Better Preprocessing for Secure Multiparty Computation

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

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

Abstract. We present techniques and protocols for the preprocessing of secure multiparty computation (MPC), focusing on the so-called SPDZ MPC scheme [19] and its derivatives [16,18,1]. 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 (such as [4,31,27]) 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 (except for [16]). 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 [34], 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 that, 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 ([24,37]) has evolved from a merely academic research topic into a practical technique for secure function evaluation (see e.g. [6]). 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. compute a statistic on the inputs (to securely compute a mean or median) or resemble 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.

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.

A secure multiparty computation protocol replaces such a trusted third party by an interactive protocol among the n parties, while still guaranteeing the above properties. In recent years, it has been shown that

* 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.

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, *raw material* or so-called 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[3]. Unfortunately, the current implementations of the preprocessing has several (non-obvious) drawbacks in terms of efficiency which we try to address in this work:

– The complexity of the preprocessing phase depends upon the size of the field over which the function f will be evaluated, such that it is much less efficient for small fields. This is because SHE schemes have no efficient reliable distributed decryption algorithm, so 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, but this approach only yields security inversely proportional to the field size. Hence, especially for small files, one has to repeat that procedure multiple times which introduces notable overhead.
– 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.
– If the goal in the end is to do secure computation over the integers, one needs to use large fields or rings to avoid overflow. Unfortunately, the parameter sizes of SHE schemes grow very quickly if one increases the size of the underlying field, rendering them very slow in practice. This makes it interesting to investigate a preprocessing scheme using Paillier encryption, which comes with a very large ring as plaintext space.

1.1 Contributions and Technical Overview

In this work, we address the aforementioned problems and show the following results:

(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 $\Theta(1-1/k)$ for error probability 2^{-k}.
(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 of Paillier and Damgård-Jurik [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 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, this did not appear in previous published work.

We will explain our contributions and techniques in more detail now.

Verifying Multiplicative Relations. Our goal is (somewhat simplified) to produce encrypted vectors $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}$ such that $\boldsymbol{x} \odot \boldsymbol{y} = \boldsymbol{z}$, where $\odot$ denotes the coordinate-wise product, or Schur product. The SPDZ protocol for creating such data uses distributed decryption during which errors may be introduced. To counter

[3] For more details, see Section 2.2.

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. Those vectors can be put into SIMD ciphertexts of the SPDZ preprocessing scheme. Note that multiplying $\boldsymbol{x}$ and $\boldsymbol{y}$ coordinate-wise yields a codeword in a related code (namely its so-called Schur transform). Now we do a protocol to obtain an encryption of $\boldsymbol{z}$, which, however, uses unreliable decryption underway. The next step is then to check if $\boldsymbol{z}$ is indeed codeword as expected, This can be done almost only by linear operations - which are basically *free* in the SPDZ MPC scheme, because they can all be done as local operations and do not involve sending messages.
Checking whether the result is a codeword is not sufficient, but we can note that if $\boldsymbol{z}$ is a codeword 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 $\boldsymbol{y}$. On a high level, this works because the adversary must have altered the resulting codeword in a number of positions that is at least the minimum distance of the code. During each check we have a constant probability of catching the adversary, and this quickly amplifies to our desired security levels.
Note that 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 with p elements for a prime p. Now, the basic step in the protocol is that a player $\mathtt{PR}$ 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 $\mathtt{PR}$ is honest 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 $\mathtt{PR}$ is corrupt, we cannot guarantee that the message m known to $\mathtt{PR}$ will be in this interval. The best we can do is to force $\mathtt{PR}$ 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 ciphertext are honestly generated, it will ensure 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 linearly homomorphic, so does not allow to perform multiplications of the plaintexts of two or more ciphertexts directly. On the other hand, it has a 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 works as follows: Assume $\mathtt{P}_1$ published some encryption $[a]$, $\mathtt{P}_2$ published $[b]$ and they want to compute values c_1, c_2 where $\mathtt{P}_1$ holds c_1 and $\mathtt{P}_2$ c_2 such that $a \cdot b = c_1 + c_2$.
In a protocol, $\mathtt{P}_2$ would send an encryption $[c_1] := b' \cdot [a] + [-c_2]$ to $\mathtt{P}_1$ and prove (among other things) that this b' is the same as the plaintext inside $[b]$. Afterwards, both use the distributed decryption towards $\mathtt{P}_1$

so that she can obtain c_1. Both parties' shares do now individually not reveal any information about the product.

Our approach is, instead of sampling all a, b independently, to let the factors be evaluations of a polynomial (that is implicitly defined), and then multiply these factors *unreliably*: Instead of giving a zero-knowledge proof that $b' = b$, we only need to prove that $\mathtt{P}_2$ knows b', c_2 such that the above equation is satisfied, which reduces the complexity of the proof.

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. It is an interesting open question how these approaches compare in practice.

1.2 Related Work

In an 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 particular instance of the scheme that we use is from [15] and allows for a simpler presentation.

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 *sec* times (for *sec* being the statistical security parameter) to achieve soundness error negligible in *sec*. Additional care must also be taken to prevent leakage of the plaintext, which normally incurs a 2^{sec} 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 *sec* plaintexts in parallel using $O(sec)$ auxiliary ciphertexts. As a drawback, the technique introduces an additional $2^{O(sec)}$ 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.[4] A different approach was taken by Ling et al. in [28]. They use a technique due to Stern and achieve knowledge error $2/3$.

Notation	What it means	Notation	What it means
W	Ring dimension of SHE scheme	$\mathtt{P}_i$	Player i
L	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	$\mathtt{P}_{Bad}$	Set of bad parties
B_P, B_R	Bound proven by ZK Proofs	$\mathtt{PR}$	Prover
$\mathrm{Enc}^S_{pk}, \mathrm{Dec}^S_{sk}$	Encryption/Decryption SHE	$\mathtt{C}$	Challenger
λ, sec	Computational/statistical security parameter	$\mathtt{VE}$	Verifier
N	Paillier modulus	α	MAC key for SPDZ sharing
$\mathrm{Enc}^P_{pk}(\cdot,\cdot), [\cdot], \mathrm{Dec}^P_{sk}(\cdot)$	Encryption/Decryption Paillier	$\langle\cdot\rangle$	Value secret shared as in SPDZ

Table 1. List of variables

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. Some protocols require commitments, which we abstract as the functionality $\mathcal{F}_{\text{COMMIT}}$. In addition, in many cases we need a random oracle (that we e.g. use for coin-flipping as in $\mathcal{P}_{\text{PROVIDERANDOM}}$[5]). We use $\odot$ for point-wise multiplication of vector entries and $(g, h) = d$ to denote that d is the greatest common divisor of g, h. Moreover, let $[r]_1 = \{1, ..., r\}$ and $[r]_0 = \{0, ..., r\}$.
Certain notation will occur repeatedly throughout the paper. To ease readability we provide a list of some special notation that we use and what they stand for in Table 1.

Functionality $\mathcal{F}_{\text{COMMIT}}$

Assume that $\mathbb{Z}_p$ is fixed.

Commit:
- On input $(\mathsf{commit}, v, r, i, j)$ by $\mathtt{P}_i$, where $v, r \in \mathbb{Z}_p \cup \{\perp\}$ and j is a unique identifier, it stores (v, r, i, j) internally and outputs (i, τ_v) to all players.

Open:
- On input (open, i, j) by $\mathtt{P}_i$, the ideal functionality outputs (v, r, i, j) to all players if there is a (v, r, i, j) stored internally.
- If $(\mathsf{no\_open}, i, j)$ is given by the adversary, and $\mathtt{P}_i$ is corrupt, the functionality outputs $(\perp, \perp, i, j)$ to all players.

Fig. 1. The Functionality for Commitments

2.1 The SPDZ Multiparty Computation Protocol

We start out with a short primer on the [19] MPC protocol which we will mostly refer to as SPDZ. Each value $c \in \mathbb{Z}_p$ of the computation is first 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 also sum-shared among the parties, where party $\mathtt{P}_i$ holds share α_i such that $\alpha = \sum_{i=1}^{n} \alpha_i$.
To make the above more formal, we define the $\langle\cdot\rangle$-representation of a shared value as follows:

[4] 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 $\mathbb{Z}$.

[5] In practice, this functionality can be implemented in several ways, e.g. using a pseudorandom function and the commitment scheme $\mathcal{F}_{\text{COMMIT}}$.

Procedure $\mathcal{P}_{\text{ProvideRandom}}$

Even though we do not mention minimum lengths of seeds here, they should be chosen according to a concrete security parameter. Let $\mathcal{U}_{\boldsymbol{s}}(\cdot,\cdot)$ be a random oracle that for parameters q, l outputs a uniformly random string $\boldsymbol{v} \in \mathbb{Z}_q^l$.

ProvideRandom(q, l) :

(1) Each party $\mathtt{P}_i$ commits to a seed $\boldsymbol{s}_i \in \{0,1\}^*$ using $\mathcal{F}_{\text{Commit}}$.
(2) Each party opens its commitment to all parties.
(3) Each party locally computes $\boldsymbol{s} = \boldsymbol{s}_1 \oplus \cdots \oplus \boldsymbol{s}_n$
(4) Each party outputs $\boldsymbol{v} \leftarrow \mathcal{U}_{\boldsymbol{s}}(q, l)$.

Fig. 2. A protocol to jointly generate random values

Definition 1. *Let $r, s, e \in \mathbb{Z}_p$, then the $\langle r\rangle$-representation of r is defined as*

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

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. Moreover, we define

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

This representation is closed under linear operations:

Remark 1. Let $r, s, e \in \mathbb{Z}_p$. 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

$$\langle r\rangle + \langle s\rangle \hat{=} \langle r + s\rangle,\ e \cdot \langle r\rangle \hat{=} \langle e \cdot r\rangle,\ e + \langle r\rangle \hat{=} \langle e + r\rangle$$

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 Figure 3. Correctness and privacy of this procedure were established before, e.g. in [19]. This

Procedure $\mathcal{P}_{\text{Mult}}$

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. Protocol to generate the product of two $\langle\cdot\rangle$-shared values

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 $\mathcal{P}_{\text{CheckMac}}$ which allows to check the MAC of the opened value without revealing the key α. 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 $\mathbb{Z}_N$ where $q|N$ is the smallest prime factor. This in essence is captured by the following Lemma which we will also need in other cases:

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

[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 $(x - r) + \langle r\rangle = \langle x\rangle$.

```
Procedure $\mathcal{P}_{\text{CheckMac}}$

CheckOutput$(v_1, ..., v_t, m)$ Here we check whether the MACs hold on $t$ partially opened values.
    (1) The parties compute $\boldsymbol{r} \leftarrow \mathcal{P}_{\text{ProvideRandom}}.\text{ProvideRandom}(m, t)$.
    (2) Each party computes $v = \sum_{i=1}^{t} \boldsymbol{r}[i] \cdot v_i$.
    (3) Each $\mathtt{P}_i$ computes $\gamma_i = \sum_{j=1}^{t} \boldsymbol{r}[j] \cdot \gamma(v_j)$ and $\sigma_i = \gamma_i - \alpha_i \cdot v$.
    (4) Each $\mathtt{P}_i$ commits to $\sigma_i$ using $\mathcal{F}_{\text{Commit}}$ as $c_i'$.
    (5) Each $c_i'$ is opened towards all players using $\mathcal{F}_{\text{Commit}}$.
    (6) If $\sigma = \sum_{i=1}^{n} \sigma_i$ is 0 then return 1, otherwise return 0.
```

Fig. 4. Procedure to check validity of MACs

Lemma 1. *Assume that $\mathcal{P}_{\text{CheckMac}}$ is executed over the field $\mathbb{Z}_p$. The protocol $\mathcal{P}_{\text{CheckMac}}$ is correct and sound:*

- *It returns 1 if all the values v_i and their corresponding MACs $\gamma(v_i)$ are correctly computed.*
- *It rejects except with probability $2/p$ in the case where at least one value or MAC is not correctly computed.*

Its proof is straightforward and can be found e.g. in [16]. For some of our settings we will choose p to be rather small (i.e. of constant size in the security parameter). In this case, one can extend the $\langle\cdot\rangle$−representation as in Definition 1 by having a larger number of MACs and then check all of these MACs in parallel.

2.2 Somewhat Homomorphic Encryption

In the aforementioned MPC scheme, we left open how to compute the random triples $(\langle a\rangle, \langle b\rangle, \langle c\rangle)$ necessary for the multiplication protocol. In [19,16] this is achieved using *Somewhat Homomorphic Encryption*(SHE). In the following, we will give a quick definition of what we mean by an SHE scheme.
Let $\mathcal{M} = \mathbb{Z}_p^l$ be the direct product of l $\mathbb{Z}_p$-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 $\oplus$ and an operation $\otimes$ 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

$$\text{encode} : \mathcal{M} \to \mathcal{A} \text{ and } \text{decode} : \mathcal{A} \to \mathcal{M}$$

where encode is injective. The ring operations from $\mathcal{M}$ must carry over to a certain degree, which is formalized as follows:

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

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^L, which outputs vectors $\boldsymbol{r} \in \mathbb{Z}^L$ such that $Pr[||\boldsymbol{r}||_\infty \geq \rho \mid \boldsymbol{r} \leftarrow \mathrm{D}_\rho^L] < negl(\lambda)$. We let C_{irc} be the set of arithmetic *Single Instruction Multiple Data* (SIMD) circuits over $\mathbb{Z}_p^l$. The SIMD property implies that there exists a function $f \in \mathbb{Z}_p[X_1, ..., X_{n(f)}]$ such that $\widehat{f} \in C_{irc}$ evaluates the function f l times on inputs in $\mathbb{Z}_p^{n(f)}$ in parallel.
For the specified algebraic structures, define the algorithms $\text{KeyGen}^S, \text{Enc}^S, \text{Dec}^S$ that represent the cryptosystem. The algorithms are probabilistic polynomial time algorithms.

$\mathrm{KeyGen}^S()$: This algorithm samples a public-key/private-key pair (pk, sk).

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

$$\mathrm{Enc}^S_{pk}(\boldsymbol{x}_1 + ... + \boldsymbol{x}_\kappa, \boldsymbol{r}_1 + ... + \boldsymbol{r}_\kappa) = \mathrm{Enc}^S_{pk}(\boldsymbol{x}_1, \boldsymbol{r}_1) \oplus ... \oplus \mathrm{Enc}^S_{pk}(\boldsymbol{x}_\kappa, \boldsymbol{r}_\kappa)$$

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

Correctness Let $n(f), f \in C_{irc}$ be the number of input values of f and let $\widehat{f}$ be the embedding of f into $\mathcal{B}$ where '+' is replaced by $\oplus$, '$\cdot$' by $\otimes$ and the constant $c \in \mathbb{Z}_p$ by $\mathrm{Enc}^S_{pk}(\mathrm{encode}(c \cdot \mathbf{1}), \mathbf{0})$. For data vectors $\boldsymbol{x}_1, ..., \boldsymbol{x}_{n(f)}$, let $f(\boldsymbol{x}_1, ..., \boldsymbol{x}_{n(f)})$ be the SIMD application of f to this data.

Definition 2 (Correctness). *The aforementioned algorithms* $(\mathrm{KeyGen}^S, \mathrm{Enc}^S, \mathrm{Dec}^S)$ *are called* (B_P, B_R, C_{irc})*-correct if*

$$Pr\Big[\ \mathrm{Dec}^S_{sk}(\boldsymbol{c}) \neq f\big(\mathrm{decode}(\boldsymbol{x}_1), ..., \mathrm{decode}(\boldsymbol{x}_{n(f)})\big)\ \Big|\ (pk, sk) \leftarrow \mathrm{KeyGen}^S() \wedge f \in C_{irc} \wedge$$
$$(\boldsymbol{x}_1, ..., \boldsymbol{x}_{n(f)}, \boldsymbol{r}_1, ..., \boldsymbol{r}_{n(f)}) \in \mathcal{A}^{n(f)} \times (\mathbb{Z}^L)^{n(f)} \wedge \boldsymbol{c} \leftarrow \widehat{f}(\boldsymbol{c}_1, ..., \boldsymbol{c}_{n(f)}) \wedge$$
$$\big(\mathrm{decode}(\boldsymbol{x}_i) \in \mathcal{M} \wedge ||\boldsymbol{x}_i||_\infty \leq B_P \wedge ||\boldsymbol{r}_i||_\infty \leq B_R \wedge \boldsymbol{c}_i \leftarrow \mathrm{Enc}^S_{pk}(\boldsymbol{x}_i, \boldsymbol{r}_i)\big)_{i \in [n(f)]_1}\Big] \leq \mathit{negl}(\lambda)$$

For SPDZ one would additionally require that the encryption scheme has a *distributed decryption* and *distributed 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_{irc} *contain formulas of the form* $\big(\sum_{i=1}^n x_i\big) \cdot \big(\sum_{i=1}^n y_i\big) + \sum_{i=1}^n z_i$ *for some* $n \in \mathbb{N}^+$. *Let* $K = (\mathrm{KeyGen}^S, \mathrm{Enc}^S, \mathrm{Dec}^S)$ *be an IND-CPA secure cryptosystem.* K *is called somewhat homomorphic if it is* (B_P, B_R, C_{irc})*-correct for some 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 zero-knowledge proofs were used before in the context of the above cryptosystem and preprocessing for MPC. The first technique was described in [19] and follows a Σ-protocol. While the challenge space is very small, an amortization technique due to [11] can be applied. Another approach was introduced in [16] and uses a LEGO-like argument.
The proofs will have statistical security parameter *sec*. We prove plaintext knowledge of the set $\mathcal{S}_P = \{\boldsymbol{c}_1, ..., \boldsymbol{c}_t\}$ of ciphertexts. More formally, one shows that the following relation holds:

$$R_{POPK} = \Big\{(\boldsymbol{a}, \boldsymbol{w})\ |\ \boldsymbol{a} = (\boldsymbol{c}_1, ..., \boldsymbol{c}_t, pk) \wedge \boldsymbol{w} = (\boldsymbol{x}_1, \boldsymbol{r}_1, ..., \boldsymbol{x}_t, \boldsymbol{r}_t) \wedge$$
$$\big[\boldsymbol{c}_i = \mathrm{Enc}^S_{pk}(\boldsymbol{x}_i, \boldsymbol{r}_i) \wedge \mathrm{decode}(\boldsymbol{x}_i) \in \mathcal{M} \wedge$$
$$||\boldsymbol{x}_i||_\infty \leq B_P \wedge ||\boldsymbol{r}_i||_\infty \leq B_R\big]_{i \in [t]_1}\Big\}$$

```
Procedure $\mathcal{P}_{CutAndChoose}$

CutAndChoose$(T, B_p, B_r)$:
  (1) PR calls the RO with seeds $f_1, ..., f_{2T}$. From the output, it generates an $\boldsymbol{s}_i, \boldsymbol{y}_i$ 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 $\boldsymbol{y}_i = \text{encode}(\boldsymbol{m}_i) + \boldsymbol{u}_i$ 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 $i = 1, ..., 2T$ PR computes $\boldsymbol{a}_i = \text{Enc}^S_{pk}(\boldsymbol{y}_i, \boldsymbol{s}_i)$ and sends $\boldsymbol{a}_1, ..., \boldsymbol{a}_{2T}$ to VE.
  (3) PR, VE sample $V$ as $V \leftarrow \mathcal{P}_{\text{ProvideRandom}}.\text{ProvideRandom}(2T, T)$ such that $|V| = T$.
  (4) For all $i \in V$, PR sends $f_i$ to 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 VE aborts.
  (5) Let $[2T]_1 \setminus V = \{i_1, ..., i_T\}$. Output $(\boldsymbol{a}_{i_1}, \boldsymbol{y}_{i_1}, \boldsymbol{s}_{i_1}, ..., \boldsymbol{a}_{i_T}, \boldsymbol{y}_{i_T}, \boldsymbol{s}_{i_T})$.
```

Fig. 5. Procedure for generating auxiliary ciphertexts using cut and choose

SPDZ1-like proofs An instance of the proof takes as input $t = sec$ ciphertexts and proves knowledge of $2 \cdot sec - 1$ linear combinations of a certain form (there are 2^{sec} matrices depicting the linear transformation). Each such individual proof is done using a Σ protocol. This gives an overhead of 2 auxiliary ciphertexts per proven ciphertext. On the downside, the soundness argument works by exploiting that the difference of two differing matrices describing the linear combinations is invertible, but the inverse may have very large coefficients - in the worst case exponentially in *sec*. This means that B_P, B_R are rather large.

SPDZ2-like proofs Another approach, that is reminiscent of the technique from [32] uses a larger number of auxiliary ciphertexts, but yields provably lower parameters. 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 $\mathcal{P}_{CutAndChoose}$, where first $2T$ auxiliary ciphertexts are generated by PR, among which T are opened. These T opened ciphertexts are chosen uniformly at random and 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. 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. PR in turn, for bucket i and $\boldsymbol{c}_i \in \mathcal{S}_P$, opens the sum $\boldsymbol{c}_i + \boldsymbol{a}_i$ and VE checks correctness. This proof yields drastically lower bounds on B_P, B_R than the one presented before, but unfortunately requires to generate $2T/t$ auxiliary ciphertext per proven plaintext. In particular for a *practical* number of ciphertexts (think $t \approx 40$) this large number makes implementations quite slow (due to the demand in RAM).

2.4 (Reed Solomon) Codes

Let $q, m, k \in \mathbb{N}^+, m > k$ and q be a prime power. Consider the two vector spaces $\mathcal{K} = \mathbb{F}_q^k, \mathcal{D} = \mathbb{F}_q^m$ and a monomorphism $C : \mathcal{K} \to \mathcal{D}$ together as a *code*, i.e. $\boldsymbol{c} = C(\boldsymbol{x})$ as an encoding of $\boldsymbol{x}$ in $\mathcal{D}$. In addition, it should be efficiently decidable whether $\boldsymbol{c}' \in C$ (error checking), where $\boldsymbol{c}' \in C \Leftrightarrow \exists \boldsymbol{x}' \in \mathcal{K} : \; C(\boldsymbol{x}') = \boldsymbol{c}'$ and the minimum 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 nonzero in as many positions as possible). Such a code is called an $[m, k, d]$ code. If, for every message $\boldsymbol{x} \in \mathcal{K}$, this $\boldsymbol{x}$ reappears directly in $C(\boldsymbol{x})$ then the code is called systematic. Without loss of generality, one can assume that the first k 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 $\boldsymbol{G}$ (the *generator matrix*), and we can write $C : \boldsymbol{x} \mapsto \boldsymbol{G}\boldsymbol{x}$ where $\boldsymbol{G} \in \mathbb{F}_q^{m \times k}$. Similarly, we assume the existence of a *check matrix* $\boldsymbol{H} \in \mathbb{F}_q^{(m-k) \times m}$ where $\boldsymbol{H}\boldsymbol{x} = \boldsymbol{0} \Leftrightarrow \boldsymbol{x} \in C$.
For a $[m, k, d]$ code C, define the *Schur transform* $C^* = span(\{\boldsymbol{x} \odot \boldsymbol{y} \mid \boldsymbol{x}, \boldsymbol{y} \in C\})$. C^* is itself a code where the message length $k' \geq k$ 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.
A code with small loss $d - d'$ with respect to the Schur transform (as we shall see later) is the so-called *Reed-Solomon* code ([35]), where the encoding C works as follows: Fix pairwise distinct and nonzero $z_1, ..., z_m \in \mathbb{F}_q$

and define the matrices $\boldsymbol{A_1} = V(z_1, ..., z_k)^{-1}$ and $\boldsymbol{A_2} = V(z_1, ..., z_m)$ where $V(\cdot)$ is the Vandermonde matrix. Consider the mapping

$$\begin{aligned} C : \mathcal{K} &\to \mathcal{D} \\ \boldsymbol{x} &\mapsto \boldsymbol{A_2 A_1 x} \end{aligned}$$

as the encoding procedure. The encoding can be made efficient since the matrices are decomposable for certain values $z_1, ..., z_m$ using the *Fast Fourier Transform* (FFT). The decoding works the same way, where one computes $\boldsymbol{y}^\top \boldsymbol{A_2}^{-1} \boldsymbol{A_1}^{-1}$.
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 $2(k-1)$. Hence the code C^* has minimum distance $d' = m - 2k + 1$.

2.5 The Paillier Cryptosystem

We use the Paillier encryption scheme as defined in [34,15] (with some practical restrictions). Let $N = p \cdot q$ be the product of two odd, κ-bit safe-primes with $(N, \phi(N)) = 1$ (we choose κ such that the scheme has λ bit security). Paillier encryption of a message $m \in \mathbb{Z}/N\mathbb{Z}$ with randomness $r \in \mathbb{Z}/N\mathbb{Z}^*$ is defined as:

$$\mathrm{Enc}^P_{pk}(m, r) := r^N \cdot (N+1)^m \bmod N^2$$

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 then proceeds as

$$m = ((c \cdot r^{-N} \mod N^2) - 1)/N \mod N$$

The KeyGen^P algorithm samples an RSA modulus $N = p \cdot q$, and we let $pk = (N)$ and $sk = (p, q, f = N^{-1} \bmod \varphi(N))$. The encryption scheme is additively homomorphic and IND-CPA secure given the Composite Residuosity problem $CR[N]$ is hard.
During the decryption of a ciphertext as described above one does completely recover the randomness used during encryption. This gives rise to a reliable distributed decryption algorithm, since the the encryption function is a bijection. To obtain such a distributed decryption, one can sum-share the value $N^{-1} \bmod \varphi(N)$: For n parties $\mathtt{P}_1, ..., \mathtt{P}_n$, assume that there exists a sum sharing $f_1, ..., f_n$ such that $f_2, ..., f_n \leftarrow \mathbb{Z}/2^{sec}N\mathbb{Z}$ chosen i.i.d. and $f_1 \in \mathbb{Z}/2^{sec}N\mathbb{Z}$ such that $f_1 = f - (\sum_{i=2}^n f_i) \bmod \varphi(N)$ for some statistical security parameter $sec \in \mathbb{N}$. Each party can then broadcast $c^{f_i} \bmod N$. This does not leak information about $\varphi(N)$ and also not about the shares f_i as long as the ciphertexts c are randomly chosen. We do not provide the actual protocol in detail, but just refer to the functionality Figure 6 which will allow distributed decryption and key generation for it.

3 More Efficient Preprocessing from Somewhat Homomorphic Encryption

In this section, we present an improved preprocessing for the [19] protocol. In a first step, we overhaul the triple generation in a way that allows more efficient checks of correctness. Moreover, we present a different proof of plaintext knowledge that is honest-verifier zero-knowledge. This is sufficient to get UC security for the overall protocol, even though we consider malicious corruption. This is explained in detail in [19] where they show similar properties for their proofs of plaintext knowledge. The idea is to get the challenge from a secure coin-flip protocol or use the Fiat-Shamir transform ([20]).

Functionality $\mathcal{F}_{\text{KGD,Paillier}}$

Generate key:

(1) On input $(\mathsf{generate\_key}, \kappa, sec)$ by all parties, randomly sample two different primes $p, q \in \mathbb{P}$ of bit length approximately κ. Let $N = p \cdot q$ and compute $f = N^{-1} \bmod \varphi(N)$.

(2) Sample uniformly random $f_2, ..., f_n \in \mathbb{Z}/2^{sec}N\mathbb{Z}$ and choose $f_1 \in \mathbb{Z}/2^{sec}N\mathbb{N}$ such that $f = \sum_i f_i \bmod \varphi(N)$.

(3) Output (N, f_i) to party $\mathtt{P}_i$.

Distributed decryption:

(1) When receiving $(\mathsf{start\_distributed\_decryption})$ from all players, check whether there exists a shared key pair (N, f). If not, return $\perp$.

(2) Upon receiving $(\mathsf{decrypt}, c)$ from all honest players, send c and $m \leftarrow \mathrm{Dec}^{P}_{sk}(c)$ to the adversary. Upon receiving $m' \in \{m, \perp\}$ from the adversary, send (result, m') to all players.

Fig. 6. Functionality that provides shared keys and decrypt ciphertexts

3.1 Generating Triples with Lower Overhead

Let C be some $[m, k, d]$ Reed-Solomon code as described in the previous section. Moreover, let C^* be its $[m, k', d']$ Schur transform. We assume that there exists a protocol to generate triples where the results are correct if all parties follow the protocol, but triples might be not correct if some party is not behaving honestly.

Offline Phase Protocol We assume the existence of a functionality that samples *faulty correlated randomness* and which is depicted in Figure 7. 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.

Functionality $\mathcal{F}_{\text{TripleGen}}$

This functionality generates a shared MAC key α and $\langle\cdot\rangle$-representations, where the latter ones can be biased by the adversary. It uses the procedure Angle as depicted in $\mathcal{P}_{\text{Angle}}$.

Initialize: On input (init, p, C) from all players, the functionality stores the prime p and the code C. $\mathtt{A}$ chooses the set of parties $\mathtt{P}_{Bad} \subset \{1, \ldots, n\}$ he corrupts.

(1) For all $i \in \mathtt{P}_{Bad}$, $\mathtt{A}$ inputs $\alpha_i \in \mathbb{Z}_p$, while for all $i \notin \mathtt{P}_{Bad}$, the functionality chooses $\alpha_i \leftarrow \mathbb{Z}_p$ at random.

(2) Set they key $\alpha = \sum_{i=1}^{n} \alpha_i$ and send α_i to $\mathtt{P}_i, i \notin \mathtt{P}_{Bad}$.

Triples: On input $(\mathsf{triples})$ from all parties, the functionality does the following to generate triples:

(1) For $i \notin \mathtt{P}_{Bad}$, the functionality samples $\boldsymbol{a}_i, \boldsymbol{b}_i \in C$ at random.

(2) For $i \in \mathtt{P}_{Bad}$, $\mathtt{A}$ inputs $\boldsymbol{a}_i, \boldsymbol{b}_i, \boldsymbol{c}_i, \boldsymbol{\delta}, \boldsymbol{\Delta}_{\gamma,a}, \boldsymbol{\Delta}_{\gamma,b}, \boldsymbol{\Delta}_{\gamma,c} \in \mathbb{Z}_p^m$. 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 $j \notin \mathtt{P}_{Bad}$ be the smallest index of an honest player. For all $i \notin \mathtt{P}_{Bad}, i \neq j$ choose $\boldsymbol{c}_i \in \mathbb{Z}_p^m$ uniformly at random. For $\mathtt{P}_j$ let $\boldsymbol{c}_j = \boldsymbol{a} \odot \boldsymbol{b} + \boldsymbol{\delta} - \sum_{i \in [n]_1, i \neq j} \boldsymbol{c}_i$. Send $\boldsymbol{a}_i, \boldsymbol{b}_i, \boldsymbol{c}_i$ to each honest $\mathtt{P}_i$.

(5) Run the macros
$\langle \boldsymbol{a} \rangle \leftarrow \mathrm{Angle}(\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n, \alpha, \boldsymbol{\Delta}_{\gamma,a}, m)$,
$\langle \boldsymbol{b} \rangle \leftarrow \mathrm{Angle}(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n, \alpha, \boldsymbol{\Delta}_{\gamma,b}, m)$,
$\langle \boldsymbol{c} \rangle \leftarrow \mathrm{Angle}(\boldsymbol{c}_1, \ldots, \boldsymbol{c}_n, \alpha, \boldsymbol{\Delta}_{\gamma,c}, m)$.

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

Fig. 7. Functionality that generates triples

Procedure $\mathcal{P}_{\text{ANGLE}}$

We let R be a ring.

Angle$(\boldsymbol{r}_1, \dots, \boldsymbol{r}_n, \alpha, \boldsymbol{\Delta}_\gamma, m, R)$: This procedure will be run by the functionality to create $\langle\cdot\rangle$-representations.

(1) Define $\boldsymbol{r} = \sum_{i=1}^n \boldsymbol{r}_i$

(2) For $i \in \mathtt{P}_{Bad}$, $\mathtt{A}$ inputs $\boldsymbol{\gamma}_i \in R^m$, and for $i \notin \mathtt{P}_{Bad}$, choose $\boldsymbol{\gamma}_i \xleftarrow{\$} R^m$ at random except for $\boldsymbol{\gamma}_j$, with j being the smallest index not in $\mathtt{P}_{Bad}$.

(3) Set $\boldsymbol{\gamma} = \alpha \cdot \boldsymbol{r} + \boldsymbol{\Delta}_\gamma$ and $\boldsymbol{\gamma}_j = \boldsymbol{\gamma} - \sum_{j \neq i=1}^n \boldsymbol{\gamma}_i$. For every honest party $\mathtt{P}_i$, send γ_i.

(4) Define $\langle \boldsymbol{r} \rangle = (\boldsymbol{r}_1, ..., \boldsymbol{r}_n, \boldsymbol{\gamma}_1, ..., \boldsymbol{\gamma}_n)$. Return $\langle \boldsymbol{r} \rangle$.

Fig. 8. Procedure to generate shared MACs on values

Figure 7 can be implemented using a SHE scheme as we defined in Definition 3 which 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, our goal is to show that one can implement $\mathcal{F}_{\text{FULLTRIPLEGEN}}$ as depicted in Figure 9. It is related to $\mathcal{F}_{\text{TRIPLEGEN}}$ but does ensure that all multiplication triples are correct.

Functionality $\mathcal{F}_{\text{FULLTRIPLEGEN}}$

Let $\mathtt{P}_{Bad}$ be the set of parties that are controlled by $\mathtt{A}$ and $u \in \mathbb{N}^+$. This functionality generates a shared MAC key α and $\langle\cdot\rangle$-representations. It uses the macro Angle as depicted in $\mathcal{P}_{\text{ANGLE}}$. Observe that it can be initialized for arbitrary rings R that allow efficient sampling.

Initialize: On input (init, R, u) from all players, the functionality stores the prime p and the vector dimension u. $\mathtt{A}$ chooses the set of parties $\mathtt{P}_{Bad} \subset \{1, \dots, n\}$ he corrupts.

(1) Store the dimension u and description of the ring R.

(2) For all $i \in \mathtt{P}_{Bad}$, $\mathtt{A}$ inputs $\alpha_i \in R$, while for all $i \notin \mathtt{P}_{Bad}$, the functionality chooses $\alpha_i \leftarrow R$ at random.

(3) Set they key $\alpha = \sum_{i=1}^n \alpha_i$ and send α_i to $\mathtt{P}_i, i \notin \mathtt{P}_{Bad}$.

Triples: On input $(\mathsf{triples})$ the functionality does the following

(1) Let $\mathtt{A}$ input $\boldsymbol{a}_i, \boldsymbol{b}_i, \boldsymbol{c}_i, \boldsymbol{\Delta}_{\gamma,a}, \boldsymbol{\Delta}_{\gamma,b}, \boldsymbol{\Delta}_{\gamma,c} \in R^u$ for each $i \in \mathtt{P}_{Bad}$.

(2) Choose $\boldsymbol{a}_i, \boldsymbol{b}_i \in R^u$ for each honest $\mathtt{P}_i$ uniformly at random. Set $\boldsymbol{a} = \sum_i \boldsymbol{a}_i, \boldsymbol{b} = \sum_i \boldsymbol{b}_i$ and define $\boldsymbol{c} = \boldsymbol{a} \odot \boldsymbol{b}$.

(3) Let j be the smallest number in $[n]_1 \backslash \mathtt{P}_{Bad}$. Choose uniformly random $\boldsymbol{c}_i \in R^u$ for each $\mathtt{P}_i$ with $i \in [n] \backslash \mathtt{P}_{Bad}, i \neq j$ and set $\boldsymbol{c}_j = \boldsymbol{c} - \sum_{i \in [n]_1, i \neq j} \boldsymbol{c}_i$.

(4) Run the macros
$\langle \boldsymbol{a} \rangle \leftarrow \text{Angle}(\boldsymbol{a}_1, \dots, \boldsymbol{a}_n, \alpha, \boldsymbol{\Delta}_{\gamma,a}, u, R)$,
$\langle \boldsymbol{b} \rangle \leftarrow \text{Angle}(\boldsymbol{b}_1, \dots, \boldsymbol{b}_n, \alpha, \boldsymbol{\Delta}_{\gamma,b}, u, R)$,
$\langle \boldsymbol{c} \rangle \leftarrow \text{Angle}(\boldsymbol{c}_1, \dots, \boldsymbol{c}_n, \alpha, \boldsymbol{\Delta}_{\gamma,c}, u, R)$.

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

Fig. 9. Functionality that generates triples

We will now show how to implement $\mathcal{F}_{\text{FULLTRIPLEGEN}}$ in the $\mathcal{F}_{\text{TRIPLEGEN}}$-hybrid model using the protocol $\Pi_{\text{TRIPLECHECK}}$. It implements the idea that was already sketched 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 $\boldsymbol{0}$ 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.

Proof of Security To prove the security of our construction, we use the following fact:

Protocol $\Pi_{\text{TripleCheck}}$

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, and are over the field $\mathbb{Z}_p$.

Initialize:

(1) All parties send (init, p, C) to $\mathcal{F}_{\text{TripleGen}}$ to receive their shares α_i of α.

Triples:

(1) All parties send $(\mathsf{triples})$ to $\mathcal{F}_{\text{TripleGen}}$ and obtain $(\langle \boldsymbol{a} \rangle, \langle \boldsymbol{b} \rangle, \langle \boldsymbol{c} \rangle)$.
(2) Let $\boldsymbol{c}_i$ be P_is 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 $\mathcal{F}_{\text{Commit}}$.
(3) Each party P_i opens its commitments to $\boldsymbol{\sigma}_i$ towards all parties. Check if $\boldsymbol{0} = \sum_i \boldsymbol{\sigma}_i$. If not, abort.
(4) Let $A = [m]_1$. For $j = 1, ..., t$ all parties do the following
  (4.1) Sample the uniformly random value $r \leftarrow \text{ProvideRandom}(m, 1)$. Set $A \leftarrow A \backslash \{r\}$.
  (4.2) Each party P_i commits to its shares $\boldsymbol{a}_i[r], \boldsymbol{b}_i[r], \boldsymbol{c}_i[r]$ using $\mathcal{F}_{\text{Commit}}$.
  (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 $U = [m]_1 \backslash A$, where $U = \{u_1, ..., u_l\}$.
Compute $d \leftarrow \mathcal{P}_{\text{CheckMac}}.\text{CheckOutput}(\boldsymbol{\sigma}, \boldsymbol{a}[u_1], \boldsymbol{b}[u_1], \boldsymbol{c}[u_1], ..., \boldsymbol{a}[u_l], \boldsymbol{b}[u_l], \boldsymbol{c}[u_l])$.
If $d \neq 0$ the parties return $\perp$.
(6) Let $O \subset A$ be the smallest $k - t - 1$ indices from A. The parties output $(\langle \boldsymbol{a}[O] \rangle, \langle \boldsymbol{b}[O] \rangle, \langle \boldsymbol{c}[O] \rangle)$

Fig. 10. Protocol that checks the correctness of triples

Remark 2. Let C be an $[m, k, d]$ code and $\boldsymbol{x} \in C \backslash \{\boldsymbol{0}\}$. Choose elements t times from $[m]_1$ uniformly at random - denote this choice as the set $S \subseteq [m]_1$. 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 from the fact that the codeword must be nonzero for at least d positions, hence the probability that we hit one of the (at least) d nonzero elements is $\frac{m-d}{m}$ for each experiment. We now independently repeat this experiment t times, which yields 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 $t = O(sec), t < k-1$ and $d' = O(m)$ where $d' < m$. Then there exists a simulator $\mathcal{S}_{\text{TripleCheck}}$ such that $\mathcal{S}_{\text{TripleCheck}} \diamond \mathcal{F}_{\text{FullTripleGen}}$ is statistically indistinguishable from $\Pi_{\text{TripleCheck}}$ in the $\mathcal{F}_{\text{TripleGen}}$, $\mathcal{F}_{\text{Commit}}$-hybrid model with random oracle and a broadcast channel where, for n players, up to $n-1$ can act maliciously. The statistical distance of both executions is negligible in the security parameter sec.*

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

Proof. In order to prove the theorem, we use the simulator as described in Figure 11. The simulator will run a local copy of the protocol, together with local copies of $\mathcal{F}_{\text{TripleGen}}$, $\mathcal{F}_{\text{Commit}}$ and the random oracle.

Correctness We first observe that the protocol $\Pi_{\text{TripleCheck}}$ is correct in the sense that, if all parties follow it without deviations, they obtain outputs as in $\mathcal{F}_{\text{FullTripleGen}}$. This trivially follows from the fact that $\mathcal{F}_{\text{TripleGen}}$ and $\mathcal{F}_{\text{FullTripleGen}}$ 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 with a correct MAC as promised.

Simulatability Neither in $\mathcal{F}_{\text{TripleGen}}, \mathcal{P}_{\text{Angle}}$ nor in $\mathcal{F}_{\text{FullTripleGen}}$ the adversary will ever receive output. Moreover, in the simulation we follow exactly the protocol for the simulated honest parties and stop whenever

Simulator $\mathcal{S}_{\text{TripleCheck}}$

SimulateInitialize:

(1) We start our own instance Π of the protocol $\Pi_{\text{TripleCheck}}$ which the adversary is communicating with, and also local instances of $\mathcal{F}_{\text{Commit}}$ and the random oracle.
(2) Send (init, p, C) on behalf of the simulated honest parties to $\mathcal{F}_{\text{TripleGen}}$ and obtain the shares α_j.
(3) Send the intercepted set $\mathtt{P}_{Bad}$ to $\mathcal{F}_{\text{FullTripleGen}}$.
(4) Send $(\mathsf{init}, \mathbb{Z}_p, k-t-1)$ in the name of the dishonest parties to $\mathcal{F}_{\text{FullTripleGen}}$.
(5) For each dishonest party $\mathtt{P}_i$, send the intercepted share α_i to $\mathcal{F}_{\text{FullTripleGen}}$.

SimulateTriples:

(1) Set the flag *cheated* $\leftarrow \perp$.
(2) Send $(\mathsf{triples})$ to $\mathcal{F}_{\text{TripleGen}}$ 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 the adversary to input $\boldsymbol{\delta}, \boldsymbol{\Delta}_{\gamma,a}, \boldsymbol{\Delta}_{\gamma,b}, \boldsymbol{\Delta}_{\gamma,c} \in \mathbb{F}^m$ 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 $\mathcal{F}_{\text{TripleGen}}$.
(6) Commit to the correct value $\boldsymbol{\sigma}_j = \boldsymbol{H}\boldsymbol{c}_j$.
(7) Let $\{\boldsymbol{\sigma_i}\}_{i\in \mathtt{P}_{Bad}}$ be the commitments $\mathtt{A}$ sent to $\mathcal{F}_{\text{Commit}}$. If $\sum_{i\in\mathtt{P}_{Bad}} \boldsymbol{\sigma}_i \neq \boldsymbol{H}(\sum_{i\in\mathtt{P}_{Bad}} \boldsymbol{c}_i)$ then set *cheated* $\leftarrow \top$.
(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 that $\mathtt{A}$ opens does not equal the expected sum of $\boldsymbol{a}_i, \boldsymbol{b}_i$ or $\boldsymbol{c}_i$, then set *cheated* $\leftarrow \top$.
(10) Do step (5) as in the protocol with the values that should have been used if we followed the protocol correctly. Abort if Π aborts, if $\boldsymbol{\delta} \neq \boldsymbol{0}$ or if *cheated* $== \top$.
(11) Send $(\mathsf{triples})$ to $\mathcal{F}_{\text{FullTripleGen}}$ in the name of $\mathtt{A}$.
(12) Compute the set O as in Π. Send the adversarial $\boldsymbol{a}_i[O], \boldsymbol{b}_i[O], \boldsymbol{c}_i[O]$ for each dishonest $\mathtt{P}_i$ to $\mathcal{F}_{\text{FullTripleGen}}$.
(13) For every call of $\mathcal{P}_{\text{Angle}}$, send the same values to $\mathcal{F}_{\text{FullTripleGen}}$ that $\mathtt{P}_i$ sent to $\mathcal{F}_{\text{TripleGen}}$ for the same call.

Fig. 11. Simulator for $\Pi_{\text{TripleCheck}}$.

the protocol would stop. In addition, we also abort if the protocol would not abort, but the adversary inserted a value $\boldsymbol{\delta} \neq \boldsymbol{0}$ or if he opened values that differ from what he obtained. We now argue that the probability that the protocol does not abort while the simulator does is negligible.
In the case of *cheated* $== \top$, this follows the argument of the correctness of the online phase of SPDZ, i.e. Lemma 1. Therefore, let us assume that *cheated* $== \perp$, but $\boldsymbol{\delta} \neq \boldsymbol{0}$. Observe that

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

$\mathcal{S}_{\text{TripleCheck}}$ is constructed such that $Pr[\mathcal{S}_{\text{TripleCheck}} \text{ aborts} \,|\, \boldsymbol{\delta} \neq \boldsymbol{0}] = 1$. 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:

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

Since *cheated* $== \perp$ both step (2) and (4) are carried out correctly except with negligible probability i.e. $\mathtt{A}$ did not provide incorrect values.
If $\boldsymbol{\delta} \notin C^*$, then $\sum_{i=1}^{n} \boldsymbol{\sigma}_i \neq \boldsymbol{0}$ due to the fact that $\boldsymbol{H}$ is a check matrix. Hence we have that

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

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

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

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

Fast and Amortized Checks Until now, we checked 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 - but assuming that this must be carried out for a number of e.g. m vectors this leads to $O(m^3)$ operations, if done trivially.
As a first optimization, let us put all the l input vectors $\boldsymbol{a_1}, ..., \boldsymbol{a_l}$ into a matrix $\boldsymbol{A} = [\boldsymbol{a_1} || \boldsymbol{a_2} || ... || \boldsymbol{a_l}]$. If all vectors are drawn from the code, then $\boldsymbol{HA} = \boldsymbol{0}$. Computing this matrix product can be optimized (see e.g. [36] and the references therein) and be done in $O(m^{2+\nu})$ for some small constant $\nu > 0$. One can do better using protocol $\mathcal{P}_{\text{MatrixMultiplicationCheck}}$ as described in Figure 12.

```
Procedure $\mathcal{P}_{\text{MatrixMultiplicationCheck}}$

CheckMultiplication($\boldsymbol{H}, \boldsymbol{A}$):
    (1) Compute the matrices $\boldsymbol{GH}$ and $\boldsymbol{AG}^\top$.
    (2) For $j = 1, ..., m'$ select a pair $(x_j, y_j) \in \{1, ..., m'\}^2$
    (3) For $j = 1, ..., m'$, compute $z_j$ as the inner product of the $x_j$th row of $\boldsymbol{GH}$ and the $y_j$th column of $\boldsymbol{AG}^\top$.
    (4) If all $z_i$ are 0 return accept, otherwise reject.
```

Fig. 12. Procedure to check whether a matrix product is zero

Here once again codes help out: 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 moreover we 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 nonzero. 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 scale product. In case that at least one $\boldsymbol{a}_i$ is not a codeword, it will be nonzero with constant probability. Repeating this experiment $\Omega(m')$ times yields 0 in all cases only with probability negligible in m'. We refer to [18] for a more formal description.

3.2 More Efficient Proofs of Plaintext Knowledge

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[8] 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.

[8] 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 $\Pi_{\text{NewPracticalZK}}$

The prover inputs $\boldsymbol{x}_1, \boldsymbol{r}_1, \ldots, \boldsymbol{x}_t, \boldsymbol{r}_t$ and proves R_{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 $||\boldsymbol{r}_i||_\infty \leq \rho$.

(1) Let $v = 1$ be a counter. Set $T = t \cdot b_1 + c \cdot b_2 \cdot 2 \cdot \sqrt{t}$.
(2) $(\boldsymbol{a}_1, \boldsymbol{y}_1, \boldsymbol{s}_1, ..., \boldsymbol{a}_T, \boldsymbol{y}_T, \boldsymbol{s}_T) \leftarrow \mathcal{P}_{CutAndChoose}(T, (W+L) \cdot \tau \cdot \delta \cdot T \cdot \sqrt{t}, (W+L) \cdot \rho \cdot \delta \cdot T \cdot \sqrt{t})$.
(3) `PR`, `VE` jointly sample a uniformly random partition $V_1, ..., V_{b_1+b_2} \subset [T]_1$ using $\mathcal{P}_{\text{ProvideRandom}}$, such that

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

(4) For $z = 1, ..., b_1$ do the following:
  (4.1) Rename $V_z = \{v_1, ..., v_t\}$. `PR` computes $\boldsymbol{\alpha}_i = \boldsymbol{x}_i + \boldsymbol{y}_{v_i}$, $\boldsymbol{\beta}_i = \boldsymbol{r}_i + \boldsymbol{s}_{v_i}$.
  (4.2) `PR` checks that $||\boldsymbol{\alpha}_i||_\infty \leq (W+L) \cdot \tau \cdot \delta \cdot T \cdot \sqrt{t} - \tau$ and $||\boldsymbol{\beta}_i||_\infty \leq (W+L) \cdot \tau \cdot \delta \cdot T \cdot \sqrt{t} - \rho$. If not, then `PR` sends $\perp$ and starts again.
  (4.3) `PR` sends $(\boldsymbol{\alpha}_i, \boldsymbol{\beta}_i)_{i \in [t]_1}$ to `VE`.
  (4.4) `VE` checks that $\text{Enc}^S_{pk}(\boldsymbol{\alpha}_i, \boldsymbol{\beta}_i) = \boldsymbol{c}_i + \boldsymbol{a}_{v_i}$ and $||\boldsymbol{\alpha}_i||_\infty, ||\boldsymbol{\beta}_i||_\infty$ follow the above bounds.
  (4.5) If a check does not hold: If $v < M$, increment v by 1 and go to step (2). If $v = M$ then `VE` *rejects*.

(5) `PR`, `VE` together sample b_2 permutations $W_1, ..., W_{b_2}$ of $[t]_1$ using $\mathcal{P}_{\text{ProvideRandom}}$. For $z = 1, ..., b_2$ do the following:
  (5.1) Let $u = 2 \cdot \sqrt{t}$ and $W_z = \{w_1, ..., w_t\}$. For $i = 1, ..., \sqrt{t}$ 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 $i = 1, ..., u$, $k = 1, ..., c$ and $V_{b_1+z} = \{v_1, ..., v_{u \cdot c}\}$ `PR` computes $\boldsymbol{\alpha}^k_i = \boldsymbol{x}'_i + \boldsymbol{y}_{v_{(i-1)\sqrt{t}+k}}$ and similarly $\boldsymbol{\beta}^k_i = \boldsymbol{r}'_i + \boldsymbol{s}_{v_{(i-1)\sqrt{t}+k}}$.
  (5.3) `PR` checks that $||\boldsymbol{\alpha}^k_i||_\infty \leq (W+L) \cdot \tau \cdot \delta \cdot T \cdot \sqrt{t} - \sqrt{t} \cdot \tau$ and $||\boldsymbol{\beta}^k_i||_\infty \leq (W+L) \cdot \tau \cdot \delta \cdot T \cdot \sqrt{t} - \sqrt{t} \cdot \rho$. If not, then he sends $\perp$ to `VE` and starts again.
  (5.4) `PR` sends $(\boldsymbol{\alpha}^k_i, \boldsymbol{\beta}^k_i)$ to `VE`.
  (5.5) `VE` checks that $\text{Enc}^S_{pk}(\boldsymbol{\alpha}^k_i, \boldsymbol{\beta}^k_i) = \boldsymbol{c}'_i + \boldsymbol{a}_{v_{(i-1)\sqrt{t}+k}}$ and if $||\boldsymbol{\alpha}^k_i||_\infty, ||\boldsymbol{\beta}^k_i||_\infty$ have the bounds from the previous step.
  (5.6) If a check does not hold: If $v < M$, increment v by 1 and go to step (2). If $v = M$ then `VE` *rejects*.
(6) `VE` accepts.

Fig. 13. Asymptotically efficient ZKPoPK $\Pi_{\text{NewPracticalZK}}$

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 Figure 14, we give a graphical description of how the extractor should work if one abstracts the problem as a *bins and balls*-game. In a more formal way, we can describe Figure 14 as an algorithm:

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

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

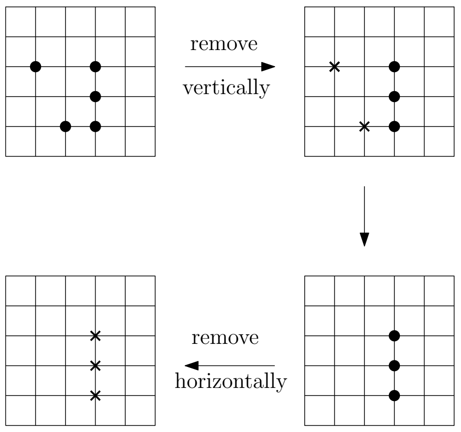

Fig. 14. Two-dimensional elimination process (strong extraction)

Compute

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

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 *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.

sec	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 $\Pi_{\text{NewPracticalZK}}$

We want to point out that we use *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 *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.

Theorem 2. *Let* $(\text{KeyGen}^S, \text{Enc}^S, \text{Dec}^S)$ *be a somewhat homomorphic cryptosystem as defined above with* $\delta > 1, T > sec$, *then the protocol* $\Pi_{\text{NewPracticalZK}}$ *is a honest-verifier zero-knowledge proof of plaintext knowledge for the relation* R_{POPK} *where*

$B_P = 2 \cdot (W+L) \cdot \tau \cdot \delta \cdot T \cdot t$, $B_R = 2 \cdot (W+L) \cdot \rho \cdot \delta \cdot T \cdot t$ *for weakly extracted ciphertexts.*
$B_P = 2 \cdot (W+L) \cdot \tau \cdot \delta \cdot T \cdot t^{3/2}$, $B_R = 2 \cdot (W+L) \cdot \rho \cdot \delta \cdot T \cdot t^{3/2}$ *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 $\mathcal{P}_{CutAndChoose}$ instead which ensures that, during the soundness argument, one must only be able to extract at most *sec* and not all t plaintexts due to the cut-and-choose. Another 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. A related, more generalized approach is subject to some ongoing follow-up work. We also want to mention that $\Pi_{\text{NewPracticalZK}}$ can be used to implement $\mathcal{F}_{\text{TripleGen}}$ from the previous subsection. 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 PR follows the protocol correctly and hence $\boldsymbol{c}_1, ..., \boldsymbol{c}_t$ were generated according to the correct distributions. Due to the linearity of $\text{Enc}^S_{pk}(\cdot,\cdot)$, the protocol only aborts if one of the coefficients of $\boldsymbol{\alpha}^j_i, \boldsymbol{\beta}^j_i$ 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 $\boldsymbol{\alpha}^j_i, \boldsymbol{\beta}^j_i$ in both step (4) or (5) lies outside of the correct bounds is at most $1/((W+L) \cdot \delta \cdot T)$. There are $W+L$ such coefficients and T sums that are computed in total, hence the probability that PR must send $\perp$ is $1/\delta$. For M such rounds, the chance that PR fails to prove correct values is $(1/\delta)^M$.

Honest-Verifier Zero-Knowledge Consider the following algorithm:

(1) Let $v = 1$ be a counter.
(2) Choose the subset $V \subset [2T]_1$ uniformly at random of size T. Choose the $b_1 + b_2$ subsets $V_1, ..., V_{b_1+b_2} \subset [2T]_1 \setminus V$ as in step (3) in $\Pi_{\text{NewPracticalZK}}$ at random according to their size constraints. Moreover, choose the b_2 random permutations $W_1, ..., W_{b_2}$ of $[t]_1$ 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 $\Pi_{\text{NewPracticalZK}}$.
(4) For each of the T sums $\boldsymbol{\alpha}^j_i$, choose for $\boldsymbol{\alpha}^j_i$ each coefficient uniformly at random from the interval $[-(W+L) \cdot \delta \cdot \tau \cdot T \cdot \sqrt{t}, (W+L) \cdot \delta \cdot \tau \cdot T \cdot \sqrt{t}]$. Moreover, choose for each of the T sums $\boldsymbol{\beta}^j_i$ each coefficient uniformly at random from the interval $[-(W+L) \cdot \delta \cdot \rho \cdot T \cdot \sqrt{t}, (W+L) \cdot \delta \cdot \rho \cdot T \cdot \sqrt{t}]$.
(5) For each of the $i \in [2T]_1$ from $\mathcal{P}_{CutAndChoose}$ we do the following:
   – If $i \in V$ then sample $\boldsymbol{y}_i, \boldsymbol{s}_i$ as in step (1) of $\mathcal{P}_{CutAndChoose}$. Then compute $\boldsymbol{a}_i \leftarrow \text{Enc}^S_{pk}(\boldsymbol{y}_i, \boldsymbol{s}_i)$.
   – If $i \notin V$ then set $\boldsymbol{a}_{v_i} = \text{Enc}^S_{pk}(\boldsymbol{\alpha}^j_i, \boldsymbol{\beta}^j_i) \ominus \boldsymbol{c}_i$ if $i \in V_1, ..., V_{b_1}$ or (with an obvious reindexing based on W_z, V_{b_1+z} for $z = 1, ..., b_2$) compute $\boldsymbol{a}_i = \text{Enc}^S_{pk}(\boldsymbol{\alpha}^j_i, \boldsymbol{\beta}^j_i) \ominus \boldsymbol{c}'_j$ if $i \in V_{b_1+1}, ..., V_{b_1+b_2}$.
(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 $j \in 1, ..., b_1$ if it holds that $\forall i \in V_j$ the plaintexts and randomness are small enough, i.e.

$$||\boldsymbol{\alpha}^j_i||_\infty \leq (W+L) \cdot \delta \cdot \tau \cdot T \cdot \sqrt{t} - \tau \text{ and } ||\boldsymbol{\beta}^j_i||_\infty \leq (W+L) \cdot \delta \cdot \rho \cdot T \cdot \sqrt{t} - \rho$$

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

$$||\boldsymbol{\alpha}^j_i||_\infty \leq (W+L) \cdot \delta \cdot \tau \cdot T \cdot \sqrt{t} - \sqrt{t} \cdot \tau \text{ and } ||\boldsymbol{\beta}^j_i||_\infty \leq (W+L) \cdot \delta \cdot \rho \cdot T \cdot \sqrt{t} - \sqrt{t} \cdot \rho$$

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

The above algorithm outputs transcripts that are perfectly indistinguishable from transcripts generated by $\Pi_{\text{NewPracticalZK}}$. 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 $\Pi_{\text{NewPracticalZK}}$.
- Let $i \in [2T]_1 \setminus V$. For each $\boldsymbol{\alpha}_i^j, \boldsymbol{\beta}_i^j$ in $\Pi_{\text{NewPracticalZK}}$, if it is given as output to `VE` by an honest `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 $\Pi_{\text{NewPracticalZK}}$ 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 $||\boldsymbol{\alpha}_i^j||_\infty, ||\boldsymbol{y}_i||_\infty \leq (W+L) \cdot \tau \cdot \delta \cdot T \cdot \sqrt{t}$ it must hold that $||\boldsymbol{x}_j||_\infty \leq 2 \cdot (W+L) \cdot \tau \cdot \delta \cdot T \cdot \sqrt{t}$. By the same argument, the extracted randomness $\boldsymbol{r}_j$ has coefficients of size at most $2 \cdot (W+L) \cdot \rho \cdot \delta \cdot T \cdot \sqrt{t}$. 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 $\mathcal{P}_{CutAndChoose}$. It is therefore computed from an expanded seed f_i which was fed into the random oracle. One can extract the RO queries that `PR` makes, and in particular the f_i seeds (more precisely, one can extract $T - sec$ of them except with probability 2^{-sec}).

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 $(W+L) \cdot \delta \cdot \tau \cdot T \cdot \sqrt{t}$) and at most $\sqrt{t} - 1$ plaintexts from step (4), which yields a total bound of $2 \cdot (W+L) \cdot \delta \cdot \tau \cdot T \cdot t$. The bound on the randomness can be established the same way. For *strong extraction*, we first observe that the process now exactly follows *matrixGame* as defined above. The plaintexts extracted during step (2) of *matrixGame* will once again have norm at most $2 \cdot (W+L) \cdot \delta \cdot \tau \cdot T \cdot t$ 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 *matrixGame*. Therefore, the bound on the plaintexts extracted during step (3) can be as high as $2 \cdot (W+L) \cdot \delta \cdot \tau \cdot T \cdot t^{3/2}$ 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 with negligible probability. This is a mere computational problem and will be answered in Appendix A.2.

4 Improved Preprocessing from Paillier Encryption

In this section we will give a novel approach on preprocessing for e.g. [19] using Paillier's cryptosystem. Before doing so, we present two certain ZK proofs which we will make heavy use of. In comparison to previous work, they will require to send less bits per ZK proof instance.

4.1 Proving Statements about Paillier Ciphertexts

The first statement that we want to prove is a regular proof of plaintext knowledge. For Paillier encryption, this can be phrased as follows:

$$\begin{aligned} R_{ZKPoPKPaillier} = \{(a, w) \mid \ & a = (c, pk), w = (x, r), x \in \mathbb{Z}/N\mathbb{Z} \wedge r \in Z/N\mathbb{Z}^* \wedge \\ & c \in \mathbb{Z}/N^2\mathbb{Z}^* \wedge c = \text{Enc}_{pk}^P(x, r)\} \end{aligned}$$

In addition, we let our parties compute linear relations. We do not force them directly to do the right computation, but instead just make sure that they *know* what they multiplied in.

$$\begin{aligned}R_{LinearPaillier} = \{(a,w) \mid\ & a=(z,\hat{z},pk), w=(b,c,r), b,c \in \mathbb{Z}/N\mathbb{Z} \wedge r \in \mathbb{Z}/N\mathbb{Z}^* \wedge \\ & z,\hat{z} \in \mathbb{Z}/N^2\mathbb{Z}^* \wedge \hat{z} = z^b \cdot \mathrm{Enc}^P_{pk}(c,r) \bmod N^2\}\end{aligned}$$

In the following, we present honest-verifier perfect zero-knowledge proofs for both $R_{ZKPoPKPaillier}, R_{LinearPaillier}$. In order to use them in the preprocessing protocol, one can either make them non-interactive using the Fiat-Shamir transformation in the Random Oracle Model, or use the secure coin-flip protocol $\mathcal{P}_{\text{ProvideRandom}}$ 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.
It is worth mentioning that, for efficiency reasons, one can choose the random value e from a smaller interval like e.g. $[0, 2^{sec}]$ where sec is the statistical security parameter. This also yields negligible cheating probability[9].

Proof of Plaintext Knowledge We first make an observation about the Paillier encryption scheme, namely that Enc^S_{pk} is a bijection:

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

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

Protocol $\Pi_{\text{ZKPoPKPaillier}}$

PR proves the relation $R_{ZKPoPKPaillier}$.

(1) PR chooses $s \leftarrow \mathbb{Z}/N\mathbb{Z}^*$ and sends $t = s^N \mod N$ to VE.
(2) VE chooses $e \leftarrow \mathbb{Z}/N\mathbb{Z}$ and sends it to PR.
(3) PR sends $k = s \cdot r^e \mod N$ to VE.
(4) VE accepts if $k^N = c^e \cdot t \mod N$ and otherwise rejects.

Fig. 15. Protocol $\Pi_{\text{ZKPoPKPaillier}}$ to prove knowledge of plaintexts of Paillier encryptions

In the preliminaries, we discussed decryption as recovering the noise first, and then using the noise to decrypt the ciphertext. Observe that the second part can be done locally and without knowledge of the secret key. This allows one to optimize the proof of plaintext knowledge from [15] to work mostly $\mod N$ and use the same proof technique.

Lemma 2. *The protocol $\Pi_{\text{ZKPoPKPaillier}}$ is an an interactive honest-verifier proof for the relation $R_{ZKPoPKPaillier}$.*

Proof.

Completeness Let $c = \mathrm{Enc}^P_{pk}(x,r) \in \mathbb{Z}/N^2\mathbb{Z}^*$, and observe that $c = r^N \mod N$. 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.

[9] 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.

Soundness Assume that VE obtains two protocol transcripts $(t, e, k), (t, e', k')$ such that $e \neq e' \bmod N$ and $e > e'$, from which one can deduce that also $k \neq k' \bmod N$. This claim follows, since $c = r^N \bmod N$ and we can write $1 = c^{e-e'} = r^{N\cdot(e-e')} \bmod N$ where $(N, \varphi(N)) = 1$. 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 obtain

$$k^N = c^e \cdot t \bmod N \text{ and } k'^N = c^{e'} \cdot t \bmod N \tag{1}$$

By dividing the first by the second equation in (1) 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 $v = (c \bmod N)^\alpha \cdot (k/k')^\beta \mod N$ to decrypt c and obtain the plaintext pair (u, v). This must be the correct plaintext, because Enc^S_{pk} 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) Choose $k \leftarrow \mathbb{Z}/N\mathbb{Z}^*$, $e \leftarrow \mathbb{Z}/N\mathbb{Z}$ uniformly at random.
(2) Set $t = k^N \cdot c^{-e} \bmod N$.
(3) Output (t, e, k).

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

PR proves the relation $R_{LinearPaillier}$.

(1) PR generates $u, v \in \mathbb{Z}/N\mathbb{Z}, w \in \mathbb{Z}/N\mathbb{Z}^*$. He then sends $f = z^u \cdot \mathrm{Enc}^P_{pk}(v, w) \mod N^2$ to VE.
(2) VE chooses a uniformly random $e \in \mathbb{Z}/N\mathbb{Z}$ and sends it to PR.
(3) PR computes $\alpha = u + e \cdot b \mod N, h = v + e \cdot c \mod N, i = w \cdot r^e \mod N$ and and sends (g, h, i) to VE.
(4) VE accepts if $z^g \cdot \mathrm{Enc}^P_{pk}(h, i) = \hat{z}^e \cdot f \mod N^2$, and rejects otherwise.

Fig. 16. Protocol $\Pi_{\text{RelPaillier}}$ to prove linear relation on ciphertexts

Proof of Linear Relation

Lemma 3. *The protocol $\Pi_{\text{RelPaillier}}$ is an an interactive honest-verifier proof for the relation $R_{LinearPaillier}$.*

Proof.

Completeness Let $\hat{z} = z^b \cdot \mathrm{Enc}^P_{pk}(c, r) \mod N^2$, 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

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

Soundness Assume that `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 `PR` were honest as it could otherwise break the security assumption in a similar way as in the proof of Lemma 2.
We obtain

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

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

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

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

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

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

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

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

Honest-Verifier Zero-Knowledge To simulate $\Pi_{\text{RelPaillier}}$, we use the following algorithm:

(1) Choose $e \leftarrow \mathbb{Z}/N\mathbb{Z}$ uniformly at random.
(2) Choose $g, h \leftarrow \mathbb{Z}/N\mathbb{Z}$, $i \leftarrow \mathbb{Z}/N\mathbb{Z}^*$ uniformly at random.
(3) Compute $f \leftarrow z^g \cdot \mathrm{Enc}^P_{pk}(h, i)/\hat{z}^{-e}$.
(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).

□

4.2 Computing and Checking Triples

With the above tools, we are now ready to construct our protocol $\Pi_{\text{PaillierTripleGen}}$, which we prove to be correct with respect to $\mathcal{F}_{\text{FullTripleGen}}$. In the following, $[m]$ will denote an encryption of message $m \in \mathbb{Z}/N\mathbb{Z}$, where the randomness is left implicit. E
In comparison to the preprocessing protocol in the preceding section, the protocol $\Pi_{\text{PaillierTripleGen}}$ as depicted in Figure 17 and Figure 18 is rather lengthy. To ease understanding, let us outline the main phases of it:

(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 $1, ..., k+2$). Interpolate this polynomial A in the next $k+2$ points locally, encrypt them and prove that the resulting points indeed lie on A. Do the same for a polynomial B.
(3) Do an unreliable point-wise multiplication of A, B. The resulting polynomial C is interpolated 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) Share the points of C among all parties as random shares.
(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 proof of security of the protocol $\Pi_{\textsc{PaillierTripleGen}}$ will be given in this section. It relies on the following

Remark 3. 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 $x \overset{\$}{\leftarrow} \mathbb{Z}/N\mathbb{Z}$. 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,

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

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 $\bmod N$, it is also uniformly random $\bmod p$ 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 $\bmod q$ 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).

Theorem 3. *The protocol $\Pi_{\textsc{PaillierTripleGen}}$ securely implements the functionality $\mathcal{F}_{\textsc{FullTripleGen}}$ using a random oracle and a broadcast channel in the $\mathcal{F}_{\mathrm{KGD},\textsc{Paillier}}$ hybrid model with security against every malicious static PPT adversary controlling at most $n-1$ parties if the $CR[N]$-problem is hard.*

We use a standard proof technique for UC secure MPC protocols with encryption, which goes as follows: 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 Z that can distinguish the transcript from such a simulator from a protocol transcript. Then we can use such an Z as a subroutine to break the IND-CPA security. Such a subroutine can rewind Z and thereby avoid decryption altogether.
The functionality $\mathcal{F}_{\textsc{FullTripleGen}}$ that our protocol implements is weaker than what $\Pi_{\textsc{PaillierTripleGen}}$ achieves, because it does not allow the adversary to introduce errors into the MACs (which is in fact possible in $\Pi_{\textsc{TripleCheck}}$ and hence allowed in $\mathcal{F}_{\textsc{FullTripleGen}}$).

Protocol $\Pi_{\text{PaillierTripleGen}}$

In each call of $\Pi_{\text{ZKPoPKPaillier}}, \Pi_{\text{RelPaillier}}$ the parties sample the challenge e either together using $\mathcal{P}_{\text{ProvideRandom}}$ or individually using the Fiat-Shamir transform.

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

(1) Each party $\mathtt{P}_i$ picks $\alpha_i \in \mathbb{Z}/N\mathbb{Z}$ uniformly at random, broadcasts a fresh encryption $[\alpha_i]$ and proves knowledge of plaintext of $[\alpha_i]$ using $\Pi_{\text{ZKPoPKPaillier}}$.
(2) The parties compute $[\alpha] \leftarrow \prod_{i=1}^{n}[\alpha_i]$.
(3) Each $\mathtt{P}_i$ stores $[\alpha]$ as the encrypted MAC key and its share α_i of the MAC key.

Triples: On input $(\mathsf{triples})$ the parties do the following:

(1) For $j = 1, \ldots, k+2$ each $\mathtt{P}_i$ picks $A_i(j), B_i(j) \in \mathbb{Z}/N\mathbb{Z}$ uniformly at random, computes $[A_i(j)], [B_i(j)]$ and broadcasts $([A_i(j)], [B_i(j)])_{j\in\{1,\ldots,k+2\}}$ together with proofs of $\Pi_{\text{ZKPoPKPaillier}}$.
(2) For $j = 1, ..., k+2$ 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 $([A_i(l)], [B_i(l)])_{l=k+3,\ldots,2k+2}$ together with proofs of plaintext knowledge using $\Pi_{\text{ZKPoPKPaillier}}$.
(3) The parties locally compute

$$[A(l)] = \prod_{i=1}^{n}[A_i(l)] \text{ and } [B(l)] = \prod_{i=1}^{n}[B_i(l)]$$

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

$$[A^{\dagger}(\beta)] = [A^{\top}(\beta)]/[A^{\perp}(\beta)] \text{ and } [B^{\dagger}(\beta)] = [B^{\top}(\beta)]/[B^{\perp}(\beta)]$$

(6) The parties decrypt $[A^{\dagger}(\beta)], [B^{\dagger}(\beta)]$ and check whether $A^{\dagger}(\beta) = B^{\dagger}(\beta) = 0 \bmod N$. Otherwise they abort.
(7) For $j = 1, \ldots, 2k + 2$ each $\mathtt{P}_i$ chooses $r_{i,j} \leftarrow \mathbb{Z}/N\mathbb{Z}^*$, computes encryptions

$$[\hat{c}_{i,j}] \leftarrow [A(j)]^{B_i(j)}\, \text{Enc}^{P}_{pk}(0, r_{i,j})$$

broadcasts the $[\hat{c}_{i,j}]$ and proves the relation using $\Pi_{\text{RelPaillier}}$.
(8) For $j = 1, \ldots, 2k + 2$ each $\mathtt{P}_i$ picks $\tilde{c}_{i,j} \in \mathbb{Z}/N\mathbb{Z}$ uniformly at random, computes $[\tilde{c}_{i,j}]$ and broadcasts $([\tilde{c}_{i,j}])_{j\in\{0,\ldots,2k+3\}}$ together with proofs of $\Pi_{\text{ZKPoPKPaillier}}$.
(9) For $j = 1, \ldots, 2k + 2$ the parties locally compute $[\hat{c}_j] = \prod_{i=1}^{n}[\hat{c}_{i,j}]/\prod_{i=1}^{n}[\tilde{c}_{i,j}]$ and publicly decrypt $\hat{c}_j$.
(10) For $j = 1, \ldots, 2k + 2$ each party $\mathtt{P}_i$ sets $[C_1(j)] = [\tilde{c}_{1,j}] \cdot [\hat{c}_j], [C_i(j)] = [\tilde{c}_{t,j}]$ for $t \in [n]_1, t \neq 1$ and $[C(j)] = \prod_{i=1}^{n}[C_i(j)]$ 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 $\beta \leftarrow \mathcal{P}_{\text{ProvideRandom}}.\,\text{ProvideRandom}(N - k, 1) + k$ so that $\beta \leftarrow \mathbb{Z}/N\mathbb{Z} \setminus \{1, ..., k\}$.
(12) The parties compute $[A(\beta)], [B(\beta)], [C(\beta)]$ locally using Lagrange interpolation and then decrypt these values.
(13) If $A(\beta) \cdot B(\beta) \neq C(\beta) \bmod N$ then abort.

Fig. 17. Protocol $\Pi_{\text{PaillierTripleGen}}$ to generate correct random triples out of random single values

Proof (of Theorem 3). For the proof, we use the simulator $\mathcal{S}_{PaillierTripleGen}$ as depicted in Figure 19. 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 $\Pi_{\text{PaillierTripleGen}}$ and simulated transcripts of $\mathcal{S}_{PaillierTripleGen}$ with non-negligible probability σ in poly-

Protocol $\Pi_{\text{PaillierTripleGen}}$ (part 2)

In this part of the protocol, we will now compute MACs for all triples.

Triples:

(14) Each $\mathtt{P}_i$ picks $s_i \in \mathbb{Z}/N\mathbb{Z}$ uniformly at random, computes $[s_i]$ and broadcasts $[s_i]$ together with a proof of $\Pi_{\text{ZKPoPKPaillier}}$. Let $s = \sum_i s_i$

(15) We define the following abbreviation:

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

(16) For $j = 0, \ldots, 3k$ each $\mathtt{P}_i$ picks $r_{i,j} \in \mathbb{Z}/N\mathbb{Z}^*$ uniformly at random and computes

$$[t_{i,j} \cdot \alpha] \leftarrow [\alpha]^{t_{i,j}} \cdot \mathrm{Enc}^P_{pk}(0, r_{i,j})$$

then broadcasts $([t_{i,j} \cdot \alpha])$ and proves the relation using $\Pi_{\text{RelPaillier}}$.

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

$$[t_j \cdot \alpha] \leftarrow \prod\nolimits_{i=1}^{n} [t_{i,j} \cdot \alpha].$$

(18) The parties sample $\beta \leftarrow \mathcal{P}_{\text{ProvideRandom}}.\text{ProvideRandom}(N, 1)$.

(19) All parties compute

$$[v] \leftarrow \prod\nolimits_{j=0}^{3k} [t_j]^{\beta^j} \text{ and } [v'] \leftarrow \prod\nolimits_{j=0}^{3k} [t_j \cdot \alpha]^{\beta^j}$$

(20) The parties jointly decrypt $[v]$ to v and check that the decryption was correct.

(21) The parties jointly decrypt

$$[M] \leftarrow [\alpha]^v / [v']$$

and verify that $M == 0$. All parties verify correctness of decryption.

(22) For $j = 1, \ldots, 3k$ each $\mathtt{P}_i$ picks $m_{i,j} \in \mathbb{Z}/N\mathbb{Z}$ uniformly at random, computes $[m_{i,j}]$ and broadcasts $([m_{i,j}])_{j \in \{1,\ldots,3k\}}$ together with proofs of $\Pi_{\text{ZKPoPKPaillier}}$.

(23) For each $j = 1, \ldots, 3k$, the parties compute

$$[O_j] \leftarrow [t_j \cdot \alpha] / \prod\nolimits_{i=1}^{n} [m_{i,j}]$$

and publicly decrypt $[O_j]$. All parties verify correctness of decryption.

(24) For each $j = 1, \ldots, 3k$, each $\mathtt{P}_i$ determines its share $\gamma(t_j)_i$, of the MAC $\gamma(t_j) = t_j \cdot \alpha$ of t_j as

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

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

Fig. 18. Protocol $\Pi_{\text{PaillierTripleGen}}$ to generate correct random triples out of random single values

nomial time. Now, we show how to use the distinguisher to break the IND-CPA property of the encryption scheme.

Let Π_{Real} be the distribution of a real protocol execution, and Π_{Sim} be the distribution that $\mathcal{S}_{PaillierTripleGen}$ 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 $\Pi_{Real} \approx_{perf} \Pi_{Real,ZK}$. Defining $\Pi_{Sim,ZK}$ for Π_{Sim} in the same way, it also holds that $\Pi_{Sim} \approx_{perf} \Pi_{Sim,ZK}$.

Simulator $\mathcal{S}_{PaillierTripleGen}$

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

SimulateInitialize:

(1) For each simulated honest party $\mathtt{P}_i, i \in [n]_1 \setminus \mathtt{P}_{Bad}$, provide a fresh encryption of the random value $r_i \leftarrow \mathbb{Z}/N\mathbb{Z}$ which is sent to the dishonest parties as $c_i = [r_i]$. Then prove plaintext knowledge using Π_{ZKPoPK}.

(2) Decrypt the ciphertexts c_i that are obtained from the malicious parties $\mathtt{P}_i, i \in \mathtt{P}_{Bad}$. Send $(\mathsf{init}, \mathbb{Z}/N\mathbb{Z}, k)$ and the plaintexts in the name of the malicious parties to $\mathcal{F}_{\text{FullTripleGen}}$.

(3) Locally compute $[\alpha] = \prod_{i=1}^{n} [c_i]$ and $\alpha = \prod_{i=1}^{n} c_i$.

SimulateTriples:

(1) Perform step (1) as in the protocol. Decrypt the ciphertexts of $\mathtt{P}_i, i \in \mathtt{P}_{Bad}$ 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 $\mathtt{P}_i, i \in \mathtt{P}_{Bad}$ after step (2) as $A_i'(l), B_i'(l)$. Now set *cheated* as

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

(3) Perform step (3) – (6) as in the protocol. Abort if *cheated* $== \top$.

(4) Perform step (7) as in the protocol. Decrypt the ciphertexts of $\mathtt{P}_i, i \in \mathtt{P}_{Bad}$ as $\hat{c}_{i,j}'$. Set *cheated* as

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

(5) Perform step (8) – (13) as in the protocol. Abort in step (13) if *cheated* $== \top$.

(6) Send $(\mathsf{triples})$ to $\mathcal{F}_{\text{FullTripleGen}}$. 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 $\mathcal{F}_{\text{FullTripleGen}}$ for each $i \in \mathtt{P}_{Bad}$ and also $\boldsymbol{\Delta}_{\gamma,A} = \boldsymbol{\Delta}_{\gamma,B} = \boldsymbol{\Delta}_{\gamma,C} = \mathbf{0}$.

(7) Perform step (14) – (16) as in the protocol, and decrypt all values obtained from players controlled by $\mathtt{A}$. If in one of the ciphertexts $[t_{i,j} \cdot \alpha]$ of the dishonest parties is not of the form $\alpha \cdot t_{i,j}$ for the respective encrypted value $[t_{i,j}]$ then set *cheated* $= \top$, otherwise *cheated* $= \bot$.

(8) Perform step (17) – (21) as in the protocol. In step (21) the simulator aborts if *cheated* $== \top$.

(9) Perform step (22) – (25) as in the protocol.

(10) Set $\boldsymbol{\gamma(t)}[j] = \gamma(t_j)_i$ from the protocol. For each $i \in \mathtt{P}_{Bad}$ send $\boldsymbol{\gamma(A)}_i, \boldsymbol{\gamma(B)}_i, \boldsymbol{\gamma(C)}_i$ to $\mathcal{F}_{\text{FullTripleGen}}$ during the procedure $\mathcal{P}_{\text{Angle}}$.

Fig. 19. Simulator $\mathcal{S}_{PaillierTripleGen}$ for the protocol $\Pi_{\text{PaillierTripleGen}}$

In a next step, we replace the checks for correctness of statements as they are done in the protocol $\Pi_{\text{PaillierTripleGen}}$, with checks as in $\mathcal{S}_{PaillierTripleGen}$. 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 $\Pi_{\text{PaillierTripleGen}}$, abort under the same conditions as in $\mathcal{S}_{PaillierTripleGen}$ in step (3).
(2) In step (13) of $\Pi_{\text{PaillierTripleGen}}$, abort under the same conditions as in $\mathcal{S}_{PaillierTripleGen}$ in step (5).
(3) In step (21) of $\Pi_{\text{PaillierTripleGen}}$, abort under the same conditions as in $\mathcal{S}_{PaillierTripleGen}$ 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 $\mathcal{S}_{PaillierTripleGen}$. 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 λ and by Remark 3, we conclude that $Pr[A_\mathcal{S}] - Pr[A_\Pi] < negl(\lambda)$ and thereby $\Pi_{Real,ZK,Correct} \approx_{stat} \Pi_{Real,ZK}$.

Now consider an environment Z that can distinguish between Π_{Real} and Π_{Sim}. Since 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.
We now run the following algorithm B with Z as a a subroutine, which is used to generate all the interactions of the malicious parties. On a high level, B will run both $\Pi_{\text{PaillierTripleGen}}, \mathcal{S}_{PaillierTripleGen}$ synchronously and combine the messages to Z from both instances so that they are consistent with exactly one of them.

(1) Obtain the public-key N from the challenger C. Start simulation for $\mathcal{F}_{\text{KGD,Paillier}}$ and the random oracle. We simulate $\mathcal{F}_{\text{KGD,Paillier}}$ as follows:
 (1.1) If $\mathcal{F}_{\text{KGD,Paillier}}$ is queried to generate a key by all parties, then first sample n uniformly random values $r_1, ..., r_n$ from the same domain as in $\mathcal{F}_{\text{KGD,Paillier}}$. Then output (N, r_i) to each party $\mathtt{P}_i$.
 (1.2) If $\mathcal{F}_{\text{KGD,Paillier}}$ is queried to decrypt a ciphertext c, then we instead send c, m' to A where m' is chosen by this modified algorithm. We will later describe how each such m' is chosen.
(2) Send the plaintexts $m_0 = 0, m_1 = 1$ to C and obtain the challenge c_q. Set $c_0 = c_q, c_1 = [1]/c_q$.
(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 $c_0^{m_0} \cdot c_1^{m_1} \cdot [0]$ 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 Z.
(6) To simulate the decryption in $\mathcal{F}_{\text{KGD,Paillier}}$, we will provide the decrypted message m' as follows:
 (6.1) If we decrypt in step (6), (13), (21) in $\Pi_{\text{PaillierTripleGen}}$ or step (3), (5), (8) in $\mathcal{S}_{PaillierTripleGen}$ then we output 0 for $\mathcal{F}_{\text{KGD,Paillier}}$ if *cheated* $== \bot$ or a random nonzero[10] number if *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 $\delta_j \in \mathbb{Z}/N\mathbb{Z}$ and compute $[\tilde{c}_{i,j}] = c_0^{x_0} c_1^{x_1} \cdot [0]$, where

$$x_0 = \sum_{r\in[n]_1} \hat{c}_{r,j} - \delta_j - \sum_{r\in[n]_1\setminus \mathtt{P}_{Bad}\cup\{i\}} \tilde{c}_{r,j} \text{ and } x_1 = \sum_{r\in[n]_1} \hat{c}'_{r,j} - \delta_j - \sum_{r\in[n]\setminus \mathtt{P}_{Bad}\cup\{i\}} \tilde{c}'_{r,j}$$

 where 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 $\mathcal{S}_{PaillierTripleGen}$. Then in step (9), output $\hat{c}_j = \delta_j - \sum_{i\in\mathtt{P}_{Bad}} \tilde{c}_{i,j}$.
 (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 $\delta_j \in \mathbb{Z}/N\mathbb{Z}$ and compute $[m_{i,j}] = c_0^{x_0} c_1^{x_1} \cdot [0]$, where

$$x_0 = t_j \cdot \alpha - \delta_j - \sum_{r\in[n]_1\setminus \mathtt{P}_{Bad}\cup\{i\}} m_{r,j} \text{ and } x_1 = t'_j \cdot \alpha' - \delta_j - \sum_{r\in[n]\setminus \mathtt{P}_{Bad}\cup\{i\}} m'_{r,j}$$

 where 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 $\mathcal{S}_{PaillierTripleGen}$. Then in step (23), output $O_j = \delta_j - \sum_{i\in\mathtt{P}_{Bad}} m_{i,j}$.
 (6.4) In step (14), first choose a random value β and set $\mathcal{P}_{\text{ProvideRandom}}$ 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 $[s_r], r \in [n]_1\setminus\mathtt{P}_{Bad}\cup\{i\}$ in step (14). For $\mathtt{P}_i$, sample a random $\delta \in \mathbb{Z}/N\mathbb{Z}$ and compute $[s_i] = c_0^{x_0} c_1^{x_1} \cdot [0]$, where

$$x_0 = \delta - \sum_{r\in[n]_1\setminus \mathtt{P}_{Bad}\cup\{i\}} s_r - \sum_{j=1}^{3k} \left(\sum_{r\in\mathtt{P}_{Bad}} t_{r,j}^{\beta^j} + \sum_{r\in[n]_1\setminus\mathtt{P}_{Bad}} t_{r,j}^{\beta^j} \right)$$

$$x_1 = \delta - \sum_{r\in[n]_1\setminus \mathtt{P}_{Bad}\cup\{i\}} s'_r - \sum_{j=1}^{3k} \left(\sum_{r\in\mathtt{P}_{Bad}} t_{r,j}^{\beta^j} + \sum_{r\in[n]_1\setminus\mathtt{P}_{Bad}} t'^{\beta^j}_{r,j} \right)$$

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

where the values $s_r, t^{\beta^j}_{r,j}$ come from the real protocol and $s'_r, t'^{\beta^j}_{r,j}$ from the modified $\mathcal{S}_{PaillierTripleGen}$. Then in step (20), output the value $\delta + \sum_{i \in \mathtt{P}_{Bad}} s_i$ for the decryption of $[v]$.

(7) Finally, after obtaining the guess b_g from Z, send b_g to 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 C choose to encrypt $c_q = [0]$ with probability ρ and hence $c_q = [1]$ with probability $1 - \rho$. If $c_q = [0]$, 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, $c_q = [1]$, then the distribution will be $\Pi_{Sim,ZK}$ which will happen with probability $1 - \rho$. Since Z can distinguish both $\Pi_{Real,ZK,Correct}, \Pi_{Sim,ZK}$ with advantage at least σ, the output to 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, and Claudio Orlandi. Publicly auditable secure multi-party computation. In *Security and Cryptography for Networks*, pages 175–196. Springer, 2014.
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. Cryptology ePrint Archive, Report 2012/642, 2012.

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.
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. Irving S Reed and Gustave Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial & Applied Mathematics*, 8(2):300–304, 1960.
36. Joachim Von Zur Gathen and Jürgen Gerhard. *Modern computer algebra*. Cambridge University Press, 2013.
37. 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 section, 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. We implemented a solver for *matrixGame* and plan to make it available to the public.

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 Figure 20.

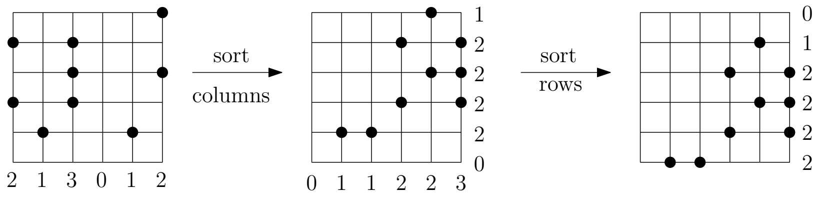

Fig. 20. Normal Form, Part 1

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 $(r_i)_{i\in[n]_1}, (c_i)_{i\in[m]_1}$ (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.[11]. 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

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

Similarly, one can obtain the number of column permutations C. Using R, C, $(r_i)_{i\in[n]_1}, (c_i)_{i\in[m]_1}$ and the number of solutions for each possible pair $(r_i)_{i\in[n]_1}, (c_i)_{i\in[m]_1}$ one can cover the whole solution space while only enumerating only 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:

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

[11] 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[12]. A drawback of the above approach is that all short vectors are enumerated, which includes vectors 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[13], 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 Figure 21. 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).

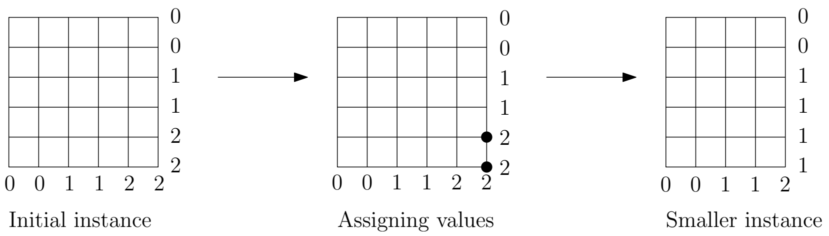

Fig. 21. Recursive elimination

[12] See [25] for a good explanation of the algorithm.

[13] 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 Figure 22. 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.

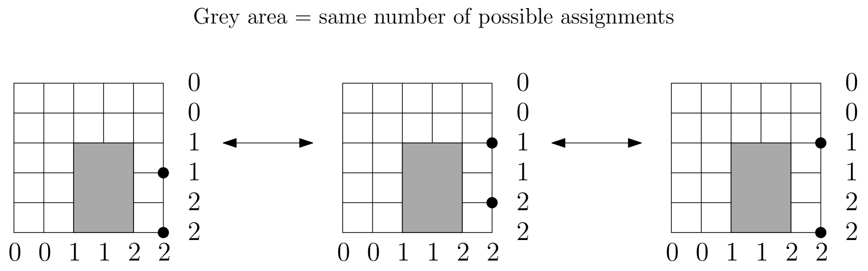

Fig. 22. Equivalent assignments

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

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

where all of the exponents are nonnegative. 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

We first assume that $T > sec$. 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^{-sec}. We denote the event where the extractor fails as A_{ExErr}. Moreover, we denote with S_i the event that 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 $i > sec$ we have that $Pr[\mathsf{A}_{ExErr}|\mathsf{S}_i, \mathsf{C}_i] \cdot 2^{-i} < 2^{-sec}$ we can ignore all $i > sec$ and simply focus on the case $i \in [1, sec]$. 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 $\Pi_{\text{NewPracticalZK}}$ 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 $\boldsymbol{\alpha}_j^1, ..., \boldsymbol{\alpha}_j^c$ will be paired up with one of the k bad ciphertexts. We can now model $Pr[\mathsf{A}_{ExErr} \mid \mathsf{S}_b]$ as

$$\begin{aligned}
Pr[\mathsf{A}_{ExErr} \mid \mathsf{S}_b] \leq \quad & Pr\Big[\bigcup_{i_1,...i_\aleph \in \mathbb{N}^+}^{i_1+...+i_\aleph=b} \bigcup_{x \in [sec]_1} \Big(\mathsf{V}_{1,i_1}, ..., \mathsf{V}_{\aleph,i_\aleph}, \mathsf{U}_{1,x}, \\
& \qquad \big(\mathsf{U}_{2,x} \cup \mathsf{F}_{1,i_{b_1+1}} \cup ... \cup \mathsf{F}_{b_2,\aleph}\big)\Big)\Big] \\
\leq \quad & \sum_{i_1,...i_\aleph \in \mathbb{N}^+}^{i_1+...+i_\aleph=b} \sum_{x \in [sec]_1} Pr\Big[\Big(\mathsf{V}_{1,i_1}, ..., \mathsf{V}_{\aleph,i_\aleph}, \mathsf{U}_{1,x}, \\
& \qquad \big(\mathsf{U}_{2,x} \cup \mathsf{F}_{1,i_{b_1+1}} \cup ... \cup \mathsf{F}_{b_2,\aleph}\big)\Big)\Big] \\
\leq \quad & \sum_{i_1,...i_\aleph \in \mathbb{N}^+}^{i_1+...+i_\aleph=b} \sum_{x \in [sec]_1} \Big(Pr\big[\mathsf{V}_{1,i_1}, ..., \mathsf{V}_{\aleph,i_\aleph}, \mathsf{U}_{1,x}, \mathsf{U}_{2,x}\big] + \\
& \qquad Pr\big[\mathsf{V}_{1,i_1}, ..., \mathsf{V}_{\aleph,i_\aleph}, \mathsf{U}_{1,x}, \big(\mathsf{F}_{1,i_{b_1+1}} \cup ... \cup \mathsf{F}_{b_2,\aleph}\big)\big]\Big)
\end{aligned}$$

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

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

An easy calculation using the hypergeometric distribution shows that

$$\begin{aligned}
& Pr\big[\mathsf{V}_{\aleph,i_\aleph} \mid \mathsf{V}_{1,i_1}, ..., \mathsf{V}_{\aleph-1,i_{\aleph-1}}\big] \cdots Pr\big[\mathsf{V}_{1,i_1}\big] = \\
& \prod_{k \in [\aleph-1]_0} \frac{\binom{b - \sum_{j=0}^{k-1} i_j}{i_k} \cdot \binom{T - \sum_{j=0}^{k-1} |V_j| - (b - \sum_{j=0}^{k-1} i_j)}{|V_k| - i_k}}{\binom{T - \sum_{j=0}^{k-1} |V_j|}{|V_k|}}
\end{aligned}$$

The value $Pr\big[\mathsf{U}_{1,x} \mid \mathsf{V}_{1,i_1}, ..., \mathsf{V}_{\aleph,i_\aleph}\big]$ 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 $0, 1, ..., i_j$ 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 $Pr\big[\big(\mathsf{F}_{1,i_{b_1+1}} \cup ... \cup \mathsf{F}_{b_2,\aleph}\big) \mid \mathsf{V}_{1,i_1}, ..., \mathsf{V}_{\aleph,i_\aleph}, \mathsf{U}_{1,x}\big]$ follows from [33] as

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

Finally, using Problem 1 we observe that

$$\begin{aligned}
& Pr\big[\mathsf{U}_{2,x} \mid \mathsf{V}_{1,i_1}, ..., \mathsf{V}_{\aleph,i_\aleph}, \mathsf{U}_{1,x}\big] = \\
& \Big(1 - Pr\Big[r + s = x \mid \boldsymbol{M} \overset{\$}{\leftarrow} \mathcal{M}_{\sqrt{t},\sqrt{t},x} \land (r, s) \leftarrow matrixGame(\boldsymbol{M}, \sqrt{t}, \sqrt{t})\Big]\Big)^{b_2}
\end{aligned}$$

in the case of strong extraction (for weak extraction, adjust the definition of *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 (due to [10]) which follows the outline of [14]. For all turing machines mentioned here, we assume that sec, λ 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 $\mathtt{P}_1, ..., \mathtt{P}_n$ 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 $\mathtt{P}_{Bad} \subset \{\mathtt{P}_1, ..., \mathtt{P}_n\}$. The size of this subset and the capabilites that $\mathtt{A}$ has is described by the *adversarial model*. In this paper, 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 $\mathtt{P}_{Bad}$ 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 $\mathtt{P}_i \notin \mathtt{P}_{Bad}$ does not know which other parties they can trust or not.
Active adversaries have full control over $\mathtt{P}_{Bad}$. 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 a $\mathcal{F}$). Such a functionality is a PPT iTM that $\mathtt{P}_1, ..., \mathtt{P}_n$ 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 $\mathtt{P}_1, ..., \mathtt{P}_n$ 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, with a $\mathcal{P}$ for procedure.
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 $\mathcal{S}$) 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 $\mathcal{S}$ for each $\mathtt{A}$ that is constrained by a fixed adversarial model). This simulator mimics some $\tilde{\mathtt{P}_i}$ in place of the $\mathtt{P}_i \notin \mathtt{P}_{Bad}$ and functionalities available during runtime. At the same time $\mathcal{S}$ 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 $\mathtt{P}_1, ..., \mathtt{P}_n$ 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 $(\mathtt{A}, \mathcal{S}, \mathcal{F})$ 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 *sec*) or *perfectly secure*(if the distributions are identical).