n} \boldsymbol{b}_j$.
(4) Let $\mathtt{P}_j \notin \widehat{\mathtt{P}}$ be the smallest index of an honest player. For all $\mathtt{P}_i \notin \widehat{\mathtt{P}}, i \neq j$ choose $\boldsymbol{c}_i \xleftarrow{\$} \mathbb{F}^m$. For $\mathtt{P}_j$ let $\boldsymbol{c}_j = \boldsymbol{a} \odot \boldsymbol{b} + \boldsymbol{\delta} - \sum_{i\in[n], i\neq j} \boldsymbol{c}_i$. Send $\boldsymbol{a}_i, \boldsymbol{b}_i, \boldsymbol{c}_i$ to each honest $\mathtt{P}_i$.
(5) Run the macros

$$\langle \boldsymbol{a} \rangle \leftarrow \mathsf{SpdzRep}(\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n, \alpha, \boldsymbol{\Delta}_{\gamma,\boldsymbol{a}}, m),$$
$$\langle \boldsymbol{b} \rangle \leftarrow \mathsf{SpdzRep}(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n, \alpha, \boldsymbol{\Delta}_{\gamma,\boldsymbol{b}}, m),$$
$$\langle \boldsymbol{c} \rangle \leftarrow \mathsf{SpdzRep}(\boldsymbol{c}_1, \ldots, \boldsymbol{c}_n, \alpha, \boldsymbol{\Delta}_{\gamma,\boldsymbol{c}}, m).$$

(6) Return $(\langle \boldsymbol{a} \rangle, \langle \boldsymbol{b} \rangle, \langle \boldsymbol{c} \rangle)$.

Fig. 8. $\mathcal{F}_{\mathsf{Offline}}^{\text{FAULTY}}$: Functionality that generates potentially faulty triples.

The main difference between $\mathcal{F}_{\mathsf{Offline}}^{\text{FAULTY}}$ and $\mathcal{F}_{\mathsf{Offline}}$ is that the former functionality may potentially output *faulty data*, which then means that the computation in the online phase will not yield a correct result. We want to stress that the original SPDZ protocol does not output potential codewords, but it can easily be modified to do so by adjusting the zero-knowledge proofs.

3.2 The preprocessing protocol

Let C be some $[m, k, d]$ Reed-Solomon code and C^* be its $[m, k', d']$ Schur transform. We assume the existence of a functionality that samples *faulty correlated randomness*, as depicted in Fig. 8. It generates random codewords as the shares of factors $\boldsymbol{a}, \boldsymbol{b}$ of multiplication triples and also enforces that malicious parties choose such codewords as their shares. The functionality then computes a product and shares it among all parties, subject to the constraint that $\mathtt{A}$ can arbitrarily modify the sum and the shares of malicious parties. Fig. 8 can be implemented using a SHE scheme as was shown in [19]. As a twist, the zero-knowledge proofs must be slightly extended to show that the vectors inside the ciphertexts contain codewords from C. Based on this available functionality, we show that one can implement $\mathcal{F}_{\mathsf{Offline}}$ as depicted in Fig. 5 using our protocol Π_{Offline}. $\mathcal{F}_{\mathsf{Offline}}$ is similar to $\mathcal{F}_{\mathsf{Offline}}^{\text{FAULTY}}$ but additionally ensures that all multiplication triples are correct.

The main idea of this protocol follows the outline as presented in the introduction:

(1) Check that the output vector $\boldsymbol{c}$ is a codeword of C^*. If so, then the error vector $\boldsymbol{\delta}$ is also a codeword, meaning that either it is $\mathbf{0}$ or *it has weight at least* d'.
(2) Open a fraction of the triples to check whether they are indeed correct. If so, then $\boldsymbol{\delta}$ must be the all-zero vector with high probability.

3.3 Fast and amortized checks

In the protocol presented in Fig. 10, we check each potential code vector separately. Let $\boldsymbol{H} \in \mathbb{F}^{l \times m}$ be the check matrix of the Schur transform of the code. Multiplication with a check matrix $\boldsymbol{H}$ can be done in $O(m^2)$ steps. If this must be carried out for a number of e.g. m vectors the cost is $O(m^3)$ operations, when done trivially. Consider all the l input vectors $\boldsymbol{a_1}, \ldots, \boldsymbol{a_l}$ concatenated as a matrix $\boldsymbol{A} = (\boldsymbol{a_1}|\boldsymbol{a_2}|\ldots|\boldsymbol{a_l})$. If all vectors are drawn from the code, then $\boldsymbol{HA} = \boldsymbol{0}$.

Procedure MatrixMultiplicationCheck

Input the matrices $\boldsymbol{H}, \boldsymbol{A}$.

(1) Compute the matrices $\boldsymbol{GH}$ and $\boldsymbol{AG}^\top$.
(2) For $j \in [m']$ select a pair $(x_j, y_j) \xleftarrow{\$} \{1, \ldots, m'\}^2$.
(3) For $j \in [m']$, compute z_j as the inner product of the x_jth row of $\boldsymbol{GH}$ and the y_jth column of $\boldsymbol{AG}^\top$.
(4) If all z_i are 0 return *accept*, otherwise *reject*.

Fig. 9. MatrixMultiplicationCheck: Procedure to check whether a matrix product is zero.

Now consider another generator matrix $\boldsymbol{G} \in \mathbb{F}^{m' \times l}$ for a Reed-Solomon code of message dimension l, where we denote the redundancy as $d \in O(m)$ again (we can easily assume that $m' \in O(m)$). Multiplication of each of the matrices $\boldsymbol{H}, \boldsymbol{A}$ with $\boldsymbol{G}$ can be done in time $m'^2 \cdot \log(m')$ using the FFT, and one can precompute $\boldsymbol{GH}$ before the actual computation takes place. $\boldsymbol{GHAG}^\top$ is a zero matrix if $\boldsymbol{A}$ only consists of codewords. On the other hand, consider $\boldsymbol{GHA}$: if one row is not a codeword, then it will be encoded to a vector with weight at least d due to the distance of the code. Multiplying with $\boldsymbol{G}^\top$ will then yield a matrix where at least d^2 entries are non-zero. Since both $m', d \in O(m)$, the fraction $\frac{d^2}{m'^2}$ is constant. One can compute both $\boldsymbol{GH}$ and $\boldsymbol{AG}^\top$ in time $m'^2 \cdot \log(m')$ using the FFT, and then choose rows/columns from both product matrices for which one then computes the inner product. In case that at least one $\boldsymbol{a}_i$ is not a codeword, the computed inner product will be non-zero with constant probability. Repeating this experiment $\Omega(m')$ times yields 0 in all cases only with probability negligible in m' if $\mathtt{A}$ cheated. We refer to [18] for more details on this technique.

4 Security of the Preprocessing from Somewhat Homomorphic Encryption

To prove the security of our construction from the previous section, we use the following fact:

Lemma 2. *Let C be an $[m, k, d]$ code and $\boldsymbol{x} \in C \setminus \{\boldsymbol{0}\}$. Choose elements t times from $[m]$ uniformly at random - denote this choice as the set $S \subseteq [m]$. Define $\mathcal{E}_{\boldsymbol{x}}$ to be the event that $\boldsymbol{x}[S] = \boldsymbol{0}$. Then*

$$\Pr[\mathcal{E}_{\boldsymbol{x}}] < \left(\frac{m-d}{m}\right)^t.$$

The statement follows because the codeword must be non-zero for at least d positions, hence the probability that we hit one of the (at least) d non-zero elements is $\frac{m-d}{m}$ for each experiment. By repeating this experiment t times independently one obtains the bound.

Theorem 1. *Let C be a $[m, k, d]$ linear block code, such that its Schur transform forms a $[m, k', d']$ linear block code C^*. Moreover, let $t = O(\kappa), t < k - 1$ and $d' = O(m)$ where $d' < m$. Then there exists a simulator $\mathcal{S}_{\mathsf{Offline}}$ such that $\mathcal{S}_{\mathsf{Offline}} \diamond \mathcal{F}_{\mathsf{Offline}}$ is statistically indistinguishable from Π_{Offline} in the $\mathcal{F}_{\mathsf{Offline}}^{\text{FAULTY}}, \mathcal{F}_{\mathsf{Commit}}, \mathcal{F}_{\mathsf{Rand}}$-hybrid model with broadcast channel where, for n players, up to $n - 1$ can act maliciously.*

For the sake of simplicity, we present our protocol for the case where running one instance of Π_{MacCheck} is sufficient (and where we hence only need one MAC). Both the $\langle\cdot\rangle$-representation and Π_{MacCheck} easily extend to the case where there are multiple MAC keys, and so does our protocol.

Protocol Π_{Offline}

Let $\boldsymbol{H}$ be the check matrix of C^* and $t \in \mathbb{N}^+, t < k-1$ be the upper bound on the number of opened triples. We assume that both C, C^* are in systematic form.

Initialize:

(1) All parties send $(\mathsf{Init}, \mathbb{F}, C)$ to $\mathcal{F}_{\mathsf{Offline}}^{\text{FAULTY}}$ to receive their shares α_i of α.

Triples:

(1) All parties send (Triple) to $\mathcal{F}_{\mathsf{Offline}}^{\text{FAULTY}}$ and obtain $(\langle \boldsymbol{a} \rangle, \langle \boldsymbol{b} \rangle, \langle \boldsymbol{c} \rangle)$.
(2) Let $\boldsymbol{c}_i$ be $\mathtt{P}_i$'s share of $\langle \boldsymbol{c} \rangle$. Each party locally computes $\boldsymbol{\sigma}_i = \boldsymbol{H}\boldsymbol{c}_i$ and commits to $\boldsymbol{\sigma}_i$ using $\mathcal{F}_{\mathsf{Commit}}$.
(3) Each party $\mathtt{P}_i$ opens its commitments to $\boldsymbol{\sigma}_i$ towards all parties. Check if $\mathbf{0} = \sum_i \boldsymbol{\sigma}_i$. If not, abort.
(4) Let $A = [m]$. The parties do the following t times in a loop:
  (4.1) Sample the random value $r \xleftarrow{\$} \mathcal{F}_{\mathsf{Rand}}(m)$. Set $A = A \setminus \{r\}$.
  (4.2) Each party $\mathtt{P}_i$ commits to its shares $\boldsymbol{a}_i[r], \boldsymbol{b}_i[r], \boldsymbol{c}_i[r]$ using $\mathcal{F}_{\mathsf{Commit}}$.
  (4.3) Each party opens its commitments towards all other parties.
  (4.4) Each party checks that $(\sum_i \boldsymbol{a}_i[r]) \cdot (\sum_i \boldsymbol{b}_i[r]) = \sum_i \boldsymbol{c}_i[r]$. If not, then they abort.
(5) Let $U = [m] \setminus A$, where $U = \{u_1, \ldots, u_l\}$. Compute

$$d \leftarrow \Pi_{\mathsf{MacCheck}}(\boldsymbol{\sigma}, \boldsymbol{a}[u_1], \boldsymbol{b}[u_1], \boldsymbol{c}[u_1], \ldots, \boldsymbol{a}[u_l], \boldsymbol{b}[u_l], \boldsymbol{c}[u_l]).$$

  If $d \neq 0$ then abort.
(6) Let $O \subset A$ be the smallest $k-t-1$ indices of A. The parties output $(\langle \boldsymbol{a}[O] \rangle, \langle \boldsymbol{b}[O] \rangle, \langle \boldsymbol{c}[O] \rangle)$.

Fig. 10. Π_{Offline}: Protocol that checks the correctness of triples.

Simulator $\mathcal{S}_{\mathsf{Offline}}$

SimulateInitialize:

(1) We start our own instance Π of the protocol Π_{Offline} which $\mathtt{A}$ is communicating with, and also local instances of $\mathcal{F}_{\mathsf{Commit}}$ and $\mathcal{F}_{\mathsf{Rand}}$.
(2) Send $(\mathsf{Init}, \mathbb{F}, C)$ on behalf of the simulated honest parties to $\mathcal{F}_{\mathsf{Offline}}^{\text{FAULTY}}$ and obtain the shares α_j. Since we simulate $\mathcal{F}_{\mathsf{Offline}}^{\text{FAULTY}}$, save the α_j for $\mathtt{P}_j \in \widehat{\mathsf{P}}$.
(3) Send $(\mathsf{Init}, \mathbb{F})$ in the name of the dishonest parties to $\mathcal{F}_{\mathsf{Offline}}$.
(4) For each dishonest party $\mathtt{P}_i$, send the intercepted share α_i to $\mathcal{F}_{\mathsf{Offline}}$.

SimulateTriples:

(1) Set the flag $\mathsf{cheated} = \bot$.
(2) Send (Triple) to $\mathcal{F}_{\mathsf{Offline}}^{\text{FAULTY}}$ in the name of the honest players $\mathtt{P}_j$.
(3) Intercept $\boldsymbol{a}_i, \boldsymbol{b}_i \in \mathbb{F}^m$ from each dishonest party $\mathtt{P}_i$. If either of these vectors is not in C, then stop here.
(4) Wait for $\mathtt{A}$ to input $\boldsymbol{\delta}, \boldsymbol{\Delta}_{\gamma,\boldsymbol{a}}, \boldsymbol{\Delta}_{\gamma,\boldsymbol{b}}, \boldsymbol{\Delta}_{\gamma,\boldsymbol{c}} \in \mathbb{F}^m$ and $\boldsymbol{c}_i \in \mathbb{F}^m$ for each dishonest $\mathtt{P}_i$.
(5) Wait for the output $\boldsymbol{a}_j, \boldsymbol{b}_j, \boldsymbol{c}_j, \boldsymbol{\gamma}(\boldsymbol{a})_j, \boldsymbol{\gamma}(\boldsymbol{b})_j, \boldsymbol{\gamma}(\boldsymbol{c})_j \in \mathbb{F}^m$ that the honest parties $\mathtt{P}_j$ obtain in Π from $\mathcal{F}_{\mathsf{Offline}}^{\text{FAULTY}}$.
(6) Commit to the correct value $\boldsymbol{\sigma}_j = \boldsymbol{H}\boldsymbol{c}_j$ for each honest $\mathtt{P}_j$ in Step (2).
(7) Let $\{\boldsymbol{\sigma_i}\}_{i \in \widehat{\mathsf{P}}}$ be the commitments $\mathtt{A}$ sent to $\mathcal{F}_{\mathsf{Commit}}$. If $\sum_{i \in \widehat{\mathsf{P}}} \boldsymbol{\sigma}_i \neq \boldsymbol{H}(\sum_{i \in \widehat{\mathsf{P}}} \boldsymbol{c}_i)$ then set $\mathsf{cheated} = \top$.
(8) Do Step (3) as in the protocol. Abort if the protocol aborts.
(9) Do Step (4) with its t iterations. In every iteration, if the sum of the values $\boldsymbol{a}_i, \boldsymbol{b}_i, \boldsymbol{c}_i$ that the dishonest parties send differs from those they sent to $\mathcal{F}_{\mathsf{Offline}}^{\text{FAULTY}}$, then set $\mathsf{cheated} = \top$.
(10) Do Step (5) as in the protocol. Abort if Π aborts, if $\boldsymbol{\delta} \neq \mathbf{0}$, if a dishonest party sends a value inconsistent with those it has or if $\mathsf{cheated} = \top$.
(11) Send $(\mathsf{Triple}, k-t-1)$ to $\mathcal{F}_{\mathsf{Offline}}$ in the name of all dishonest parties.
(12) Compute the set O as in Π. Send the adversarially chosen $\boldsymbol{a}_i[O], \boldsymbol{b}_i[O], \boldsymbol{c}_i[O]$ for each dishonest $\mathtt{P}_i$ to $\mathcal{F}_{\mathsf{Offline}}$.
(13) For every call of $\mathsf{SpdzRep}$, send the same values to $\mathcal{F}_{\mathsf{Offline}}$ that $\mathtt{P}_i$ sent to $\mathcal{F}_{\mathsf{Offline}}^{\text{FAULTY}}$ for the same call.

Fig. 11. $\mathcal{S}_{\mathsf{Offline}}$: Simulator for the protocol Π_{Offline}.

Proof. In order to prove the theorem, we use the simulator as described in Fig. 11. The simulator will run a local copy of the protocol, together with local copies of $\mathcal{F}_{\mathsf{Offline}}^{\text{FAULTY}}$, $\mathcal{F}_{\mathsf{Commit}}, \mathcal{F}_{\mathsf{Rand}}$

Correctness. We first observe that the protocol Π_{Offline} is correct in the sense that, if all parties follow it without deviations, they obtain outputs as in $\mathcal{F}_{\mathsf{Offline}}$. This trivially follows from the fact that $\mathcal{F}_{\mathsf{Offline}}^{\text{FAULTY}}$ and $\mathcal{F}_{\mathsf{Offline}}$ only deviate in $\boldsymbol{\delta}$, which would never be set if all parties behave honestly. The introduced redundancy due to the code C is fully discarded, hence the output of the protocol will consist of correct and randomly distributed multiplication triples. Since the MAC shares of the dishonest parties are the same in $\mathcal{F}_{\mathsf{Offline}}^{\text{FAULTY}}$ and $\mathcal{F}_{\mathsf{Offline}}$ they are consistent with the honest parties' outputs.

Simulatability. Neither in $\mathcal{F}_{\mathsf{Offline}}^{\text{FAULTY}}$ nor in $\mathcal{F}_{\mathsf{Offline}}$ the dishonest parties will ever receive output. Moreover, in the simulation we follow exactly the protocol for the simulated honest parties and stop whenever the protocol would stop. In addition, we also abort if the protocol would not abort, but the adversary provided a value $\boldsymbol{\delta} \neq \mathbf{0}$ or if the dishonest parties sent inconsistent values. We now argue that the probability that the protocol does not abort while the simulator does abort is negligible.

In the case of $\mathsf{cheated} = \top$ or if the dishonest parties send inconsistent values in Π_{MacCheck}, this follows from Lemma 1. Therefore, let us assume that $\mathsf{cheated} = \bot$, but $\boldsymbol{\delta} \neq \mathbf{0}$. Observe that

$$\Pr[\Pi \text{ aborts} \,|\, \boldsymbol{\delta} \neq \mathbf{0}] \leq \Pr[\mathcal{S}_{\mathsf{Offline}} \text{ aborts} \,|\, \boldsymbol{\delta} \neq \mathbf{0}].$$

$\mathcal{S}_{\mathsf{Offline}}$ is constructed such that $\Pr[\mathcal{S}_{\mathsf{Offline}} \text{ aborts} \,|\, \boldsymbol{\delta} \neq \mathbf{0}] = 1$. On the other hand, we can either catch the adversary if $\boldsymbol{\delta} \notin C^*$ or if it is indeed a codeword, but not the correct one:

$$\begin{aligned}\Pr[\Pi \text{ aborts} \,|\, \boldsymbol{\delta} \neq \mathbf{0}] = \Pr[\Pi \text{ aborts} \,|\, \boldsymbol{\delta} \notin C^*] \cdot \frac{\Pr[\boldsymbol{\delta} \notin C^*]}{\Pr[\boldsymbol{\delta} \neq \mathbf{0}]} \\ + \Pr[\Pi \text{ aborts} \,|\, (\boldsymbol{\delta} \in C^* \wedge \boldsymbol{\delta} \neq 0)] \cdot \frac{\Pr[\boldsymbol{\delta} \in C^* \wedge \boldsymbol{\delta} \neq 0]}{\Pr[\boldsymbol{\delta} \neq \mathbf{0}]}.\end{aligned}$$

Since $\mathsf{cheated} = \bot$ both Step (2) and Step (4) are carried out correctly except with negligible probability i.e. the dishonest parties provided correct values.

If $\boldsymbol{\delta} \notin C^*$, then $\sum_{i=1}^{n} \boldsymbol{\sigma}_i \neq \mathbf{0}$ due to the fact that $\boldsymbol{H}$ is a check matrix. Hence we have that

$$\Pr[\Pi \text{ aborts} \,|\, \boldsymbol{\delta} \notin C^*] = 1 - \mathsf{negl}(\kappa).$$

On the other hand, we can use Lemma 2 to give a lower bound on the second term, which is

$$\Pr[\Pi \text{ aborts} \,|\, (\boldsymbol{\delta} \in C^* \wedge \boldsymbol{\delta} \neq 0)] > 1 - \left(\frac{m - d'}{m}\right)^t - \mathsf{negl}(\kappa).$$

Letting t, d' be chosen as mentioned in the theorem, the statement follows. □

A reformulation of Lemma 2

Here we give an alternative to Lemma 2. It performs slightly worse when using actual numbers, but is more natural as we always check a *constant fraction* of the positions whereas Lemma 2 uses t values sampled uniformly at random from $[m]$.

Lemma 3. *Let C be an $[m, k, d]$ code and $\boldsymbol{x} \in C \setminus \{\mathbf{0}\}$. Moreover, let $S \subset [m]$ be chosen uniformly at random such that $|S| = t$ and define $\mathcal{E}_{\boldsymbol{x}}$ to be the event that $\boldsymbol{x}[S] = \mathbf{0}$. Then*

$$\Pr[\mathcal{E}_{\boldsymbol{x}}] < \left(\frac{m - d}{m - t}\right)^t.$$

Moreover, for practical applications one should use the sharper bound

$$\Pr[\mathcal{E}_{\boldsymbol{x}}] < \left(\frac{2m - 2d - t + 1}{2m - 2t + 2}\right)^t.$$

Let us first define the function

$$\begin{aligned} \# : C &\to \mathbb{N} \\ \boldsymbol{x} &\mapsto \sum_{i=1}^{m} (1 - \delta_{\boldsymbol{x}[i]0}), \end{aligned}$$

where δ_{ij} is the *Kronecker Delta function*. This function computes the *Hamming weight* of the vector $\boldsymbol{x}$.

Proof. For an arbitrary $\boldsymbol{x}$, the event $\mathcal{E}_{\boldsymbol{x}}$ occurs with probability

$$\begin{aligned} \Pr[\mathcal{E}_{\boldsymbol{x}}] &= \frac{m - \#(x)}{m} \cdot \frac{m - \#(x) - 1}{m - 1} \cdots \frac{m - \#(x) - t + 1}{m - t + 1} \\ &= \prod_{i=0}^{t-1} \frac{m - \#(x) - i}{m - i}. \end{aligned}$$

Now the code has a minimal distance d, hence $\#(x) \geq d$. We can therefore give an upper bound on $\Pr[\mathcal{E}_{\boldsymbol{x}}]$ as

$$\begin{aligned} \Pr[\mathcal{E}_{\boldsymbol{x}}] &\leq \prod_{i=0}^{t-1} \frac{m - d - i}{m - i} \\ &< \frac{(m-d)^t}{(m-t+1)^t} \\ &< \left(\frac{m-d}{m-t}\right)^t, \end{aligned} \tag{1}$$

which proves the first claim. To give a somewhat tighter bound, observe that we can look at the product in Equation (1) as a geometric mean. We can find an upper bound using the arithmetic mean as

$$\prod_{i=0}^{t-1} \frac{m - d - i}{m - i} \leq \left(\frac{1}{t} \sum_{i=0}^{t-1} \frac{m - d - i}{m - i}\right)^t. \tag{2}$$

Now observe that the sum in Equation (2) can be upper-bounded again as

$$\begin{aligned} \sum_{i=0}^{t-1} \frac{m - d - i}{m - i} &= \frac{\sum_{j=0}^{t-1} \left((m - d - j) \left(\prod_{i=0, i\neq j}^{t-1} (m - i)\right)\right)}{\prod_{i=0}^{t-1} (m - i)} \\ &< \frac{\sum_{j=0}^{t-1} \left((m - d - j) \left(\prod_{i=0}^{t-2} (m - i)\right)\right)}{\prod_{i=0}^{t-1} (m - i)} \\ &= \frac{\left(\prod_{i=0}^{t-2} (m - i)\right) \sum_{j=0}^{t-1} (m - d - j)}{\prod_{i=0}^{t-1} (m - i)} \\ &= \frac{\sum_{i=0}^{t-1} (m - d - i)}{m - t + 1} \\ &= \frac{2mt - 2dt - t^2 + t}{2m - 2t + 2}. \end{aligned}$$

Putting things together, this yields

$$\Pr[\mathcal{E}_{\boldsymbol{x}}] < \left(\frac{2m - 2d - t + 1}{2m - 2t + 2}\right)^t.$$

□

5 Amortized Zero-Knowledge proofs for Lattice-based Encryption

In the following, we present an amortized ZKPoPK which extends the idea of [16]. The proven upper bounds on the output ciphertexts will be worse than [16], but only by a factor t. On the other hand, we will be able to reduce the number of auxiliary ciphertexts by a factor[9] of ≈ 2.

The overall strategy of the proof is as follows:

(1) Sample a number of *auxiliary* ciphertexts $2T$, then open a (random) half in a cut-and-choose process.
(2) For b_1 rounds, open the sum of a ciphertext $\boldsymbol{c}_i$ and a random auxiliary ciphertext.
(3) Then for b_2 rounds, randomly put the t ciphertexts into a matrix of width and height $\sqrt{t}$. Compute all row and column sums and prove plaintext knowledge of them.

The rationale behind this strategy is that, after the cut-and-choose, most of the T auxiliary ciphertexts are generated correctly (except with small probability). Hence most of the sums that are opened in the second step allow extraction and only a small number of the t ciphertexts could not be extracted after b_1 rounds. The third step works best for a very small number of remaining bad ciphertexts (up to around $\sqrt{t}/2$), but then allows extraction in a small number of repetitions. The proof can be found in Fig. 12 and Fig. 13

Protocol Π_{NewZK} (Part 1)

The prover inputs $\boldsymbol{x}_1, \boldsymbol{r}_1, \ldots, \boldsymbol{x}_t, \boldsymbol{r}_t$ and proves R^H_{PoPK} for given $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_t$. Let $b_1, b_2, c \in \mathbb{N}$. For the sake of simplicity, we assume that $\sqrt{t} \in \mathbb{N}$. Honestly generated $\boldsymbol{c}_i$ will have $||\boldsymbol{x}_i||_\infty \leq \tau$ and $||\boldsymbol{r}_i||_\infty \leq \rho$. Let $v = (W+d) \cdot \delta \cdot T \cdot \sqrt{t}$.

(1) Let $w \leftarrow 1$ be a counter. Set $T = t \cdot b_1 + c \cdot b_2 \cdot 2 \cdot \sqrt{t}$.
(2) $(\boldsymbol{a}_1, \boldsymbol{y}_1, \boldsymbol{s}_1, \ldots, \boldsymbol{a}_T, \boldsymbol{y}_T, \boldsymbol{s}_T) \leftarrow \mathsf{CutAndChoose}(T, v \cdot \tau, v \cdot \rho)$.
(3) $\mathtt{Pr}, \mathtt{Ve}$ jointly sample a uniformly random partitions $V_1, \ldots, V_{b_1+b_2} \subset [T]$ using $\mathcal{F}_{\mathsf{Rand}}$, such that

$$|V_i| = \begin{cases} t & \text{if } i \in \{1, \ldots, b_1\} \\ 2 \cdot c \cdot \sqrt{t} & \text{if } i \in \{b_1+1, \ldots, b_1+b_2\} \end{cases}$$

(4) For $z \in [b_1]$ do the following:
  (4.1) Rename $V_z = \{v_1, \ldots, v_t\}$. $\mathtt{Pr}$ computes $\boldsymbol{\alpha}_i = \boldsymbol{x}_i + \boldsymbol{y}_{v_i}$, $\boldsymbol{\beta}_i = \boldsymbol{r}_i + \boldsymbol{s}_{v_i}$.
  (4.2) $\mathtt{Pr}$ checks that $||\boldsymbol{\alpha}_i||_\infty \leq (v-1) \cdot \tau$ and $||\boldsymbol{\beta}_i||_\infty \leq (v-1) \cdot \rho$. If not, then $\mathtt{Pr}$ sends $\perp$ and starts again.
  (4.3) $\mathtt{Pr}$ sends $(\boldsymbol{\alpha}_i, \boldsymbol{\beta}_i)_{i \in [t]}$ to $\mathtt{Ve}$.
  (4.4) $\mathtt{Ve}$ checks that $\mathsf{Enc}_{\mathsf{pk}}(\boldsymbol{\alpha}_i, \boldsymbol{\beta}_i) = \boldsymbol{c}_i \boxplus \boldsymbol{a}_{v_i}$ and $||\boldsymbol{\alpha}_i||_\infty, ||\boldsymbol{\beta}_i||_\infty$ follow the above bounds.
  (4.5) If a check does not hold: if $w < M$, increment w by 1 and go to Step (2). If $w = M$ then $\mathtt{Ve}$ *rejects*.

Fig. 12. Π_{NewZK}: Asymptotically efficient ZKPoPK.

5.1 The problem of soundness

Whereas the first two steps of our proof are mathematically simple to analyze, the third step seems somewhat more cumbersome. In Fig. 14, we give a graphical description of how the extractor in the soundness argument should work if one abstracts the problem as an instance of a *bins and balls*-game.

In a more formal way, we can describe Fig. 14 as an algorithm:

$\mathsf{matrixGame}(\boldsymbol{M}, m, n)$:
  (1) If $\boldsymbol{M} \in \mathbb{Z}_2^{m \times n}$ then continue, otherwise abort.

[9] This is particularly critical because in implementations, one would try to keep all ciphertexts in the RAM (which was a particular problem in [16]).

Protocol Π_{NewZK} (Part 2)

(5) $\mathtt{Pr}, \mathtt{Ve}$ together sample b_2 permutations $W_1, \ldots, W_{b_2}$ of $[t]$ using $\mathcal{F}_{\mathsf{Rand}}$. For $z \in [b_2]$ do the following:
  (5.1) Let $u = 2 \cdot \sqrt{t}$ and $W_z = \{w_1, \ldots, w_t\}$. For $i \in [\sqrt{t}]$ we define

$$\boldsymbol{x}'_i = \sum_{k=1}^{\sqrt{t}} \boldsymbol{x}_{w_{(i-1)\sqrt{t}+k}} \text{ and } \boldsymbol{x}'_{i+\sqrt{t}} = \sum_{k=1}^{\sqrt{t}} \boldsymbol{x}_{w_{(k-1)\sqrt{t}+i}}$$

  and $\boldsymbol{r}'_i$, $\boldsymbol{c}'_i$ accordingly.
  (5.2) For $i \in [u]$, $k \in [c]$ and $V_{b_1+z} = \{v_1, \ldots, v_{u \cdot c}\}$ $\mathtt{Pr}$ computes $\boldsymbol{\alpha}_i^k = \boldsymbol{x}'_i + \boldsymbol{y}_{v_{(i-1)\sqrt{t}+k}}$ and similarly $\boldsymbol{\beta}_i^k = \boldsymbol{r}'_i + \boldsymbol{s}_{v_{(i-1)\sqrt{t}+k}}$.
  (5.3) $\mathtt{Pr}$ checks that $||\boldsymbol{\alpha}_i^k||_\infty \leq (v - \sqrt{t}) \cdot \tau$ and $||\boldsymbol{\beta}_i^k||_\infty \leq (v - \sqrt{t}) \cdot \rho$. If not, then he sends $\perp$ to $\mathtt{Ve}$ and starts again.
  (5.4) $\mathtt{Pr}$ sends $(\boldsymbol{\alpha}_i^k, \boldsymbol{\beta}_i^k)$ to $\mathtt{Ve}$.
  (5.5) $\mathtt{Ve}$ checks that $\mathsf{Enc}_{\mathsf{pk}}(\boldsymbol{\alpha}_i^k, \boldsymbol{\beta}_i^k) = \boldsymbol{c}'_i \boxplus \boldsymbol{a}_{v_{(i-1)\sqrt{t}+k}}$ and if $||\boldsymbol{\alpha}_i^k||_\infty, ||\boldsymbol{\beta}_i^k||_\infty$ have the bounds from the previous step.
  (5.6) If a check does not hold: if $w < M$, increment w by 1 and go to Step (2). If $w = M$ then $\mathtt{Ve}$ *rejects*.
(6) $\mathtt{Ve}$ accepts.

Fig. 13. Π_{NewZK}: Asymptotically efficient ZKPoPK, continued.

(2) Let r be the number of ones that are alone in a column of $\boldsymbol{M}$. Remove all such r ones to form the matrix $\boldsymbol{M}'$.
(3) Let s be the number of ones that are alone in a row of $\boldsymbol{M}'$.
(4) Output (r, s).

This one can rephrase into an expression that will turn out helpful in the security proof.

Problem 1 *Let $m, n, k \in \mathbb{N}^+, k \leq m \cdot n, c \in \mathbb{R}^+$ and*

$$\mathcal{M}_{m,n,k} = \{\boldsymbol{A} \in \mathbb{Z}_2^{m \times n} \mid \boldsymbol{A} \text{ has exactly } k \text{ ones}\}$$

Compute

$$\Pr\left[r + s \geq \lfloor k/c \rfloor \;\middle|\; \boldsymbol{M} \xleftarrow{\$} \mathcal{M}_{m,n,k} \wedge (r, s) \leftarrow \mathsf{matrixGame}(\boldsymbol{M}, m, n)\right]$$

We do not have a formula in closed form to compute the probabilities in the above problem. But a brute-force approach does also not seem plausible: we observe that

$$|\mathcal{M}_{m,n,k}| = \binom{m \cdot n}{k}$$

is the total number of ways of arranging k balls on an $m \times n$ grid. Sampling all such matrices and computing the output of $\mathsf{matrixGame}$ for each of them for larger values of m, n, k is computationally intractable. In Appendix A, we describe how to solve the problem efficiently for $k < 40$ which is sufficient for our application. Based on this, Table 2 contains some parameter recommendations for the protocol based on our analysis.

We want to point out that we use $\mathsf{matrixGame}$ in two different ways in our analysis: if a plaintext is extracted in one round of an instance of Problem 1, then we call such a ciphertext *strongly extracted*. If one only extracts ciphertexts that, already in $\boldsymbol{M}$ from the problem are either alone in a row or column (that is, we do not consider rows that, only after Step (2) of $\mathsf{matrixGame}$, have exactly one element), then we call such ciphertexts *weakly extracted*. Using strong extraction allows to lower the number of rounds and auxiliary ciphertexts in the protocol, but increases the bound on the size of the extracted ciphertexts by a factor $\sqrt{t}$.

Theorem 2. *Let $H = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ be a somewhat homomorphic cryptosystem with $\delta > 1, T > \kappa$ and $v = (W + d) \cdot \delta \cdot T \cdot \sqrt{t}$, then the protocol Π_{NewZK} is a honest-verifier zero-knowledge proof of plaintext knowledge for the relation R^H_{PoPK} where*

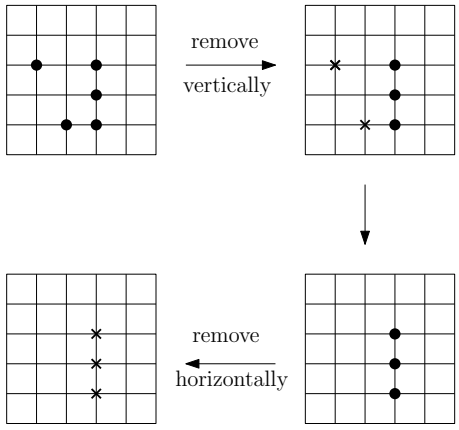

Fig. 14. Two-dimensional elimination process (strong extraction)

κ	t	b_1	b_2	c	strong extraction?	T/t	T/t in [16]
40	32	3	1	5	no	9.75	16
40	64	3	1	4	no	8	14
40	64	2	1	6	yes	7	14
40	100	2	1	6	yes	6.4	14
40	256	2	2	4	yes	6	12

Table 2. Example parameter sets for the proof Π_{NewZK}

$B_P = 2 \cdot v \cdot \tau \cdot \sqrt{t}$, $B_R = 2 \cdot v \cdot \rho \cdot \sqrt{t}$ *for weakly extracted ciphertexts.*
$B_P = 2 \cdot v \cdot \tau \cdot t$, $B_R = 2 \cdot v \cdot \rho \cdot t$ *for strongly extracted ciphertexts.*

Before proving the above theorem, we want to make a few remarks. First of all, the first of the b_1 rounds of Step (4) can be avoided if one is ok with random plaintexts. In such a case, one would sample t plaintext/ciphertext pairs using $\mathsf{CutAndChoose}$ instead. A side effect of this proof technique is that by setting t very large and $b_1 = 0$ one can end up in a situation where $T < t$ i.e. one performs less individual proofs than there are ciphertexts. Unfortunately, this behavior only occurs for very large values of t and is therefore of no practical relevance. We also want to mention that Π_{NewZK} can be used in the implementation of $\mathcal{F}_{\mathsf{Offline}}^{\text{FAULTY}}$. To achieve this, one must sample all the plaintexts of the (auxiliary) ciphertexts as codewords and check that all opened sums and all opened plaintexts from the cut-and-choose phase are from the code C. Since all steps in the soundness argument that are used to extract plaintexts only perform linear operations, this then works out of the box.

Proof (of Theorem 2). This proof is split in two halves: as mentioned above, the soundness of the proof relies on the solution of Problem 1 which is discussed in more detail in Appendix A.

Completeness. Assume that $\mathtt{Pr}$ follows the protocol correctly and hence $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_t$ were generated according to the correct distributions. Due to the linearity of $\mathsf{Enc}_{\mathsf{pk}}(\cdot,\cdot)$, the protocol only aborts if one of the coefficients of $\boldsymbol{\alpha}_i^j, \boldsymbol{\beta}_i^j$ becomes too large and therefore leaks information (which is when it is restarted). Let us focus on the case where this happens: the probability that a coefficient of any of the $\boldsymbol{\alpha}_i^j, \boldsymbol{\beta}_i^j$ in both Step (4) or Step (5) lies outside of the correct bounds is at most $1/((W+d) \cdot \delta \cdot T)$. There are $W + d$ such coefficients and T sums that are computed in total, hence the probability that $\mathtt{Pr}$ must send $\perp$ is $1/\delta$. For M such rounds, the chance that $\mathtt{Pr}$ fails to prove correct values is $(1/\delta)^M$.

Honest-Verifier Zero-Knowledge. Consider the following algorithm:

(1) Let $w \leftarrow 1$ be a counter.
(2) Choose the subset $V \subset [2T]$ uniformly at random of size T. Choose the $b_1 + b_2$ subsets $V_1, \ldots, V_{b_1+b_2} \subset [2T] \setminus V$ as in Step (3) in Π_{NewZK} at random according to their size constraints. Moreover, choose the b_2 random permutations $W_1, \ldots, W_{b_2}$ of $[t]$ as in Step (5) of the protocol.
(3) Compute all the $\boldsymbol{x}'_i, \boldsymbol{r}'_i, \boldsymbol{c}'_i$ as in Step (5.1) of Π_{NewZK}.
(4) For each of the T sums $\boldsymbol{\alpha}_i^j$, choose for $\boldsymbol{\alpha}_i^j$ each coefficient uniformly at random from the interval $[-v\cdot\tau, v\cdot\tau]$. Moreover, choose for each of the T sums $\boldsymbol{\beta}_i^j$ each coefficient uniformly at random from $[-v \cdot \rho, v \cdot \rho]$.
(5) For each of the $i \in [2T]$ from $\mathsf{CutAndChoose}$ we do the following:
   – If $i \in V$ then sample $\boldsymbol{y}_i, \boldsymbol{s}_i$ as in Step (1) of $\mathsf{CutAndChoose}$. Then compute $\boldsymbol{a}_i \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\boldsymbol{y}_i, \boldsymbol{s}_i)$.
   – If $i \notin V$ then set $\boldsymbol{a}_{v_i} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\boldsymbol{\alpha}_i^j, \boldsymbol{\beta}_i^j) \boxminus \boldsymbol{c}_i$ if $i \in \{V_1, \ldots, V_{b_1}\}$ or (with an obvious reindexing based on W_z, V_{b_1+z} for $z \in [b_2]$) compute $\boldsymbol{a}_i \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\boldsymbol{\alpha}_i^j, \boldsymbol{\beta}_i^j) \boxminus \boldsymbol{c}'_j$ if $i \in \{V_{b_1+1}, \ldots, V_{b_1+b_2}\}$.
(6) For $i \in V$ output $\boldsymbol{a}_i, \boldsymbol{y}_i, \boldsymbol{s}_i$ and its PRF seeds. For $i \notin V$ output $\boldsymbol{a}_i$.
(7) For each $j \in [b_1]$ if it holds that $\forall i \in V_j$ the plaintexts and randomness are small enough, i.e.

$$||\boldsymbol{\alpha}_i^j||_\infty \leq (v-1)\cdot\tau \text{ and } ||\boldsymbol{\beta}_i^j||_\infty \leq (v-1)\cdot\rho$$

   then output $\boldsymbol{\alpha}_i^j, \boldsymbol{\beta}_i^j$ else output $\perp$ and increase w by one. If $\perp$ was returned: stop if $w = M$, otherwise go to Step (2).
(8) For each $j \in \{b_1, \ldots, b_1 + b_2\}$ if it holds that $\forall i \in V_j$ the plaintexts and randomness are small enough, i.e.

$$||\boldsymbol{\alpha}_i^j||_\infty \leq (v-\sqrt{t})\cdot\tau \text{ and } ||\boldsymbol{\beta}_i^j||_\infty \leq (v-\sqrt{t})\cdot\rho$$

   then output $\boldsymbol{\alpha}_i^j, \boldsymbol{\beta}_i^j$ else output $\perp$ and increase w by one. If $\perp$ was returned: stop if $w = M$, otherwise go to Step (2).

The above algorithm outputs transcripts that are perfectly indistinguishable from transcripts generated by Π_{NewZK}. This can be seen as follows:

– All the V_i, W_i and V are chosen as in the protocol, and for all $i \in V$ the ciphertexts $\boldsymbol{a}_i$ are generated as in the protocol.
– For each honest choice of $\boldsymbol{x}_i$, by adding a uniform vector $\boldsymbol{y}_i$ from the given interval the probability that $\boldsymbol{x}_i + \boldsymbol{y}_i$ exceeds the bound is the same, which also holds for $\boldsymbol{r}_i$ and $\boldsymbol{s}_i$. Hence the probability of outputting $\perp$ is the same as in Π_{NewZK}.
– Let $i \in [2T] \setminus V$. For each $\boldsymbol{\alpha}_i^j, \boldsymbol{\beta}_i^j$ in Π_{NewZK}, if it is given as output to $\mathtt{Ve}$ by an honest $\mathtt{Pr}$ then it holds that $\forall \boldsymbol{x}_i \exists! \boldsymbol{y}_i : \boldsymbol{x}_i + \boldsymbol{y}_i = \boldsymbol{\alpha}_i^j$ and similarly for $\boldsymbol{x}'_i, \boldsymbol{r}_i, \boldsymbol{r}'_i, \boldsymbol{s}_i$. Therefore, the probability of outputting $\boldsymbol{\alpha}_i^j, \boldsymbol{\beta}_i^j$ in Π_{NewZK} is independent of $\boldsymbol{x}_i, \boldsymbol{r}_i$ and one can safely choose them in the simulation at random.

Soundness. To establish the bound on the plaintext and randomness size, we first observe that a plaintext can either be extracted in one of the b_1 iterations of Step (4) or in the b_2 rounds of Step (5).

Extraction in Step (4)*:* Assume that the sum of a ciphertext and an *extractable* auxiliary ciphertext $\boldsymbol{c}_j + \boldsymbol{a}_i$ was opened during an iteration of Step (4). Using $\boldsymbol{y}_i, \boldsymbol{s}_i$, for each such $\boldsymbol{c}_j$ one can extract the $\boldsymbol{x}_j, \boldsymbol{r}_j$ from $\boldsymbol{\alpha}_i^j, \boldsymbol{\beta}_i^j$. Since $||\boldsymbol{\alpha}_i^j||_\infty, ||\boldsymbol{y}_i||_\infty \leq v \cdot \tau$ it must hold that $||\boldsymbol{x}_j||_\infty \leq 2 \cdot v \cdot \tau$. By the same argument, the extracted randomness $\boldsymbol{r}_j$ has coefficients of size at most $2 \cdot v \cdot \rho$.

In order to obtain the $\boldsymbol{y}_i, \boldsymbol{s}_i$ that are necessary in the above reasoning we observe that each auxiliary ciphertext $\boldsymbol{a}_i$ was generated in Step (1) of $\mathsf{CutAndChoose}$. It is therefore computed from an expanded seed f_i which was fed into the random oracle. One can extract the RO queries that $\mathtt{Pr}$ makes, and in particular the f_i seeds (more precisely, one can extract $T - \kappa$ of them except with probability $2^{-\kappa}$).

Extraction in Step (5)*:* For the sake of simplicity, let $b_2 = 1$. In the case of *weak extraction*, each plaintext in question can be computed as a sum of an auxiliary ciphertext (of norm $v\cdot\tau$) and at most $\sqrt{t}-1$ plaintexts

from Step (4), which yields a total bound of $2 \cdot v \cdot \tau \cdot \sqrt{t}$. The bound on the randomness can be established the same way. For *strong extraction*, we first observe that the process now exactly follows $\mathsf{matrixGame}$ as defined above. The plaintexts extracted during Step (2) of $\mathsf{matrixGame}$ will once again have norm at most $2 \cdot v \cdot \tau \cdot \sqrt{t}$ by the same argument as before. In Step (3), it might happen that all the $\sqrt{t} - 1$ plaintexts that one uses in the extraction were only just computed in Step (2) of $\mathsf{matrixGame}$. Therefore, the bound on the plaintexts extracted during Step (3) can be as high as $2 \cdot v \cdot \tau \cdot t$ which proves the claim. Once again, the norm on the randomness follows accordingly. For arbitrary b_2, the above argument also holds because we then just extract all plaintexts at once in one of the b_2 rounds, given this is in fact possible.

It remains to show for which choice of parameters b_1, b_2, c the above procedure works except with negligible probability. This is a mere computational problem and will be answered in Appendix A.2. □

6 Preprocessing from Paillier Encryption

In this section we present a novel approach to produce multiplication triples using Paillier's cryptosystem. In comparison to previous work which uses heavy zero-knowledge machinery to prove that multiplications are done correctly, we choose a somewhat different approach that is related to the preprocessing protocol from the previous section. Moreover, we present two zero-knowledge proofs which are used in the protocol (they require to send fewer bits per proof instance in comparison to previous work) and a distributed decryption technique that differs from existing work.

6.1 Proving statements about Paillier ciphertexts

First, consider a regular proof of plaintext knowledge. For Paillier encryption, one would prove the following relation:

$$R^P_{\text{PoPK}} = \left\{(a, w) \;\middle|\; a = (c, \mathsf{pk}) \wedge w = (x, r) \wedge x \in \mathbb{Z}/N\mathbb{Z} \wedge r \in \mathbb{Z}/N\mathbb{Z}^* \wedge c = \mathsf{Enc}_{\mathsf{pk}}(x, r)\right\}$$

Throughout the protocol, the parties must compute products with ciphertexts, where we want to establish that a party *knows* which value it multiplied in. This can be captured as follows:

$$R_{\text{PoM}} = \left\{ (a, w) \middle| \begin{array}{l} a = (z, \hat{z}, \mathsf{pk}) \wedge w = (b, c, r) \wedge b, c \in \mathbb{Z}/N\mathbb{Z} \wedge \\ r \in \mathbb{Z}/N\mathbb{Z}^* \wedge \hat{z} = z^b \cdot \mathsf{Enc}_{\mathsf{pk}}(c, r) \bmod N^2 \end{array} \right\}$$

In the following, we present honest-verifier perfect zero-knowledge proofs for both R^P_{PoPK}, R_{PoM} between a prover $\mathtt{Pr}$ and verifier $\mathtt{Ve}$. In order to use them in the preprocessing protocol, one can either make them non-interactive using the Fiat-Shamir [20] transformation in the Random Oracle Model, or use a secure coin-flip protocol to sample the challenge e. Since during a protocol instance, many proofs are executed in parallel, one can use the same challenge for all instances and so the complexity of doing the coin-flip is not a significant cost. For practical implementations, one can choose the random value e from a smaller interval like e.g. $[0, 2^\kappa]$. This also yields negligible cheating probability[10].

Proof of knowledge

Lemma 4. *The protocol $\Pi^{\text{P}}_{\mathsf{PoPK}}$ is an an interactive honest-verifier proof for the relation R^P_{PoPK}.*

In the proof it is used that Enc is the bijection

$$\begin{aligned} \mathsf{Enc} : \mathbb{Z}/N\mathbb{Z}^* \times \mathbb{Z}/N\mathbb{Z} &\to \mathbb{Z}/N^2\mathbb{Z}^* \\ (x, r) &\mapsto r^N (N+1)^x. \end{aligned}$$

This follows directly from [34] and $ord_{\mathbb{Z}/N^2\mathbb{Z}^*}(N+1) = N$.

Proof.

[10] For the soundness of the proof, we rely on the fact that $(e - e', N) = 1$ which indeed is always true if $e, e' \ll \sqrt{N}$ and N is a safe RSA modulus.

Protocol $\Pi^{\mathrm{P}}_{\mathsf{PoPK}}$

(1) $\mathtt{Pr}$ samples $s \xleftarrow{\$} \mathbb{Z}/N\mathbb{Z}^*$ and sends $t = s^N \mod N$ to $\mathtt{Ve}$.
(2) $\mathtt{Ve}$ samples $e \xleftarrow{\$} \mathbb{Z}/N\mathbb{Z}$ and sends it to $\mathtt{Pr}$.
(3) $\mathtt{Pr}$ sends $k = s \cdot r^e \mod N$ to $\mathtt{Ve}$.
(4) $\mathtt{Ve}$ accepts if $k^N = c^e \cdot t \mod N$ and otherwise rejects.

Fig. 15. $\Pi^{\mathrm{P}}_{\mathsf{PoPK}}$: Protocol to prove knowledge of plaintexts of Paillier encryptions.

Correctness. Let $c = \mathsf{Enc}_{\mathsf{pk}}(x, r) \in \mathbb{Z}/N^2\mathbb{Z}^*$, and observe that $c = r^N \bmod N$. If both parties follow the protocol, then

$$\begin{aligned} k^N &= (s \cdot r^e)^N \bmod N \\ &= s^N \cdot (r^N)^e \bmod N \\ &= t \cdot c^e \bmod N, \end{aligned}$$

which proves correctness.

Soundness. Assume that $\mathtt{Ve}$ obtains two protocol transcripts $(t, e, k), (t, e', k')$ such that $e \neq e' \bmod N$ and $e > e'$. Then it additionally must hold that $k \neq k' \bmod N$: for the sake of contradiction, assume that $k = k'$, then $(k/k')^N = 1 = c^{e-e'} \bmod N$ where $c = r^N \bmod N$, so we can write $1 = c^{e-e'} = r^{N\cdot(e-e')} \bmod N$ with $(N, \varphi(N)) = 1$. Both e, e' are chosen uniformly at random from $\mathbb{Z}/N\mathbb{Z}$ and with overwhelming probability $e - e' \neq 1$ which means that $ord(r)|(e - e')$. So $e - e'$ is either a factor of $\varphi(N)$ (remember that N is a safe RSA prime) or it is a multiple of $\varphi(N)$, both of which allows to break the $CR[N]$ assumption. We can therefore assume that $k \neq k'$.

We obtain

$$k^N = c^e \cdot t \bmod N \text{ and } k'^N = c^{e'} \cdot t \bmod N. \tag{3}$$

By dividing the first by the second equation in Equation (3) we yield

$$(k/k')^N = c^{e-e'} \bmod N.$$

Now we observe that, with high probability, $(e - e', N) = 1$ (otherwise we could efficiently break the security of the underlying encryption scheme). Using the extended euclidean algorithm, there exist values $\alpha, \beta \in \mathbb{Z}$ such that $\alpha N + \beta(e - e') = 1$. One can use the randomness $v = (c \bmod N)^\alpha \cdot (k/k')^\beta \bmod N$ to decrypt c and obtain the plaintext pair (u, v). This must be the correct plaintext, because Enc is a bijection and

$$\begin{aligned} v^N &= (c)^{\alpha N} \cdot (k/k')^{\beta N} \bmod N \\ &= (c)^{\alpha N} \cdot (c)^{\beta(e-e')} \bmod N \\ &= c \bmod N. \end{aligned}$$

Honest-Verifier Zero-Knowledge. The simulator works as follows

(1) Sample $k \xleftarrow{\$} \mathbb{Z}/N\mathbb{Z}^*$, $e \xleftarrow{\$} \mathbb{Z}/N\mathbb{Z}$ uniformly at random.
(2) Set $t = k^N \cdot c^{-e} \bmod N$.
(3) Output (t, e, k).

The algorithm terminates because $(c \bmod N) \in \mathbb{Z}/N\mathbb{Z}^*$ and hence $c^{-e} \bmod N$ is well defined. One can easily verify that this is a correct transcript for $\Pi^{\mathrm{P}}_{\mathsf{PoPK}}$. Since the setup of the encryption scheme comes from a CRS, we have that $N^{-1} \bmod \varphi(N)$ is well defined and

$$\begin{aligned} \phi : \mathbb{Z}/N\mathbb{Z}^* &\to \mathbb{Z}/N\mathbb{Z}^* \\ x &\mapsto x^N \bmod N \end{aligned}$$

is bijective. Hence, for t coming from the simulation, an Nth root s must exist in $\mathbb{Z}/N\mathbb{Z}^*$. In the protocol, we choose s uniformly at random whereas we do this for k in the simulator. Since (for a fixed k, e) raising to the Nth power or computing the Nth root is a bijection, the distributions are perfectly indistinguishable.

Protocol $\Pi^{\mathrm{P}}_{\mathsf{PoM}}$

(1) $\mathtt{Pr}$ samples $t, u \xleftarrow{\$} \mathbb{Z}/N\mathbb{Z}, v \xleftarrow{\$} \mathbb{Z}/N\mathbb{Z}^*$. He then sends $f = z^t \cdot \mathsf{Enc}_{\mathsf{pk}}(u, v) \mod N^2$ to $\mathtt{Ve}$.
(2) $\mathtt{Ve}$ chooses a uniformly random $e \xleftarrow{\$} \mathbb{Z}/N\mathbb{Z}$ and sends it to $\mathtt{Pr}$.
(3) $\mathtt{Pr}$ computes $g = t + e \cdot b \mod N, h = u + e \cdot c \mod N, i = v \cdot r^e \mod N$ and and sends (g, h, i) to $\mathtt{Ve}$.
(4) $\mathtt{Ve}$ accepts if $z^g \cdot \mathsf{Enc}_{\mathsf{pk}}(h, i) = \hat{z}^e \cdot f \mod N^2$, and rejects otherwise.

Fig. 16. $\Pi^{\mathrm{P}}_{\mathsf{PoM}}$: Protocol to prove linear relation on ciphertexts.

Linear relations

Lemma 5. *The protocol $\Pi^{\mathrm{P}}_{\mathsf{PoM}}$ is an an interactive honest-verifier proof for the relation R_{PoM}.*

Proof.

Correctness. Let $\hat{z} = z^b \cdot \mathsf{Enc}_{\mathsf{pk}}(c, r) \bmod N^2$, then the elements are in the right group i.e. $z, \hat{z} \in \mathbb{Z}/N^2\mathbb{Z}^*$ as required. Choose g, h, i as in the protocol, and observe that

$$\begin{aligned} z^g \cdot \mathsf{Enc}_{\mathsf{pk}}(h, i) &= z^t \cdot z^{e \cdot b} \cdot \mathsf{Enc}_{\mathsf{pk}}(h, i) \bmod N^2 \\ &= z^t \cdot z^{e \cdot b} \cdot \mathsf{Enc}_{\mathsf{pk}}(u, v) \cdot \mathsf{Enc}_{\mathsf{pk}}(e \cdot c, r^e) \bmod N^2 \\ &= f \cdot z^{e \cdot b} \cdot \mathsf{Enc}_{\mathsf{pk}}(c, r)^e \bmod N^2 \\ &= f \cdot \hat{z}^e \bmod N^2. \end{aligned}$$

Soundness. Assume that $\mathtt{Ve}$ obtains two protocol transcripts $(f, e, (g, h, i))$, $(f, e', (g', h', i'))$ such that $e \neq e' \bmod N$. We require that $(g, h, i) \neq (g', h', i')$ (1-2 of the coordinates might agree) which is true if $\mathtt{Pr}$ were honest as it could otherwise break the security assumption in a similar way as in the proof of Lemma 4.

We obtain

$$z^g \cdot \mathsf{Enc}_{\mathsf{pk}}(h, i) = \hat{z}^e \cdot f \bmod N^2 \text{ and } z^{g'} \cdot \mathsf{Enc}_{\mathsf{pk}}(h', i') = \hat{z}^{e'} \cdot f \bmod N^2.$$

If we divide the first by the second equation, we yield

$$z^{g-g'} \cdot \mathsf{Enc}_{\mathsf{pk}}(h - h', i \cdot i'^{-1}) = \hat{z}^{e-e'} \bmod N^2.$$

Because $e \neq e' \bmod N$ we can compute the multiplicative inverse and raise both sides to that power. Let $\omega = (e - e')^{-1} \bmod N$ then

$$z^{(g-g') \cdot \omega} \cdot \mathsf{Enc}_{\mathsf{pk}}(h - h', i \cdot i'^{-1})^{\omega} = \hat{z}^{1+tN} \bmod N^2$$

for some $t \in \mathbb{Z}$ that we can compute. Now divide both sides by $\hat{z}^{tN}$, then

$$\begin{aligned} \hat{c} &= (N+1)^{\omega \cdot (h-h')} \big((i \cdot i'^{-1}) \cdot \omega\big)^N \cdot z^{(g-g')\omega} \cdot \hat{z}^{-tN} \bmod N^2 \\ &= (N+1)^{\omega \cdot (h-h')} \big((i \cdot i'^{-1}) \cdot \omega \cdot \hat{z}^{-t}\big)^N \cdot z^{(g-g')\omega} \bmod N^2. \end{aligned}$$

By setting $b = (g - g') \cdot \omega \bmod N$, $r = (i \cdot i'^{-1}) \cdot \omega \cdot \hat{z}^{-t} \bmod N$ and $c = \omega \cdot (h - h') \bmod N$ we obtain a witness for R_{PoM}.

Honest-Verifier Zero-Knowledge. To simulate $\Pi^{\mathrm{P}}_{\mathsf{PoM}}$, we use the following algorithm:

(1) Sample $e \xleftarrow{\$} \mathbb{Z}/N\mathbb{Z}$ uniformly at random.
(2) Sample $g, h \xleftarrow{\$} \mathbb{Z}/N\mathbb{Z}$, $i \xleftarrow{\$} \mathbb{Z}/N\mathbb{Z}^*$ uniformly at random.
(3) Compute $f \leftarrow z^g \cdot \mathsf{Enc}_{\mathsf{pk}}(h, i)/\hat{z}^{-e}$.
(4) Output $(f, e, (g, h, i))$.

Observe that the algorithm terminates, because $\hat{z}^{-e} \bmod N^2$ is well defined. Moreover, the algorithm outputs a transcript that is correct. Let $(f, e, (g, h, i))$ be a transcript generated by the simulator, then e is chosen as in the protocol. Moreover, we choose g, h, i from the same distribution that they have in the protocol, and the bijection property uniquely determines f (as argued already in the previous lemma).

6.2 Distributed decryption

Let us start with the idea behind the distributed decryption key and the decryption protocol that recovers the randomness completely (and why this is a good idea): for n parties $\mathtt{P}_1, \ldots, \mathtt{P}_n$, assume that there exists a sum sharing $\mathsf{sk}_1, \ldots, \mathsf{sk}_n$ such that $\mathsf{sk}_2, \ldots, \mathsf{sk}_n \xleftarrow{\$} \mathbb{Z}/2^\kappa N\mathbb{Z}$ and $\mathsf{sk}_1 \in \mathbb{Z}/2^\kappa N\mathbb{Z}$ such that $\mathsf{sk}_1 = \mathsf{sk} - (\sum_{i=2}^n \mathsf{sk}_i) \bmod \varphi(N)$ (this is the output for the shared secret key that $\mathcal{F}^{\mathrm{P}}_{\mathsf{KeyGenDec}}$ generates). To decrypt a ciphertext c, each party can now simply broadcast $c^{\mathsf{sk}_i} \bmod N$. Each party then multiplies together all the obtained values and uses the regular decryption procedure to recover x.

Protocol $\Pi^{\mathrm{P}}_{\mathsf{Dec}}$

Let c be a ciphertext. Each $\mathtt{P}_i$ holds a share sk_i of sk.

(1) Each $\mathtt{P}_i$ locally computes $t_i = c^{\mathsf{sk}_i} \bmod N$ and then broadcasts it.
(2) Each $\mathtt{P}_i$ locally computes

$$r = \prod_{j=1}^{n} t_i \bmod N,$$

as well as

$$x = \left(\left(c \cdot r^{-N} \bmod N^2\right) - 1\right)/N \bmod N.$$

(3) Each $\mathtt{P}_i$ checks that $c = \mathsf{Enc}_{\mathsf{pk}}(x, r)$. If yes then output (x, r) else output $\perp$.

Fig. 17. $\Pi^{\mathrm{P}}_{\mathsf{Dec}}$: Protocol to decrypt a Paillier ciphertext c by recovering its randomness.

The protocol $\Pi^{\mathrm{P}}_{\mathsf{Dec}}$ from Fig. 17 follows the above sketch. This, in fact, is only a good idea if the original randomness in c was chosen uniformly at random as we will see in the simulator.

Distributed decryption that does not recover the randomness. The above protocol $\Pi^{\mathrm{P}}_{\mathsf{Dec}}$ is already good enough for our application, since the ciphertexts and their randomness that we decrypt will always be uniformly random. For other applications, it may be necessary to hide the randomness during the decryption, hence we now show how to extend our approach to yield this property. Since we still want to rely on our above decryption routine we will have to cheat a little: instead of decrypting c directly we decrypt some $\hat{c}$ that is a rerandomization of c, but with uniform randomness. Therefore, each party provides a fresh encryption of 0 and proves in zero-knowledge that it indeed is such a ciphertext. Unfortunately this is not enough, at least if we want to use a proof where the simulator does not have the sk_i shares of the honest parties. For technical reasons, we have an additional round where each $\mathtt{P}_i$ commits to z_i i.e. its contribution to the rerandomization in advance.

In order to prove that a ciphertext decrypts to 0, we can use the proof from [15] which is depicted in Fig. 18. The proven statement is the same as in R^P_{PoPK} just with $x = 0$.

Protocol $\Pi^{\mathrm{P}}_{\mathsf{ZKZero}}$

$\mathtt{Pr}$ proves the relation R^{P}_{PoPK} for $x = 0$.

(1) $\mathtt{Pr}$ samples $s \xleftarrow{\$} \mathbb{Z}/N^2\mathbb{Z}^*$ and sends $t = s^N \mod N^2$ to $\mathtt{Ve}$.
(2) $\mathtt{Ve}$ samples $e \xleftarrow{\$} \mathbb{Z}/N\mathbb{Z}$ and sends it to $\mathtt{Pr}$.
(3) $\mathtt{Pr}$ sends $k = s \cdot r^e \mod N^2$ to $\mathtt{Ve}$.
(4) $\mathtt{Ve}$ accepts if $k^N = c^e \cdot t \mod N^2$ and otherwise rejects.

Fig. 18. $\Pi^{\mathrm{P}}_{\mathsf{ZKZero}}$: Protocol to prove that a Paillier ciphertext contains 0.

Lemma 6. *The protocol $\Pi^{\mathrm{P}}_{\mathsf{ZKZero}}$ is an interactive honest-verifier proof for the relation R^{P}_{PoPK} with $x = 0$.*

Proof. See [15, Lemma 2].

Interestingly, the protocols $\Pi^{\mathrm{P}}_{\mathsf{PoPK}}$ and $\Pi^{\mathrm{P}}_{\mathsf{ZKZero}}$ are very, except that the former uses values $\mod N$ whereas the latter uses $\mod N^2$ instead. However, under the hood, both protocols prove quite different statements:

- $\Pi^{\mathrm{P}}_{\mathsf{PoPK}}$ considers $c \bmod N$, where $(N+1)^x r^N \bmod N = r^N \bmod N$, i.e. the modular reduction removes the message part from the ciphertext. Based on this, we then prove existence of an r, which must be the same as in $\mathbb{Z}/N^2\mathbb{Z}^*$.
- $\Pi^{\mathrm{P}}_{\mathsf{ZKZero}}$ instead considers c in the original group $\mathbb{Z}/N^2\mathbb{Z}^*$ and proves knowledge of an Nth root of c where the root is from $\mathbb{Z}/N\mathbb{Z}^*$. Hence c must have the form $c = r^N \bmod N^2$ for some r i.e. $x = 0$.

The protocol $\Pi^{\mathrm{P}}_{\mathsf{DecAlt}}$ is given in Fig. 19 and follows directly the steps outlined above.

Protocol $\Pi^{\mathrm{P}}_{\mathsf{DecAlt}}$

Let c be a ciphertext. Each $\mathtt{P}_i$ holds a share sk_i of sk.

(1) Each $\mathtt{P}_i$ samples $r_i \xleftarrow{\$} \mathbb{Z}/N\mathbb{Z}^*$ and computes $z_i \leftarrow \mathsf{Enc}_{\mathsf{pk}}(0, r_i)$.
(2) Each $\mathtt{P}_i$ uses $\mathcal{F}_{\mathsf{Commit}}$ to commit to z_i to all parties.
(3) Each $\mathtt{P}_i$ opens the commitment from the previous step as z_i to all parties and proves knowledge of r_i using $\Pi^{\mathrm{P}}_{\mathsf{ZKZero}}$ to all parties. If a proof fails or a commitment was not opened correctly, abort.
(4) Set $\hat{c} = c \cdot \prod_{i \in [n]} z_i$.
(5) Each $\mathtt{P}_i$ locally computes $t_i = \hat{c}^{\mathsf{sk}_i} \bmod N$ and then broadcasts it.
(6) Each $\mathtt{P}_i$ locally computes

$$r = \prod_{j=1}^{n} t_i \bmod N,$$

as well as

$$x = \left(\left(\hat{c} \cdot r^{-N} \bmod N^2\right) - 1\right) / N \bmod N.$$

(7) Each $\mathtt{P}_i$ checks that $\hat{c} = \mathsf{Enc}_{\mathsf{pk}}(x, r)$. If yes then output x else output $\perp$.

Fig. 19. $\Pi^{\mathrm{P}}_{\mathsf{DecAlt}}$: Protocol to decrypt a Paillier ciphertext c without recovering the original randomness.

Theorem 3. *The protocols $\Pi^{\mathrm{P}}_{\mathsf{Dec}}$, $\Pi^{\mathrm{P}}_{\mathsf{DecAlt}}$ implement $\mathcal{F}^{\mathrm{P}}_{\mathsf{KeyGenDec}}$ in the $\mathcal{F}^{\mathrm{P}}_{\mathsf{KeyGen}}$, $\mathcal{F}_{\mathsf{Rand}}$, $\mathcal{F}_{\mathsf{Commit}}$-hybrid model with broadcast with security against a static adversary corrupting up to $n-1$ parties maliciously.*

Proof. In a nutshell, the proof uses that we are in the $\mathcal{F}^{\mathrm{P}}_{\mathsf{KeyGen}}$-hybrid model, which means that the simulator has the key shares sk_i of the dishonest parties. It samples and fixes random key shares for all but one of

Simulator $\mathcal{S}^{\mathrm{P}}_{\mathsf{Dec}}$

Let $\mathcal{J} = \mathtt{P} \setminus \widehat{\mathtt{P}}$ be the honest parties. For simplicity, assume $\mathtt{P}_1 \in \mathcal{J}$.

(1) Send $(\mathsf{Decrypt}, c)$ to $\mathcal{F}^{\mathrm{P}}_{\mathsf{KeyGenDec}}$. If this simulator runs for the first time, sample uniformly random $\mathsf{sk}'_i \xleftarrow{\$} \mathbb{Z}/2^{\kappa}N\mathbb{Z}$ for each $i \in \mathcal{J} \setminus \{1\}$.
(2) Send (Rand) to to $\mathcal{F}^{\mathrm{P}}_{\mathsf{KeyGenDec}}$ and obtain (x, r).
(3) For each $\mathtt{P}_i \in \mathcal{J} \setminus \{1\}$ send $t_i = c^{\mathsf{sk}'_i} \bmod N$. For $\mathtt{P}_1$, set $\alpha = 1 - \sum_{i \in \mathcal{J} \setminus \{1\}} \mathsf{sk}'_i - \sum_{j \in \widehat{\mathtt{P}}} \mathsf{sk}_j$ and send $t_1 = r^{\alpha}$.
(4) Wait for the t_j for the dishonest $\mathtt{P}_j \in \widehat{\mathtt{P}}$ and set $\beta = \sum_{j \in \widehat{\mathtt{P}}} \mathsf{sk}_j$. If

$$r^{N \cdot \beta} \bmod N = \prod_{j \in \widehat{\mathtt{P}}} t_j$$

then send (x, r) to $\mathcal{F}^{\mathrm{P}}_{\mathsf{KeyGenDec}}$, otherwise send $\perp$.

Fig. 20. $\mathcal{S}^{\mathrm{P}}_{\mathsf{Dec}}$: Simulator for the protocol $\Pi^{\mathrm{P}}_{\mathsf{Dec}}$.

the simulated honest parties, and adjusts the value sent by this party according to the decrypted value. The distributions of $\Pi^{\mathrm{P}}_{\mathsf{Dec}}$ and $\mathcal{S}^{\mathrm{P}}_{\mathsf{Dec}}$ differ in the potential abort in Step (3) of the protocol or in the distribution of the simulated t_i.

Consider accepting transcripts for $\Pi^{\mathrm{P}}_{\mathsf{Dec}}$. Let $s = \prod_{j \in \widehat{\mathtt{P}}} t_j \bmod N$ then

$$(c^N)^{\Sigma_{i \in \mathcal{J}} \mathsf{sk}_i} \cdot (c^N)^{\beta} = r = (c^N)^{\Sigma_{i \in \mathcal{J}} \mathsf{sk}_i} \cdot s \bmod N,$$

and equality follows because $\mathbb{Z}/N\mathbb{Z}^*$ is a group. Therefore, if the dishonest parties send a different s in the protocol then it will abort which is the same as in $\mathcal{S}^{\mathrm{P}}_{\mathsf{Dec}}$. Therefore, given $\mathtt{A}$ the abort happens in identical cases in the real and ideal world. Now consider the t_i values in the simulation, then for all but t_1 these are sampled as in $\Pi^{\mathrm{P}}_{\mathsf{Dec}}$. But by the exact same argument as above, there is exactly one t_1 such that the equation holds, which is the one chosen by the simulator. Since the simulator fixes its sk'_i for the simulated honest parties in the first run, the computed t_i are consistent over multiple protocol runs.

The proof for $\Pi^{\mathrm{P}}_{\mathsf{DecAlt}}$ follows along the same lines, but the simulator does not have the randomness r. To send the correct values in Step (3), the simulator chooses some randomness $r' \xleftarrow{\$} \mathbb{Z}/N\mathbb{Z}^*$ uniformly at random which will be the randomness of $\hat{c}$. Using the committed values, the simulator can then simulate a z_1 ciphertext as $z_1 = \mathsf{Enc}_{\mathsf{pk}}(x, r')/(c \cdot \prod_{i=2}^{n} z_i)$.

In the practical protocols we will again make the HVZK protocols secure against arbitrary adversaries by converting them into NIZKs using the Fiat-Shamir transform.

6.3 The preprocessing protocol

The preprocessing protocol $\Pi^{\mathrm{P}}_{\mathsf{Offline}}$, on a high level, runs in the following phases:

(1) In a first step, every party encrypts uniformly random values.
(2) Take $k+2$ values which define a polynomial A of degree $k+1$ uniquely (when considered as evaluations in the points $1, \ldots, k+2$). Interpolate this polynomial A in the next $k+2$ points locally, encrypt these points and prove that the encrypted values are indeed points that lie on A. Then the same is done for a polynomial B.
(3) An unreliable point-wise multiplication of A, B is performed. The resulting polynomial C is evaluated in a random point β, and it is checked whether the multiplicative relation holds. This is enough to check correctness of all triples due to the size of N.
(4) The points of C that we obtained from the last step are currently only available as ciphertexts. Reshare them among all parties as random shares.

Protocol $\Pi^{\mathrm{P}}_{\mathsf{Offline}}$ (Part 1)

A protocol to perform preprocessing for the SPDZ protocol using Paillier encryption.

Initialize: On input $(\mathsf{Init}, \mathbb{Z}/N\mathbb{Z})$ the parties do the following:

(1) Each $\mathtt{P}_i$ chooses $\alpha_i \xleftarrow{\$} \mathbb{Z}/N\mathbb{Z}$, broadcasts a fresh encryption $\langle\!\langle \alpha_i \rangle\!\rangle$ and proves knowledge of plaintext of $\langle\!\langle \alpha_i \rangle\!\rangle$ using $\Pi^{\mathrm{P}}_{\mathsf{PoPK}}$.
(2) The parties compute $\langle\!\langle \alpha \rangle\!\rangle = \prod_{i=1}^{n} \langle\!\langle \alpha_i \rangle\!\rangle$.
(3) Each $\mathtt{P}_i$ stores $\langle\!\langle \alpha \rangle\!\rangle$ as the encrypted MAC key and its share α_i of the MAC key.

Triples: On input (Triple, k) the parties do the following:

(1) For $j \in [k+2]$ each $\mathtt{P}_i$ samples $A_i(j), B_i(j) \xleftarrow{\$} \mathbb{Z}/N\mathbb{Z}$, computes $\langle\!\langle A_i(j) \rangle\!\rangle, \langle\!\langle B_i(j) \rangle\!\rangle$ and broadcasts it together with proofs of $\Pi^{\mathrm{P}}_{\mathsf{PoPK}}$.
(2) For $j \in [k+2]$ every party $\mathtt{P}_i$ defines the polynomials $A_i(\cdot), B_i(\cdot)$ using $A_i(j), B_i(j)$ as evaluations. Each party computes and broadcasts $(\langle\!\langle A_i(l) \rangle\!\rangle, \langle\!\langle B_i(l) \rangle\!\rangle)_{l=k+3,\ldots,2k+2}$ together with proofs of plaintext knowledge using $\Pi^{\mathrm{P}}_{\mathsf{PoPK}}$.
(3) The parties locally compute

$$\langle\!\langle A(l) \rangle\!\rangle = \prod_{i=1}^{n} \langle\!\langle A_i(l) \rangle\!\rangle \text{ and } \langle\!\langle B(l) \rangle\!\rangle = \prod_{i=1}^{n} \langle\!\langle B_i(l) \rangle\!\rangle.$$

(4) The parties sample $\beta \xleftarrow{\$} \mathcal{F}_{\mathsf{Rand}}(\mathbb{Z}/(N-2k-3)\mathbb{Z}) + 2k + 3$ so that $\beta \in \mathbb{Z}/N\mathbb{Z} \setminus \{0, \ldots, 2k+2\}$.
(5) Define $A^{\top}(\beta)$ to be the value $A(\beta)$ computed using Lagrange interpolation and the values $A(1), \ldots, A(k+2)$ and similarly $A^{\perp}(\beta)$ to be $A(\beta)$ computed using $A(1), \ldots, A(2k+2)$. Every $\mathtt{P}_i$ locally computes

$$\langle\!\langle A^{\dagger}(\beta) \rangle\!\rangle = \langle\!\langle A^{\top}(\beta) \rangle\!\rangle / \langle\!\langle A^{\perp}(\beta) \rangle\!\rangle \text{ and } \langle\!\langle B^{\dagger}(\beta) \rangle\!\rangle = \langle\!\langle B^{\top}(\beta) \rangle\!\rangle / \langle\!\langle B^{\perp}(\beta) \rangle\!\rangle.$$

(6) The parties decrypt $\langle\!\langle A^{\dagger}(\beta) \rangle\!\rangle, \langle\!\langle B^{\dagger}(\beta) \rangle\!\rangle$ and check whether $A^{\dagger}(\beta) = B^{\dagger}(\beta) = 0 \bmod N$. Otherwise they abort.

Fig. 21. $\Pi^{\mathrm{P}}_{\mathsf{Offline}}$: Protocol to generate correct random triples out of random single values from Paillier encryption, Part 1.

(5) For all of the shares of A, B, C that were generated in the protocol, products with the MAC key α are computed. Correctness of the multiplication with α is checked and if the check is passed, the MACs are reshared among the parties in the same way as the points of C.

The protocol $\Pi^{\mathrm{P}}_{\mathsf{Offline}}$ can be found in Fig. 21, Fig. 22 and Fig. 23.

7 Security of the Preprocessing from Paillier Encryption

Similarly as in Π_{Offline} we implicitly define polynomials, multiply them and evaluate the product in a random point. The following remark helps us to establish later that one such evaluation in a random point is in fact enough to prove that the polynomials were correctly multiplied.

Remark 2. Let $N = p \cdot q$ being an RSA modulus with $p < q$ and $f, g \in (\mathbb{Z}/N\mathbb{Z})[X], f \neq g$ with $\max\{\deg(f), \deg(g)\} = d$. Let moreover $x \xleftarrow{\$} \mathbb{Z}/N\mathbb{Z}$. Then

$$\Pr[(f-g)(x) = 0 \bmod N] < \frac{2 \cdot d}{p}.$$

Proof. The polynomial $f - g$ is non-zero modulo N, hence non-zero modulo p or q (or both). Moreover,

$$(f-g)(x) = 0 \bmod N \Leftrightarrow (f-g)(x) = 0 \bmod p \wedge (f-g)(x) = 0 \bmod q.$$

Protocol $\Pi^{\mathrm{P}}_{\mathsf{Offline}}$ (Part 2)

Triples:

(7) For $j \in [2k+2]$ each $\mathtt{P}_i$ chooses $r_{i,j} \xleftarrow{\$} \mathbb{Z}/N\mathbb{Z}^*$, computes encryptions

$$\langle\!\langle \hat{c}_{i,j} \rangle\!\rangle \leftarrow \langle\!\langle A(j) \rangle\!\rangle^{B_i(j)} \mathsf{Enc}_{\mathsf{pk}}(0, r_{i,j})$$

broadcasts the $\langle\!\langle \hat{c}_{i,j} \rangle\!\rangle$ and proves the relation using $\Pi^{\mathrm{P}}_{\mathsf{PoM}}$.

(8) For $j \in [2k+2]$ each $\mathtt{P}_i$ chooses $\tilde{c}_{i,j} \xleftarrow{\$} \mathbb{Z}/N\mathbb{Z}$, computes $\langle\!\langle \tilde{c}_{i,j} \rangle\!\rangle$ and broadcasts $(\langle\!\langle \tilde{c}_{i,j} \rangle\!\rangle)_{j \in \{0,\ldots,2k+3\}}$ together with proofs of $\Pi^{\mathrm{P}}_{\mathsf{PoPK}}$.

(9) For $j \in [2k+2]$ the parties locally compute

$$\langle\!\langle \hat{c}_j \rangle\!\rangle = \textstyle\prod_{i=1}^{n} \langle\!\langle \hat{c}_{i,j} \rangle\!\rangle / \prod_{i=1}^{n} \langle\!\langle \tilde{c}_{i,j} \rangle\!\rangle,$$

and publicly decrypt $\hat{c}_j$.

(10) For $j \in [2k+2]$ each party sets

$$\langle\!\langle C_i(j) \rangle\!\rangle := \begin{cases} \langle\!\langle \tilde{c}_{1,j} \rangle\!\rangle \cdot \langle\!\langle \hat{c}_j \rangle\!\rangle & \text{if } i = 1 \\ \langle\!\langle \tilde{c}_{i,j} \rangle\!\rangle & \text{else} \end{cases},$$

as well as $\langle\!\langle C(j) \rangle\!\rangle = \prod_{i=1}^{n} \langle\!\langle C_i(j) \rangle\!\rangle$ and its share of $C(j)$ as

$$C_i(j) := \begin{cases} \tilde{c}_{1,j} + \hat{c}_j & \text{if } i = 1 \\ \tilde{c}_{i,j} & \text{else} \end{cases}.$$

(11) The parties sample $\beta \xleftarrow{\$} \mathcal{F}_{\mathsf{Rand}}(\mathbb{Z}/(N-k-1)\mathbb{Z}) + k + 1$ so that $\beta \in \mathbb{Z}/N\mathbb{Z} \setminus \{0, \ldots, k\}$.

(12) The parties compute $\langle\!\langle A(\beta) \rangle\!\rangle, \langle\!\langle B(\beta) \rangle\!\rangle, \langle\!\langle C(\beta) \rangle\!\rangle$ locally using Lagrange interpolation and then decrypt these values.

(13) If $A(\beta) \cdot B(\beta) \neq C(\beta) \bmod N$ then abort.

(14) Each $\mathtt{P}_i$ chooses $s_i \in \mathbb{Z}/N\mathbb{Z}$, computes $\langle\!\langle s_i \rangle\!\rangle$ and broadcasts $\langle\!\langle s_i \rangle\!\rangle$ together with a proof of $\Pi^{\mathrm{P}}_{\mathsf{PoPK}}$. Let $s = \sum_i s_i$.

(15) We define the following abbreviation:

$$t_{i,j} := \begin{cases} s_i \text{ for } j = 0 \\ A_i(j) \text{ for } j = 1, \ldots, k \\ B_i(j) \text{ for } j = k+1, \ldots, 2k \\ C_i(j) \text{ for } j = 2k+1, \ldots, 3k \end{cases} \quad \text{and } t_j := \begin{cases} s \text{ for } j = 0 \\ A(j) \text{ for } j = 1, \ldots, k \\ B(j) \text{ for } j = k+1, \ldots, 2k \\ C(j) \text{ for } j = 2k+1, \ldots, 3k \end{cases}$$

Fig. 22. $\Pi^{\mathrm{P}}_{\mathsf{Offline}}$: Protocol to generate correct random triples out of random single values from Paillier encryption, Part 2.

If $f - g \neq 0 \bmod p$ then it will be zero in at most $d-1$ positions by the fundamental law of algebra. Since x is uniformly random mod$N$, it is also uniformly random mod$p$ and therefore

$$\Pr\left[(f-g)(x) = 0 \bmod p \mid f - g \neq 0 \bmod p\right] < \frac{d}{p}.$$

The same reasoning goes for the case mod$q$ where $d/q < d/p$. Hence whenever $f - g$ is 0 either modulo p or q then the above bound holds. By a union bound, this then applies to the case where $f - g$ is non-zero modulo both p, q. □

We make use of the above remark three times in our protocol. First in Step (6), where we use it to establish that the polynomial defined has only degree k. Second, we also use it in Step (13) to check that the triples are indeed multiplicative and also in Step (21) implicitly to establish that all MACs are correct (we can consider multiplication with the MAC α as multiplication with the constant polynomial α).

Protocol $\Pi^{\mathrm{P}}_{\mathsf{Offline}}$ (Part 3)

Triples:

(16) For each $j \in [3k] \cup \{0\}$ each $\mathtt{P}_i$ chooses $r_{i,j} \xleftarrow{\$} \mathbb{Z}/N\mathbb{Z}^*$ and computes

$$\langle\!\langle t_{i,j} \cdot \alpha \rangle\!\rangle \leftarrow \langle\!\langle \alpha \rangle\!\rangle^{t_{i,j}} \cdot \mathsf{Enc}_{\mathsf{pk}}(0, r_{i,j}),$$

then broadcasts $(\langle\!\langle t_{i,j} \cdot \alpha \rangle\!\rangle)$ and proves the relation using $\Pi^{\mathrm{P}}_{\mathsf{PoM}}$.

(17) For each $j = [3k] \cup \{0\}$ $\mathtt{P}_1, \ldots, \mathtt{P}_n$ compute

$$\langle\!\langle t_j \cdot \alpha \rangle\!\rangle = \prod_{i=1}^{n} \langle\!\langle t_{i,j} \cdot \alpha \rangle\!\rangle.$$

(18) The parties sample $\beta \xleftarrow{\$} \mathcal{F}_{\mathsf{Rand}}(\mathbb{Z}/N\mathbb{Z})$.

(19) All parties compute

$$\langle\!\langle v \rangle\!\rangle = \prod_{j=0}^{3k} \langle\!\langle t_j \rangle\!\rangle^{\beta^j} \text{ and } \langle\!\langle v' \rangle\!\rangle = \prod_{j=0}^{3k} \langle\!\langle t_j \cdot \alpha \rangle\!\rangle^{\beta^j}.$$

(20) The parties jointly decrypt $\langle\!\langle v \rangle\!\rangle$ to v and check that the decryption was correct.

(21) The parties jointly decrypt

$$\langle\!\langle M \rangle\!\rangle = \langle\!\langle \alpha \rangle\!\rangle^{v} / \langle\!\langle v' \rangle\!\rangle$$

and verify that $M = 0$, otherwise they abort. All parties verify correctness of decryption.

(22) For each $j \in [3k]$ each $\mathtt{P}_i$ chooses $m_{i,j} \xleftarrow{\$} \mathbb{Z}/N\mathbb{Z}$, computes $\langle\!\langle m_{i,j} \rangle\!\rangle$ and broadcasts $\langle\!\langle m_{i,j} \rangle\!\rangle$ together with proofs of $\Pi^{\mathrm{P}}_{\mathsf{PoPK}}$.

(23) For each $j \in [3k]$ the parties compute

$$\langle\!\langle O_j \rangle\!\rangle = \langle\!\langle t_j \cdot \alpha \rangle\!\rangle / \prod_{i=1}^{n} \langle\!\langle m_{i,j} \rangle\!\rangle$$

and publicly decrypt $\langle\!\langle O_j \rangle\!\rangle$. All parties verify correctness of decryption.

(24) For each $j \in [3k]$, each $\mathtt{P}_i$ determines its share $\gamma(t_j)_i$, of the MAC $\gamma(t_j)$ of t_j as

$$\gamma(t_j)_i := \begin{cases} O_j + m_{i,j} & \text{for } i = 1 \\ m_{i,j} & \text{for } 1 < i \leq n \end{cases}$$

(25) Each party $\mathtt{P}_i$ uses $t_{i,j}, \gamma(t_j)_i$ as its shares of $\langle t_j \rangle$.

Fig. 23. $\Pi^{\mathrm{P}}_{\mathsf{Offline}}$: Protocol to generate correct random triples out of random single values from Paillier encryption, Part 3.

Theorem 4. *The protocol $\Pi^{\mathrm{P}}_{\mathsf{Offline}}$ securely implements the functionality $\mathcal{F}_{\mathsf{Offline}}$ using a random oracle and a broadcast channel in the $\mathcal{F}^{\mathrm{P}}_{\mathsf{KeyGenDec}}$ hybrid model with security against any malicious static PPT adversary controlling at most $n-1$ parties if the $CR[N]$-problem is hard.*

Our security proof uses, on a high level, the following idea: we allow the simulator to decrypt the ciphertexts that it obtains from the dishonest parties. This itself is not UC secure, but assume that there exists an environment $\mathtt{Z}$ that can distinguish the transcript from such a simulator from a protocol transcript. Then we can use such an $\mathtt{Z}$ as a subroutine to break the IND-CPA security. Such a subroutine can rewind $\mathtt{Z}$ and thereby avoid decryption altogether, so it does not rely on having the secret key from the setup.

The functionality $\mathcal{F}_{\mathsf{Offline}}$ that our protocol implements is weaker than what $\Pi^{\mathrm{P}}_{\mathsf{Offline}}$ achieves, because it does not allow the adversary to introduce errors into the MACs (which is in fact possible in Π_{Offline} and hence allowed in $\mathcal{F}_{\mathsf{Offline}}$).

Proof (Proof of Theorem 4.). For the proof, we use the simulator $\mathcal{S}^{\mathrm{P}}_{\mathsf{Offline}}$ as depicted in Fig. 24. This simulator uses decryption since it has the secret key for the encryption scheme. For the sake of contradiction, assume that there exists an environment $\mathtt{Z}$ that can distinguish between protocol transcripts of $\Pi^{\mathrm{P}}_{\mathsf{Offline}}$ and simulated transcripts of $\mathcal{S}^{\mathrm{P}}_{\mathsf{Offline}}$ with non-negligible probability σ in polynomial time. Now, we show how to use the distinguisher to break the IND-CPA property of the encryption scheme.

Simulator $\mathcal{S}^{\mathrm{P}}_{\mathsf{Offline}}$

In this simulator, we make the assumption that the secret key is known. Let $\widehat{\mathtt{P}}$ be the malicious parties. Let k be the number of triples.

SimulateInitialize:

(1) For each simulated honest party $\mathtt{P}_i \in \mathtt{P} \setminus \widehat{\mathtt{P}}$, provide a fresh encryption of the random value $r_i \xleftarrow{\$} \mathbb{Z}/N\mathbb{Z}$ which is sent to the dishonest parties as $c_i = \langle\langle r_i \rangle\rangle$. Then prove plaintext knowledge using $\Pi^{\mathrm{P}}_{\mathsf{PoPK}}$. Set the flag $\mathsf{cheated} = \bot$.
(2) Decrypt the ciphertexts c_i that are obtained from the malicious parties $\mathtt{P}_i \in \widehat{\mathtt{P}}$. Send $(\mathsf{Init}, \mathbb{Z}/N\mathbb{Z})$ and the plaintexts in the name of the malicious parties to $\mathcal{F}_{\mathsf{Offline}}$.
(3) Locally compute $\langle\langle \alpha \rangle\rangle = \prod_{i=1}^{n} \langle\langle c_i \rangle\rangle$ and $\alpha = \prod_{i=1}^{n} c_i$.

SimulateTriples:

(1) Perform Step (1) as in the protocol. Decrypt the ciphertexts of $\mathtt{P}_i \in \widehat{\mathtt{P}}$ after Step (1). Then compute the polynomials $A_i(\cdot), B_i(\cdot)$ as defined in Step (2).
(2) Perform Step (2) as in the protocol. Decrypt the ciphertexts of $\mathtt{P}_i \in \widehat{\mathtt{P}}$ after Step (2) as $A'_i(l), B'_i(l)$. Now set $\mathsf{cheated}$ as

$$\mathsf{cheated} = \begin{cases} \bot & \text{if } \forall i, l: \ A_i(l) = A'_i(l) \bmod N \text{ and } B_i(l) = B'_i(l) \bmod N \\ \top & \text{else} \end{cases}.$$

(3) Perform Steps $(3) - (6)$ as in the protocol. Abort if $\mathsf{cheated} = \top$.
(4) Perform Step (7) as in the protocol. Decrypt the ciphertexts of $\mathtt{P}_i \in \widehat{\mathtt{P}}$ as $\hat{c}'_{i,j}$. Set $\mathsf{cheated}$ as

$$\mathsf{cheated} = \begin{cases} \bot & \text{if } \forall i, j: \ \hat{c}'_{i,j} = A(j) \cdot B_i(j) \\ \top & \text{else} \end{cases}.$$

(5) Perform Steps $(8) - (13)$ as in the protocol. Abort in Step (13) if $\mathsf{cheated} = \top$.
(6) Send (Triple, k) to $\mathcal{F}_{\mathsf{Offline}}$. Let $\boldsymbol{A}_i[j] = A_i(j), \boldsymbol{C}_i[j] = C_i(j), \boldsymbol{C}_i[j] = C_i(j)$ from the protocol. Then send $\boldsymbol{A}_i, \boldsymbol{B}_i, \boldsymbol{C}_i$ to $\mathcal{F}_{\mathsf{Offline}}$ for each $\mathtt{P}_i \in \widehat{\mathtt{P}}$ and also $\boldsymbol{\Delta}_{\gamma(\boldsymbol{A})} = \boldsymbol{\Delta}_{\gamma(\boldsymbol{B})} = \boldsymbol{\Delta}_{\gamma(\boldsymbol{C})} = \mathbf{0}$.
(7) Perform Steps $(14) - (16)$ as in the protocol, and decrypt all values obtained from players controlled by $\mathtt{A}$. If in one of the ciphertexts $\langle\langle t_{i,j} \cdot \alpha \rangle\rangle$ of the dishonest parties is not of the form $\alpha \cdot t_{i,j}$ for the respective encrypted value $\langle\langle t_{i,j} \rangle\rangle$ then set $\mathsf{cheated} = \top$, otherwise $\mathsf{cheated} = \bot$.
(8) Perform Steps $(17) - (21)$ as in the protocol. In Step (21) abort if $\mathsf{cheated} = \top$.
(9) Perform Steps $(22) - (25)$ as in the protocol.
(10) Set $\boldsymbol{\gamma(t)}[j] = \gamma(t_j)_i$ from the protocol. For each $\mathtt{P}_i \in \widehat{\mathtt{P}}$ send $\boldsymbol{\gamma(A)}_i, \boldsymbol{\gamma(B)}_i, \boldsymbol{\gamma(C)}_i$ to $\mathcal{F}_{\mathsf{Offline}}$ during the macro $\mathsf{SpdzRep}$.

Fig. 24. $\mathcal{S}^{\mathrm{P}}_{\mathsf{Offline}}$: Simulator for the protocol $\Pi^{\mathrm{P}}_{\mathsf{Offline}}$.

Let Π_π be the distribution of a real protocol execution, and Π_{sim} be the distribution that $\mathcal{S}^{\mathrm{P}}_{\mathsf{Offline}}$ outputs. We first define $\Pi_{Real,ZK}$ to be the distribution of a real protocol execution where the zero-knowledge proofs are replaced with simulations of the proofs. The simulations of the proofs are perfectly indistinguishable from real proofs, and therefore $\Pi_\pi \overset{p}{\approx} \Pi_{Real,ZK}$. Defining $\Pi_{Sim,ZK}$ for Π_{sim} in the same way, it also holds that $\Pi_{sim} \overset{p}{\approx} \Pi_{Sim,ZK}$.

In a next step, we replace the checks for correctness of statements as they are done in the protocol $\Pi^{\mathrm{P}}_{\mathsf{Offline}}$, with checks as in $\mathcal{S}^{\mathrm{P}}_{\mathsf{Offline}}$. That is, instead of choosing a random element, then evaluating the polynomial at that position, and then decrypting the result and comparing, we abort if the statement is not true. Formally, we do the following

(1) In Step (6) of $\Pi^{\mathrm{P}}_{\mathsf{Offline}}$, abort under the same conditions as in $\mathcal{S}^{\mathrm{P}}_{\mathsf{Offline}}$ in Step (3).
(2) In Step (13) of $\Pi^{\mathrm{P}}_{\mathsf{Offline}}$, abort under the same conditions as in $\mathcal{S}^{\mathrm{P}}_{\mathsf{Offline}}$ in Step (5).
(3) In Step (21) of $\Pi^{\mathrm{P}}_{\mathsf{Offline}}$, abort under the same conditions as in $\mathcal{S}^{\mathrm{P}}_{\mathsf{Offline}}$ in Step (8).

Let us denote the resulting distribution as $\Pi_{Real,ZK,Correct}$, A_Π be the event that a check in the protocol fails and $A_\mathcal{S}$ be the event that the check fails in $\mathcal{S}^{\mathrm{P}}_{\mathsf{Offline}}$. We first observe that, if the statement is true, the check will never fail. Thereby, $A_\Pi \subseteq A_\mathcal{S}$. Conversely, by letting the degree of the polynomials be polynomial in the security parameter κ and by Remark 2, we conclude that $\Pr[A_\mathcal{S}] - \Pr[A_\Pi] < \mathsf{negl}(\kappa)$ and thereby $\Pi_{Real,ZK,Correct} \overset{s}{\approx} \Pi_{Real,ZK}$.

Consider an environment $\mathtt{Z}$ that can distinguish between Π_π and Π_{sim}. Since $\mathtt{Z}$ runs in polynomial time and by the above reasoning, it will distinguish $\Pi_{Real,ZK,Correct}$ and $\Pi_{Sim,ZK}$ with essentially the same advantage σ. Run the following algorithm B with $\mathtt{Z}$ as a subroutine, which is used to generate all the interactions of the malicious parties. On a high level, B will run both $\Pi^{\mathrm{P}}_{\mathsf{Offline}}, \mathcal{S}^{\mathrm{P}}_{\mathsf{Offline}}$ synchronously and combine the messages to $\mathtt{Z}$ from both instances so that they are consistent with exactly one of them.

(1) Obtain the public key pk from the challenger $\mathtt{C}$. Start simulation for $\mathcal{F}^{\mathrm{P}}_{\mathsf{KeyGenDec}}$ and the random oracle. We simulate $\mathcal{F}^{\mathrm{P}}_{\mathsf{KeyGenDec}}$ as follows:
  (1.1) If $\mathcal{F}^{\mathrm{P}}_{\mathsf{KeyGenDec}}$ is queried to generate a key by all parties, then first sample n uniformly random values $r_1, \ldots, r_n$ from the same domain as in $\mathcal{F}^{\mathrm{P}}_{\mathsf{KeyGenDec}}$. Then output (pk, r_i) to each party $\mathtt{P}_i$.
  (1.2) If $\mathcal{F}^{\mathrm{P}}_{\mathsf{KeyGenDec}}$ is queried to decrypt a ciphertext c, then we instead send c, m' to $\mathtt{A}$ where m' is chosen by this modified algorithm. We will later describe how each such m' is chosen.
(2) Send $m_0 = 0, m_1 = 1$ to $\mathtt{C}$ and obtain the challenge c_q. Set $c_0 = c_q, c_1 = \langle\langle 1 \rangle\rangle / c_q$.
(3) Simulate how messages would be generated in the real protocol and in the simulator. Whenever a simulated honest party would send an encrypted message, there are now two choices m_0 for the message as in the protocol and m_1 as in the simulated case. We send the encrypted value $c_0^{m_0} \cdot c_1^{m_1} \cdot \langle\langle 0 \rangle\rangle$ instead, unless stated otherwise below.
(4) For every zero-knowledge proof to be given, use the simulator of the proof and the programmable random oracle to simulate the proof.
(5) For every encrypted value that an adversarial party sends, extract the input from the zero-knowledge proofs by rewinding $\mathtt{Z}$.
(6) To simulate the decryption in $\mathcal{F}^{\mathrm{P}}_{\mathsf{KeyGenDec}}$, we will provide the decrypted message m' as follows:
  (6.1) If we decrypt in Step $(6), (13), (21)$ in $\Pi^{\mathrm{P}}_{\mathsf{Offline}}$ or Step $(3), (5), (8)$ in $\mathcal{S}^{\mathrm{P}}_{\mathsf{Offline}}$ then we output 0 for $\mathcal{F}^{\mathrm{P}}_{\mathsf{KeyGenDec}}$ if $\mathsf{cheated} = \bot$ or a random non-zero[11] number if $\mathsf{cheated} = \top$.
  (6.2) In Step (8), let $\mathtt{P}_i$ be an honest party. For all other honest parties, generate encryptions as described above. For $\mathtt{P}_i$, sample a random $\delta_j \xleftarrow{\$} \mathbb{Z}/N\mathbb{Z}$ and compute $\langle\langle \tilde{c}_{i,j} \rangle\rangle = c_0^{x_0} c_1^{x_1} \cdot \langle\langle 0 \rangle\rangle$, where

$$x_0 = \sum_{r \in [n]} \hat{c}_{r,j} - \delta_j - \sum_{\mathtt{P}_r \in \mathtt{P} \setminus \widehat{\mathtt{P}} \cup \{\mathtt{P}_i\}} \tilde{c}_{r,j} \text{ and}$$
$$x_1 = \sum_{r \in [n]} \hat{c}'_{r,j} - \delta_j - \sum_{\mathtt{P}_r \in \mathtt{P} \setminus \widehat{\mathtt{P}} \cup \{\mathtt{P}_i\}} \tilde{c}'_{r,j}.$$

  The values $\hat{c}_{r,j}, \tilde{c}_{r,j}$ come from the real protocol and $\hat{c}'_{r,j}, \tilde{c}'_{r,j}$ from the modified $\mathcal{S}^{\mathrm{P}}_{\mathsf{Offline}}$. Then in Step (9), output $\hat{c}_j = \delta_j - \sum_{i \in \widehat{\mathtt{P}}} \tilde{c}_{i,j}$.
  (6.3) In Step (22), let $\mathtt{P}_i$ be an honest party. For all other honest parties, generate encryptions as described above. For $\mathtt{P}_i$, sample a random $\delta_j \xleftarrow{\$} \mathbb{Z}/N\mathbb{Z}$ and compute $\langle\langle m_{i,j} \rangle\rangle = c_0^{x_0} c_1^{x_1} \cdot \langle\langle 0 \rangle\rangle$, where

$$x_0 = t_j \cdot \alpha - \delta_j - \sum_{\mathtt{P}_r \in \mathtt{P} \setminus \widehat{\mathtt{P}} \cup \{\mathtt{P}_i\}} m_{r,j} \text{ and}$$
$$x_1 = t'_j \cdot \alpha' - \delta_j - \sum_{\mathtt{P}_r \in \mathtt{P} \setminus \widehat{\mathtt{P}} \cup \{\mathtt{P}_i\}} m'_{r,j}.$$

  The values $t_j \cdot \alpha, m_{r,j}$ come from the real protocol and $t'_j \cdot \alpha', m'_{r,j}$ from the modified $\mathcal{S}^{\mathrm{P}}_{\mathsf{Offline}}$. Then in Step (23), output $O_j = \delta_j - \sum_{i \in \widehat{\mathtt{P}}} m_{i,j}$.

[11] We can also just abort since the adversary has cheated.

(6.4) In Step (14), first choose a random value β and set $\mathcal{F}_{\mathsf{Rand}}$ to output β in Step (18). Let $\mathtt{P}_i$ be an honest party. For all other honest parties, generate encryptions as described above for $\langle\langle s_r \rangle\rangle, \mathtt{P}_r \in \mathtt{P} \setminus \widehat{\mathtt{P}} \cup \{\mathtt{P}_i\}$ in Step (14). For $\mathtt{P}_i$, sample a random $\delta \xleftarrow{\$} \mathbb{Z}/N\mathbb{Z}$ and compute $\langle\langle s_i \rangle\rangle = c_0^{x_0} c_1^{x_1} \cdot \langle\langle 0 \rangle\rangle$, where

$$x_0 = \delta - \sum_{\mathtt{P}_r \in \mathtt{P} \setminus \widehat{\mathtt{P}} \cup \{\mathtt{P}_i\}} s_r - \sum_{j=1}^{3k} \sum_{\mathtt{P}_r \in \mathtt{P}} t_{r,j} \cdot \beta^j \text{ and}$$

$$x_1 = \delta - \sum_{\mathtt{P}_r \in \mathtt{P} \setminus \widehat{\mathtt{P}} \cup \{\mathtt{P}_i\}} s'_r - \sum_{j=1}^{3k} \sum_{\mathtt{P}_r \in \mathtt{P}} t'_{r,j} \cdot \beta^j.$$

The values $s_r, t_{r,j}$ come from the real protocol and $s'_r, t'_{r,j}$ from the modified $\mathcal{S}^{\mathrm{P}}_{\mathsf{Offline}}$. Then in Step (20), output the value $\delta + \sum_{i \in \widehat{\mathtt{P}}} s_i$ for the decryption of $\langle\langle v \rangle\rangle$.

(7) Finally, after obtaining the guess b_g from $\mathtt{Z}$, send b_g to $\mathtt{C}$.

The correctness of Steps (6.2), (6.3), (6.4) follows by combining the values in both cases with the equations in the protocol, which is left out here.

Let $\mathtt{C}$ choose to encrypt $c_q = \langle\langle 0 \rangle\rangle$ with probability ρ and hence $c_q = \langle\langle 1 \rangle\rangle$ with probability $1 - \rho$. If $c_q = \langle\langle 0 \rangle\rangle$, then the distribution of B is exactly the same as $\Pi_{Real,ZK,Correct}$, which will happen with probability ρ. If, on the other hand, $c_q = \langle\langle 1 \rangle\rangle$, then the distribution will be $\Pi_{Sim,ZK}$ which will happen with probability $1 - \rho$. Since $\mathtt{Z}$ can distinguish both $\Pi_{Real,ZK,Correct}, \Pi_{Sim,ZK}$ with advantage at least σ, the output to $\mathtt{C}$ will be correct with non-negligible advantage at least σ, contradicting that the encryption scheme is IND-CPA secure. □

Acknowledgments

We thank Nigel Smart and Michael Walter for helpful discussions.

References

1. Carsten Baum, Ivan Damgård, Kasper Larsen, and Michael Nielsen. How to prove knowledge of small secrets. In *Advances in Cryptology-CRYPTO 2016*. Springer, 2016.
2. Donald Beaver. Efficient multiparty protocols using circuit randomization. In *Advances in Cryptology — CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pages 420–432. Springer, Berlin, Germany, 1992.
3. Eli Ben-Sasson, Serge Fehr, and Rafail Ostrovsky. Near-linear unconditionally-secure multiparty computation with a dishonest minority. In *Advances in Cryptology–CRYPTO 2012*, pages 663–680. Springer, 2012.
4. Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In *Advances in Cryptology – EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 169–188. Springer Berlin Heidelberg, 2011.
5. Fabrice Benhamouda, Jan Camenisch, Stephan Krenn, Vadim Lyubashevsky, and Gregory Neven. Better zero-knowledge proofs for lattice encryption and their application to group signatures. In *Advances in Cryptology–ASIACRYPT 2014*, pages 551–572. Springer, 2014.
6. Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, et al. Secure multiparty computation goes live. In *Financial Cryptography and Data Security*, pages 325–343. Springer, 2009.
7. Zvika Brakerski, Craig Gentry, and Shai Halevi. Packed ciphertexts in lwe-based homomorphic encryption. In *Public-Key Cryptography–PKC 2013*, pages 1–13. Springer, 2013.
8. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, pages 309–325, New York, NY, USA, 2012. ACM.
9. Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) lwe. *SIAM Journal on Computing*, 43(2):831–871, 2014.

10. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Foundations of Computer Science, 2001. Proceedings. 42nd IEEE Symposium on*, pages 136–145. IEEE, 2001.
11. Ronald Cramer and Ivan Damgård. On the amortized complexity of zero-knowledge protocols. In *Advances in Cryptology-CRYPTO 2009*, pages 177–191. Springer, 2009.
12. Ronald Cramer, Ivan Damgård, and Jesper Nielsen. Multiparty computation from threshold homomorphic encryption. *Advances in Cryptology—EUROCRYPT 2001*, pages 280–300, 2001.
13. Ronald Cramer, Ivan Damgård, and Valerio Pastro. On the amortized complexity of zero knowledge protocols for multiplicative relations. In *Information Theoretic Security*, pages 62–79. Springer, 2012.
14. Ronald Cramer, Ivan Damgård, and Jesper Nielsen. *Secure Multiparty Computation and Secret Sharing*. 2015.
15. Ivan Damgård and Mads Jurik. A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In *Public Key Cryptography*, pages 119–136. Springer, 2001.
16. 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 *Computer Security–ESORICS 2013*, pages 1–18. Springer, 2013.
17. Ivan Damgård and Jesper Buus Nielsen. Universally composable efficient multiparty computation from threshold homomorphic encryption. In *Advances in Cryptology-CRYPTO 2003*, pages 247–264. Springer, 2003.
18. Ivan Damgård and Sarah Zakarias. Constant-overhead secure computation of boolean circuits using preprocessing. In *Theory of Cryptography*, pages 621–641. Springer, 2013.
19. Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 643–662. Springer, Berlin, Germany, 2012.
20. Amos Fiat and Adi Shamir. How to prove yourself: practical solutions to identification and signature problems. In *Advances in Cryptology—CRYPTO'86*, pages 186–194. Springer, 1987.
21. Ulrich Fincke and Michael Pohst. Improved methods for calculating vectors of short length in a lattice, including a complexity analysis. *Mathematics of computation*, 44(170):463–471, 1985.
22. Tore Kasper Frederiksen, Marcel Keller, Emmanuela Orsini, and Peter Scholl. A unified approach to MPC with preprocessing using OT. In *Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part I*, pages 711–735, 2015.
23. Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *Advances in Cryptology–CRYPTO 2013*, pages 75–92. Springer, 2013.
24. Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 218–229. ACM, 1987.
25. Guillaume Hanrot, Xavier Pujol, and Damien Stehlé. Algorithms for the shortest and closest lattice vector problems. In *Coding and Cryptology*, pages 159–190. Springer, 2011.
26. Arjen Klaas Lenstra, Hendrik Willem Lenstra, and László Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, 1982.
27. Yehuda Lindell, Benny Pinkas, Nigel P Smart, and Avishay Yanai. Efficient constant round multi-party computation combining bmr and spdz. In *Advances in Cryptology–CRYPTO 2015*, pages 319–338. Springer, 2015.
28. San Ling, Khoa Nguyen, Damien Stehlé, and Huaxiong Wang. Improved zero-knowledge proofs of knowledge for the isis problem, and applications. In *Public-Key Cryptography–PKC 2013*, pages 107–124. Springer, 2013.
29. Vadim Lyubashevsky. Lattice-based identification schemes secure under active attacks. In *Public Key Cryptography–PKC 2008*, pages 162–179. Springer, 2008.
30. Vadim Lyubashevsky. Fiat-shamir with aborts: Applications to lattice and factoring-based signatures. In *Advances in Cryptology–ASIACRYPT 2009*, pages 598–616. Springer, 2009.
31. Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 681–700. Springer Berlin Heidelberg, 2012.
32. Jesper Buus Nielsen and Claudio Orlandi. Lego for two-party secure computation. In *Theory of Cryptography*, pages 368–386. Springer, 2009.
33. Claudio Orlandi. *Secure Computation in Untrusted Environments*. PhD thesis, Aarhus University, 2011.
34. Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in cryptology—EUROCRYPT'99*, pages 223–238. Springer, 1999.
35. Andrew Chi-Chih Yao. How to generate and exchange secrets. In *Foundations of Computer Science, 1986., 27th Annual Symposium on*, pages 162–167. IEEE, 1986.

A Concluding the Proof of Theorem 2

In this appendix, we will provide the material necessary to conclude the proof of Theorem 2. To do so, we will first show how to solve matrixGame instances efficiently. Our algorithmic approach allows for k to be as large as ≈ 40, while the runtime of the solver is mostly independent of m, n. After establishing a high-level intuition of a potential algorithm, we conclude the proof of the theorem which crucially relies on the quantities our algorithm outputs.

A.1 Computing matrixGame efficiently

First, let us describe how computing the problem can be simplified by putting it into a *normal form*. This is depicted in Fig. 25.

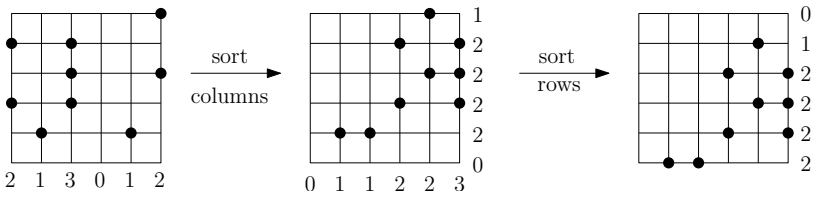

Fig. 25. Normal Form, Part 1

Here, we reorder the rows such that for rows i, j and row sums r_i, r_j it must hold that $i < j \Rightarrow r_i \leq r_j$ (note that if $r_i = r_j$ then we leave their order relative to each other untouched). A similar permutation is then applied to the columns with column sums c_i. It must hold that $\sum r_i = \sum c_i = k$, hence we can conversely start by sampling all such possible sequences $(r_i)_{i\in[n]}, (c_i)_{i\in[m]}$ (that are monotone, have non-zero coefficients and add up to k), then consider how many variations of the k balls fit such a pair of sequences, and multiply it with the number of possible row and column permutations that are described above.[12]. To obtain the number of row permutations R we have to consider that for each such permutation π rows of equal weight keep relative order, i.e. $i < j, r_i = r_j \Rightarrow \pi(i) < \pi(j)$. Now define $r_i^a = |\{t \mid r_t = i\}|$, then the number of permutations equals the multinomial coefficient

$$R = \binom{n}{r_0^a, r_1^a, \ldots, r_k^a}$$

Similarly, one can obtain the number of column permutations C. Using R, C, $(r_i)_{i\in[n]}, (c_i)_{i\in[m]}$ and the number of solutions for each possible pair $(r_i)_{i\in[n]}, (c_i)_{i\in[m]}$ one can cover the whole solution space while only enumerating a small subspace.

A first attempt - Lattice enumeration. In a first solution attempt, we will show how the problem reduces to a special case of lattice point enumeration. This can best be seen by an example: let $n = 3, m = 2$. At each point (i, j) of the grid, there is either a ball or not. Hence we can model each such point as a variable $x_{i,j} \in \{0, 1\}$, which looks as follows:

$$\begin{array}{c|cc} r_1 & x_{1,1} & x_{1,2} \\ r_2 & x_{2,1} & x_{2,2} \\ r_3 & x_{3,1} & x_{3,2} \\ \hline & c_1 & c_2 \end{array}$$

[12] One can additionally prune the search tree by the following observation: if a column contains c_i elements, then there must be at least c_i nonzero rows. More formally, $\max\{c_i\} \leq \sum_{r_i \neq 0} 1$ and $\max\{r_i\} \leq \sum_{c_i \neq 0} 1$.

It must hold that $r_i = \sum_j x_{i,j}, c_i = \sum_j x_{j,i}$. We can write this in homogeneous form as

$$\begin{aligned} -r_1 \ \ +x_{1,1} +x_{1,2} &= 0 \\ -r_2 \ \ +x_{2,1} +x_{2,2} &= 0 \\ -r_3 \ \ +x_{3,1} +x_{3,2} &= 0 \\ -c_1 +x_{1,1} +x_{2,1} +x_{3,1} &= 0 \\ -c_2 +x_{1,2} +x_{2,2} +x_{3,2} &= 0 \end{aligned}$$

Finding solutions to the above system of equations is equivalent to finding all $\boldsymbol{x} \in \ker \boldsymbol{T} \setminus \{\boldsymbol{0}\}, \boldsymbol{x} \in \{0,1\}^7, \boldsymbol{x}[1] = 1$, where

$$\boldsymbol{T} = \begin{pmatrix} -r_1 & 1 & 1 & 0 & 0 & 0 & 0 \\ -r_2 & 0 & 0 & 1 & 1 & 0 & 0 \\ -r_3 & 0 & 0 & 0 & 0 & 1 & 1 \\ -c_1 & 1 & 0 & 1 & 0 & 1 & 0 \\ -c_2 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

which can be solved by enumerating all vectors of norm $l_\infty = 1$ from the lattice $\Lambda(\ker \boldsymbol{T})$. Obtaining a basis $\boldsymbol{B}$ of $\Lambda(\ker \boldsymbol{T})$ is computationally cheap. This basis can then be LLL-reduced ([26]) and the short vectors be enumerated with the algorithm from [21] due to Fincke and Pohst[13]. A drawback of the above approach is that all short vectors are enumerated, which includes those that have $-1, 0, 1$-coefficients which we do not consider. For large dimensions, the fraction of vectors that have only $0, 1$-coefficients is negligible and the above lattice-based approach infeasible. In addition, even the number of such binary vectors can already be exponential[14], so enumerating the whole kernel will not lead to an efficient solution. We also want to mention that using the Gaussian heuristic to approximate $|\ker \boldsymbol{T}|$ does not work.

Directly solving the problem. We will now describe a different approach to compute the number of binary solutions to the system of equations as described above. To algorithmically tackle the problem, we use a mixture of dynamic programming, additional normal forms and tricks for efficient computation of binomial/multinomial coefficients.

(1) We first observe that the problem is recursive, which we depict in Fig. 26. What is shown there is that, for each assignment of balls to a certain column, to find all solutions with this assignment in that specific column one has to find all remaining assignments for the other columns which reduces to solving the exact same problem, but for a smaller number of balls. One can hence precompute a table of solutions for a smaller number of balls and thereby drastically prune the recursion tree. This approach is somewhat similar to a dynamic programming solution (we avoid using actual dynamic programming as such a solver would compute too many configurations that will never be reached).

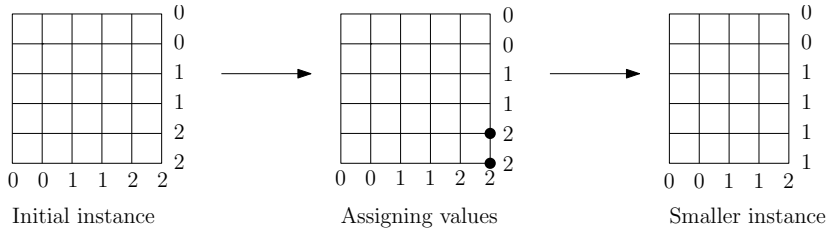

Fig. 26. Recursive elimination

[13] See [25] for a good explanation of the algorithm.
[14] Think about the case where $m = n, r_i = c_i = 1$. Then there are m^n possible solutions.

(2) Moreover, certain assignments of balls to a column can imply solving the same subproblem multiple times, as depicted in Fig. 27. After eliminating the rightmost column, the computational subproblem that has to be solved is the same in all three cases. Hence it is efficient to consider each such assignment only once and multiply it with a correction factor.

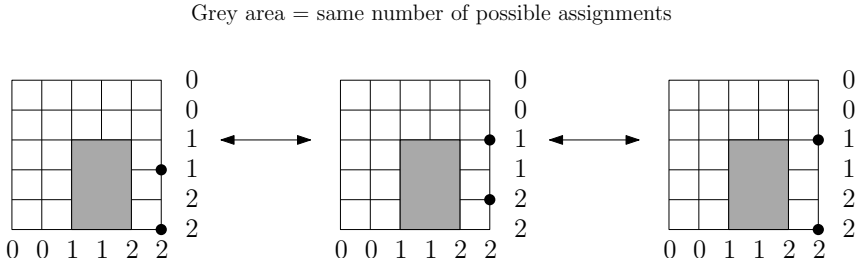

Fig. 27. Equivalent assignments

(3) We need to evaluate a large number of binomial and multinomial coefficients to solve the above problem. To do this efficiently, observe that multiplication and division for large factorial numbers can be done efficiently as follows:
Let $x \in \mathbb{N}$ be the largest value whose factorial we have to compute. Let $p_1^{l_1} \cdot \dots \cdot p_n^{l_n} = x!$ such that $i \neq j \Rightarrow p_i \neq p_j$ and $\forall i \in [n] : \ l_i \geq 1, p_i \in \mathbb{P}$ i.e. be the prime factorization of $x!$. We observe that for each $1 \leq y \leq x$ we can write $y!$ as $y! = p_1^{l'_1} \cdot \dots \cdot p_n^{l'_n}$ for different $l'_1, \dots, l'_n$. In particular, it holds that

$$x! \cdot y! = p_1^{l_1+l'_1} \cdot \dots \cdot p_n^{l_n+l'_n} \text{ and } x!/y! = p_1^{l_1-l'_1} \cdot \dots \cdot p_n^{l_n-l'_n}$$

where all of the exponents are non-negative. Hence in this representation one can efficiently compute multinomial coefficients. Conversion into the above form and back to integers can be efficiently precomputed (as long as an upper bound on x is known).

A.2 Proof of Theorem 2, continued

Let $T > \kappa$. To argue soundness, we have to show that an extractor will fail to obtain all $\boldsymbol{x}_i, \boldsymbol{r}_i$ with probability at most $2^{-\kappa}$. We denote the event where the extractor fails as A_{ExErr}. Moreover, we denote with S_i the event that i *bad* ciphertexts survived the initial cut-and-choose and with C_i that the adversary initially chose i bad ciphertexts. We can write

$$\begin{aligned}
\Pr[\mathsf{A}_{ExErr}] &\leq \max_i \Pr[\mathsf{A}_{ExErr} \mid \mathsf{S}_i, \mathsf{C}_i] \cdot \Pr[\mathsf{S}_i \mid \mathsf{C}_i] \cdot \Pr[\mathsf{C}_i] \\
&= \max_i \Pr[\mathsf{A}_{ExErr} \mid \mathsf{S}_i, \mathsf{C}_i] \cdot 2^{-i} \cdot \Pr[\mathsf{C}_i] \\
&= \max_i \Pr[\mathsf{A}_{ExErr} \mid \mathsf{S}_i] \cdot 2^{-i}
\end{aligned}$$

because $\mathsf{S}_i \supset \mathsf{C}_i$. Since, for $i > \kappa$ we have that $\Pr[\mathsf{A}_{ExErr} | \mathsf{S}_i, \mathsf{C}_i] \cdot 2^{-i} < 2^{-\kappa}$ we can ignore all $i > \kappa$ and simply focus on the case $i \in [1, \kappa]$. In the following, we will describe how to upper-bound $\Pr[\mathsf{A}_{ExErr} \mid \mathsf{S}_i]$ for any such i.

For the extractor not being able to extract a certain plaintext $\boldsymbol{x}_j, \boldsymbol{r}_j$ it is necessary that, for all sums that were opened in Π_{NewZK} and that include $\boldsymbol{c}_j$, either another ciphertext $\boldsymbol{c}_k$ that was not extracted yet is part of the sum or a sum including $\boldsymbol{c}_j, \boldsymbol{a}_k$ was opened where $\boldsymbol{a}_k$ is one of the i non-extractable auxiliary ciphertexts. We denote with $\mathsf{V}_{i,k}$ the event that, in set $V_i \subset T$ there are k *bad* auxiliary ciphertexts. Moreover, let $\mathsf{U}_{1,k}$ be the event that k ciphertexts could not be extracted after Step (4), $\mathsf{U}_{2,k}$ be the event that not all k ciphertexts end up alone in a sum $\boldsymbol{x}', \boldsymbol{r}'$ in Step (5.1) (for all b_2 repetitions) and $\mathsf{F}_{i,k}$ be the event that, in Step (5.1) for

$z = i$, for some of the u sums $\boldsymbol{x}_j$ all the $\boldsymbol{\alpha}_j^1, \ldots, \boldsymbol{\alpha}_j^c$ will be paired up with one of the k bad ciphertexts. We can now model $\Pr[\mathsf{A}_{ExErr} \mid \mathsf{S}_b]$ as

$$\begin{aligned}
\Pr[\mathsf{A}_{ExErr} \mid \mathsf{S}_b] \leq \quad & \Pr\Bigg[\bigcup_{i_1,\cdots_{\aleph}\in\mathbb{N}^+}^{i_1+\cdots+i_{\aleph}=b} \bigcup_{x\in[\kappa]} \Big(\mathsf{V}_{1,i_1}, \ldots, \mathsf{V}_{\aleph,i_{\aleph}}, \mathsf{U}_{1,x}, \\
& \qquad \big(\mathsf{U}_{2,x} \cup \mathsf{F}_{1,i_{b_1+1}} \cup \cdots \cup \mathsf{F}_{b_2,\aleph}\big)\Big)\Bigg] \\
\leq \quad & \sum_{i_1,\cdots_{\aleph}\in\mathbb{N}^+}^{i_1+\cdots+i_{\aleph}=b} \sum_{x\in[\kappa]} \Pr\Bigg[\Big(\mathsf{V}_{1,i_1}, \ldots, \mathsf{V}_{\aleph,i_{\aleph}}, \mathsf{U}_{1,x}, \\
& \qquad \big(\mathsf{U}_{2,x} \cup \mathsf{F}_{1,i_{b_1+1}} \cup \cdots \cup \mathsf{F}_{b_2,\aleph}\big)\Big)\Bigg] \\
\leq \quad & \sum_{i_1,\cdots_{\aleph}\in\mathbb{N}^+}^{i_1+\cdots+i_{\aleph}=b} \sum_{x\in[\kappa]} \Big(\Pr\left[\mathsf{V}_{1,i_1}, \ldots, \mathsf{V}_{\aleph,i_{\aleph}}, \mathsf{U}_{1,x}, \mathsf{U}_{2,x}\right] + \\
& \qquad \Pr\left[\mathsf{V}_{1,i_1}, \ldots, \mathsf{V}_{\aleph,i_{\aleph}}, \mathsf{U}_{1,x}, \big(\mathsf{F}_{1,i_{b_1+1}} \cup \cdots \cup \mathsf{F}_{b_2,\aleph}\big)\right]\Big)
\end{aligned}$$

where $\aleph = b_1 + b_2$. Using the chain rule, we can write

$$\begin{aligned}
\Pr\left[\mathsf{V}_{1,i_1}, \ldots, \mathsf{V}_{\aleph,i_{\aleph}}, \mathsf{U}_{1,x}, \mathsf{U}_{2,x}\right] = \quad & \Pr\left[\mathsf{U}_{2,x} \mid \mathsf{V}_{1,i_1}, \ldots, \mathsf{V}_{\aleph,i_{\aleph}}, \mathsf{U}_{1,x}\right] \cdot \\
& \Pr\left[\mathsf{U}_{1,x} \mid \mathsf{V}_{1,i_1}, \ldots, \mathsf{V}_{\aleph,i_{\aleph}}\right] \cdot \\
& \Pr\left[\mathsf{V}_{\aleph,i_{\aleph}} \mid \mathsf{V}_{1,i_1}, \ldots, \mathsf{V}_{\aleph-1,i_{\aleph-1}}\right] \cdots \Pr\left[\mathsf{V}_{1,i_1}\right]
\end{aligned}$$

An easy calculation using the hypergeometric distribution shows that

$$\begin{aligned}
& \Pr\left[\mathsf{V}_{\aleph,i_{\aleph}} \mid \mathsf{V}_{1,i_1}, \ldots, \mathsf{V}_{\aleph-1,i_{\aleph-1}}\right] \cdots \Pr\left[\mathsf{V}_{1,i_1}\right] = \\
& \qquad \prod_{k\in[\aleph-1]\cup\{0\}} \frac{\binom{b-\sum_{j=0}^{k-1} i_j}{i_k} \cdot \binom{T-\sum_{j=0}^{k-1}|V_j|-(b-\sum_{j=0}^{k-1} i_j)}{|V_k|-i_k}}{\binom{T-\sum_{j=0}^{k-1}|V_j|}{|V_k|}}
\end{aligned}$$

The value $\Pr\left[\mathsf{U}_{1,x} \mid \mathsf{V}_{1,i_1}, \ldots, \mathsf{V}_{\aleph,i_{\aleph}}\right]$ can best be computed by modeling each of the b_1 rounds of Step (4) as a step in a Markov process. That is, for each i_j one can compute the probability distribution that $0, 1, \ldots, i_j$ of the original ciphertexts which have not been extracted yet end up in a sum with an auxiliary ciphertext where the extractor does not have the PRF keys. This can easily be computationally solved. A bound on $\Pr\left[\big(\mathsf{F}_{1,i_{b_1+1}} \cup \cdots \cup \mathsf{F}_{b_2,\aleph}\big) \mid \mathsf{V}_{1,i_1}, \ldots, \mathsf{V}_{\aleph,i_{\aleph}}, \mathsf{U}_{1,x}\right]$ follows from [33] as

$$\Pr\left[\big(\mathsf{F}_{1,i_{b_1+1}} \cup \cdots \cup \mathsf{F}_{b_2,\aleph}\big) \mid \mathsf{V}_{1,i_1}, \ldots, \mathsf{V}_{\aleph,i_{\aleph}}, \mathsf{U}_{1,x}\right] \leq \sum_{j=1}^{b_2} \begin{cases} 0 \text{ if } i_j < c \\ 2 \cdot \sqrt{t} \cdot \left(\frac{i_j}{2\sqrt{t}c}\right)^c \end{cases}$$

Finally, using Problem 1 we observe that

$$\begin{aligned}
& \Pr\left[\mathsf{U}_{2,x} \;\middle|\; \mathsf{V}_{1,i_1}, \ldots, \mathsf{V}_{\aleph,i_{\aleph}}, \mathsf{U}_{1,x}\right] = \\
& \left(1 - \Pr\left[r + s = x \mid \boldsymbol{M} \xleftarrow{\$} \mathcal{M}_{\sqrt{t},\sqrt{t},x} \wedge (r,s) \leftarrow \mathsf{matrixGame}(\boldsymbol{M}, \sqrt{t}, \sqrt{t})\right]\right)^{b_2}
\end{aligned}$$

in the case of strong extraction (for weak extraction, adjust the definition of $\mathsf{matrixGame}$ accordingly). We then solve all of the above quantities computationally to compute $\Pr[\mathsf{A}_{ExErr} \mid \mathsf{S}_b]$.

B Universal Composability - A Short Introduction

For completeness, we include a short introduction into the UC framework [10] which follows the outline of [14]. For all turing machines mentioned here, we assume that λ, κ are an implicit input and polynomial runtime is defined as being polynomial in both of these parameters.

Parties and adversaries. We assume that there are n parties $\mathtt{P}_1, \ldots, \mathtt{P}_n$ that want to participate in a distributed computation, and these parties are probabilistic polynomial time (PPT) interactive Turing Machines (iTMs). An adversary $\mathtt{A}$ is an iTM which gains certain influence over a subset $\widehat{\mathtt{P}} \subset \{\mathtt{P}_1, \ldots, \mathtt{P}_n\}$. The size of this subset and the capabilities that $\mathtt{A}$ has is described by the *adversarial model*. In this work, we consider security against *static*, *active* adversaries that control up to $n-1$ parties.

Static in this context means that, at the beginning of a protocol run, $\mathtt{A}$ defines a set $\widehat{\mathtt{P}}$ of at most $n-1$ parties he intends to corrupt, but he cannot change his choice adaptively throughout the protocol run. This information is not given to the honest parties though, i.e. each $\mathtt{P}_i \notin \widehat{\mathtt{P}}$ does not know which other parties they can trust or not.

Active adversaries have full control over $\widehat{\mathtt{P}}$. They can choose the inputs to the computation, read all information these parties obtain and change messages arbitrarily.

Functionalities. One models the capabilities of a protocol and potential runtime-leakage in a so-called *ideal world* using a *functionality* (functionalities are denoted with $\mathcal{F}$). Such a functionality is a PPT iTM that $\mathtt{P}_1, \ldots, \mathtt{P}_n$ and $\mathtt{A}$ can communicate with. It resembles an *idealized* version of a protocol and specifies the goal in terms of information that $\mathtt{A}$ could obtain, the computation that is done and the values that the honest parties should obtain.

Protocols. In the *real world*, we state the actual protocol as a collection of PPT iTMs $\mathtt{P}_1, \ldots, \mathtt{P}_n$ which communicate according to a given pattern, plus some eventually available resources. A protocol is denoted with a Π or, if it is a small subroutine, as a procedure (no special symbol).

In a protocol it is defined which party at which point sends which message to which other party or resource or which computation it performs locally. Resources (such as e.g. other protocols) are abstractly made available as ideal functionalities of them, which are iTMs themselves. Herein lies the strength of UC – if we prove a protocol to be secure in a hybrid setting (where the resources are ideal functionalities), and prove security for subprotocols implementing these functionalities separately, then the general protocol instantiated with the subprotocols will be secure as well.

Defining security. In order to prove security of a protocol, one has to provide a *simulator* (denoted by $\mathtt{S}$) that will interact with the ideal functionality in the ideal world. This simulator is a PPT iTM interacting with $\mathtt{A}$ and the dishonest parties (this is one fixed $\mathtt{S}$ for each $\mathtt{A}$ that is constrained by a fixed adversarial model). This simulator mimics some $\tilde{\mathtt{P}}_i$ in place of the $\mathtt{P}_i \notin \widehat{\mathtt{P}}$ and functionalities available during runtime. At the same time $\mathtt{S}$ replaces the dishonest parties towards the ideal functionality $\mathcal{F}$ with which the actual honest parties $\mathtt{P}_i$ communicate.

Now let there be a PPT iTM $\mathtt{Z}$ (the *environment*) which provides inputs to all parties $\mathtt{P}_1, \ldots, \mathtt{P}_n$ as well as $\mathtt{A}$ (and obtains output from all of them). For the security game, sample a bit b uniformly at random. If $b = 0$ then one runs an experiment where $\mathtt{Z}$ interacts in the *real world* $(\mathtt{A}, \Pi)$, whereas we let $\mathtt{Z}$ talk in the *ideal world* defined by $(\mathtt{A}, \mathtt{S}, \mathcal{F})$ if $b = 1$. After the execution of either the ideal world or real world setting, $\mathtt{Z}$ outputs a bit b' which is a guess of $\mathtt{Z}$ about the setting $\mathtt{Z}$ currently is in. Depending on the distance of the distributions of the random variables b, b' we call a protocol *computationally secure*(if distinguishing the distribution reduces to solving a problem conjectured to be computationally hard in λ), *statistically secure* (if the distributions are statistically indistinguishable in κ) or *perfectly secure*(if the distributions are identical).