Publicly Auditable Secure Multi-Party Computation*

Carsten Baum†, Ivan Damgård‡, and Claudio Orlandi§

{cbaum,ivan,orlandi}@cs.au.dk
Aarhus University, Denmark

Abstract. In the last few years, the efficiency of secure multi-party computation (MPC) increased in several orders of magnitudes. However, this alone might not be enough if we want MPC protocols to be used in practice. A crucial property that is needed in many applications is that everyone can check that a given (secure) computation was performed correctly – even in the extreme case where all the parties involved in the computation are corrupted, and even if the party who wants to verify the result was not participating. This is especially relevant in the *clients-servers* setting, where many clients provide input to a secure computation performed by a few servers. An obvious example of this is electronic voting, but also in many types of auctions one may want independent verification of the result. Traditionally, this is achieved by using non-interactive zero-knowledge proofs during the computation.

A recent trend in MPC protocols is to have a more expensive preprocessing phase followed by a very efficient online phase, e.g., the recent so-called SPDZ protocol by Damgård et al. Applications such as voting and some auctions are perfect use-case for these protocols, as the parties usually know well in advance when the computation will take place, and using those protocols allows us to use only cheap information-theoretic primitives in the actual computation. Unfortunately no protocol of the SPDZ type supports an audit phase.

In this paper, we show how to achieve efficient MPC with a public audit. We formalize the concept of *publicly auditable secure computation* and provide an enhanced version of the SPDZ protocol where, even if all the servers are corrupted, anyone with access to the transcript of the protocol can check that the output is indeed correct. Most importantly, we do so without significantly compromising the performance of SPDZ i.e. our online phase has complexity approximately twice that of SPDZ.

Keywords: Efficient Multi-Party Computation, Public Verifiability, Electronic Voting.

*This is the full version of the paper which was published under the same title at the 9th Conference on Security and Cryptography for Networks (SCN 2014).

†Partially supported by the European Research Commission Starting Grant 279447. Work done in part while visiting IDC Herzliya, supported by the ERC under the EU's Seventh Framework Programme (FP/2007-2013) ERC Grant Agreement n. 307952.

‡Supported by the Danish National Research Foundation, the National Science Foundation of China (under the grant 61061130540) and also from the CFEM research center within which part of this work was performed.

§Supported by The Danish Council for Independent Research (DFF).

1 Introduction

During the last few years MPC has evolved from a purely theoretical to a more practical tool. Several recent protocols (e.g. BeDOZa [7], TinyOT [33] and the celebrated SPDZ [20,18]) achieve incredible performance for the actual function evaluation, even if all but one player are actively corrupted. This is done by pushing all the expensive cryptographic work into an offline phase and using only simple arithmetic operations during the online phase[1]. Since these protocols allow the evaluation of an arbitrary circuit over a finite field or ring, one can in particular use these protocols to implement, for instance, a *shuffle-and-decrypt* operation for a voting application or the function that computes the winning bid in an auction. It is often the case that we know well in advance the time at which a computation is to take place, and in any such case, the aforementioned protocols offer very good performance. In fact the computational work per player in the SPDZ protocol is comparable to the work one has to perform to compute the desired function in the clear, with no security.

However, efficiency is not always enough: if the result we compute securely has large economic or political consequences, such as in voting or auction applications, it may be required that correctness of the result can be verified later. Ideally, we would want that this can done even if all parties involved in the computation are corrupted, and even if the party who wants to verify the result was not involved in the computation.

The traditional solution to this is to ask every player to commit to all his secret data and to prove in zero-knowledge for every message he sends, that this message was indeed computed according to the protocol. If a common reference string is available, we can use non-interactive zero-knowledge proofs, which allow anyone to verify the proofs and hence the result at any later time. However, this adds a very significant computational overhead, and would lead to a horribly inefficient protocol, compared to the online phase of SPDZ, for instance.

It is therefore natural to ask whether it is possible to achieve the best of both worlds and have *highly efficient MPC protocols with a high-speed online phase that are auditable*, in the sense that everyone who has access to the transcripts of the protocol can check if the result is correct *even when all the servers are corrupted.* In this work we answer this question in the affirmative.

1.1 Contributions and Technical Overview

The model. We will focus on client-server MPC protocols, where a set of parties (called the input parties) provide inputs to the actual working parties, who run the MPC protocol among themselves and make the output public[2]. We will focus on the setting of MPC protocols for dishonest majority (and static corruptions): as long as there is one honest party we can guarantee privacy of the inputs and correctness of the results, but we can neither guarantee termination nor fairness. We will enhance the standard network model with a *bulletin board* functionality. Parties are allowed to exchange messages privately, but our protocol will instruct them also to make part of their conversation public.

Auditable MPC. Our first contribution is to provide a formal definition of the notion of *publicly auditable MPC* as an extension of the classic formalization of secure function evaluation. We require correctness and privacy when there is at least one honest party, and in addition ask that anyone, having only access to the transcript of the computation published on the bulletin board, can check the correctness of the output. This is formalized by introducing an extra, non-corruptible party (the *auditor*) who can ask the functionality if the output was correct or not. We stress that the auditor does not need to be involved (or even exist!) before

[1]The offline phase is independent from the inputs and the circuit to be computed – only an upper bound on the number of multiplication gates is needed.

[2]Note that the sets need not be distinct, and using standard transformations we can make sure that the servers do not learn the inputs nor the output of the computation (think of the inputs/output being encrypted or secret shared).

and during the protocol. The role of the auditor is simply to check, once the computation is over, whether the output was computed correctly or not.[3]

SPDZ recap. Given the motivation of this work, we are only interested in the notion of auditable MPC if it can be achieved efficiently. Therefore our starting point is one of the most efficient MPC protocols for arithmetic circuits with a cheap, information-theoretic online phase, namely SPDZ.

In a nutshell SPDZ works as follows: at the end of the offline phase all parties hold additive shares of multiplicative triples (x, y, z) with $z = x \cdot y$. Now the players can use these preprocessed triples to perform multiplications using only linear operations over the finite field (plus some interaction). Moreover, these linear operations can now be performed locally and are therefore essentially for free. However an adversary could send the honest parties a share that is different from what he received at the end of the offline phase. To make sure this is not the case, SPDZ adds information-theoretic MACs of the form $\gamma = \alpha \cdot x$ to each shared value x, where both the MAC γ and the key α are shared among the parties. These MACs are trivially linear and can therefore *follow the computation.* Once the output is reconstructed, the MAC keys are also revealed and the MACs checked for correctness, and in the case the check goes through, the honest parties accept the output.

Auditable SPDZ. In order to make SPDZ auditable, we enhance each shared value x with a Pedersen commitment $g^x h^r$ to x with randomness r. The commitment key (g, h) comes from a common reference string (CRS), such that even if all parties are corrupted, those commitments are still (computationally) binding. To allow the parties to open their commitments, we provide them also with a sharing of the randomness r (each party already knows a share of x). This new *representation* of values is still linear and is therefore compatible with the existing SPDZ framework. During the computation phase, the parties ignore the commitments (they are created during the offline phase, and only the openings must be sent to $\mathcal{F}_{\mathsf{Bulletin}}$) and it will be the job of the auditor to use the linear properties of the commitments to verify that each step of the computation was carried out correctly. Clearly the *offline phase* of SPDZ needs to be modified, in order to produce the commitments to be used by the auditor. Moreover, we have to make this preprocessing step auditable as well.

An example application: Low-latency voting from MPC. Our work can be seen as a part of a recent trend in understanding how generic MPC protocols perform (in terms of efficiency) in comparison to special-purpose protocols (see [27,17] for a discussion on private-set intersection). A notable example of *special purpose* secure computation protocols are mixed-networks (mix-nets), first introduced by Chaum [14]. Here we show how our publicly auditable version of SPDZ compares favorably with mix-nets in terms of latency.

In mix-nets a number of clients submit their encrypted inputs to some servers, who jointly shuffle and decrypt the inputs in such a way that no one should be able to link the input ciphertexts with the output plaintexts, if at least one of the shuffling servers is honest. Mix-nets are of prime importance in electronic voting (like e.g. the *Helios* system [1]). A disadvantage of mix-nets is that they are *inherently sequential*: server i cannot start shuffling before receiving the output of the shuffle performed by server $i-1$. Now, given that the voter's privacy depends on the assumption that there is at least 1 uncorrupted server (out of n), it is desirable to increase the number of parties involved in the shuffle as much as possible. However, when using mix-nets the latency of the protocol is linear in n, and therefore increasing n has a very negative impact on the total efficiency of the protocol, here measured by the time between the last voter casts his vote and the output of the election is announced. We argue here that implementing a shuffle using a generic protocol like SPDZ makes the latency independent of the number of servers performing the shuffle.

More formally, let n be the number of servers and m the number of input ciphertexts. The *computational latency* of mix-nets, here defined as the time we have to wait before all servers have done their computational

[3]In terms of feasibility, auditable MPC can be achieved by compiling a strong semi-honest protocol with NIZKs – a semi-honest MPC protocol alone would not suffice as we cannot force the parties to sample uniform randomness, nor can we trust them to force each other to do so by secure coin-tossing when everyone is corrupted. However, this would not lead to a very practical solution.

work, will be at least $O(n \cdot m \cdot \lambda^3)$ where λ is the computational security parameter.[4] Using SPDZ, the computational latency is $O(m \cdot \log(m) \cdot \kappa^2)$,[5] since the *total* complexity of SPDZ is linear in n and the servers work in parallel (κ is the statistical security parameter). Therefore mix-nets are more expensive by a factor of $\left(\frac{n}{\log m} \cdot \frac{\lambda^3}{\kappa^2}\right)$: this is a significant speed-up when n grows – note also that typical values of λ for public-key cryptography can be one or two orders of magnitudes greater than typical values for a statistical security parameter κ (only field operations are performed during the SPDZ online phase). Clearly, to verify the impact in practice one would have to implement both approaches and compare them.

1.2 Related Work

For certain applications, there already exist *auditable protocols.* The idea is known in the context of e.g. electronic voting as *public verifiability*, and can also be found concerning online auctions and secret sharing. To the best of our knowledge, the term *public verifiability* was first used by Cohen and Fischer in [15]. Widely known publicly auditable voting protocols are those of Schoenmakers [37] and Chaum et al. [13] and the practical Helios [1]. Also stronger notions for voting protocols have been studied, see e.g. [36,31,39]. Verifiability also appeared for secret sharing schemes [37,23,38] and auctions [32,35]. We refer the reader to the mentioned papers and the references therein for more information on these subjects. It is crucial to point out that our suggested approach is not just another voting protocol – instead we lift verifiability to arbitrary secure computations. In this setting, the notion of public verifiability has not been studied, with the exception of [21], where the author presents a general transformation that turns *universally satisfiable* protocols into instances that are *auditable* in our sense. This transformation is general and slows down the computational phase of protocols, whereas our approach is tailor-made for fast computations.

In publicly verifiable delegation of computation (see e.g. [24,22] and references therein) a computationally limited device delegates a computation to the cloud and wants to check that the result is correct. Verifiable delegation is useless unless verification is more efficient than the evaluation. Note that in some sense our requirement is the opposite: we want our workers to work as little as possible, while we are fine with asking the auditor to perform more expensive computation.

External parties have been used before in cryptography to achieve otherwise impossible goals like fairness [29], but in our case *anyone can be the auditor* and does not need to be online while the protocol is executed. This is a qualitative difference with most of the other semi-trusted parties that appear in the literature. The work by Asharov and Orlandi [4] investigated an enhanced notion of covert security, that allows anyone to determine if a party cheated or not given the transcript of the protocol – the goal of our notion is different, as we are interested in what happens when *all* parties are corrupted.

2 The Model

To formalize auditable MPC we add a new party $\mathcal{P}^A$ to the standard MPC model. This new party *only performs the auditing* and does not need to participate during the offline or the online phase. This auditor does not even have to exist when the protocol is executed, but he can check the correctness of a result based on a protocol transcript. This *formal hack* makes it possible to guarantee correctness even if everyone participating in the computation is corrupted[6].

As mentioned, we put ourselves in the client-server model, so the parties involved in an auditable MPC protocols are:

The input parties: We consider m parties $\mathcal{P}^I_1, \ldots, \mathcal{P}^I_m$ with inputs $(x_1, \ldots, x_m)$.

[4]The λ^3 factor is there because of the rerandomization step that is crucially done in every mix-net. Using "onions" of encryptions would not be more efficient.

[5]The $m \cdot \log m$ factor comes from the optimal shuffle of Ajtai et al. [3].

[6]We are not adding a semi-trusted third party to the actual protocol: our guarantee is that if there exist at least one honest party in the universe who cares about the output of the computation, that party can check at any time that the output is correct.

The computing parties: We consider n parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$ that participate in the computation phase. Given a set of inputs $x_1, \ldots, x_m$ they compute an output $y = \mathcal{C}(x_1, \ldots, x_m)$ for some circuit $\mathcal{C}$ over a finite field $\mathbb{F}$. Note that $\{\mathcal{P}_1^I, \ldots, \mathcal{P}_m^I\}$ and $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ may not be distinct.

The auditor: After the protocol run is over, anyone acting as the auditor $\mathcal{P}^A$ can retrieve the transcript of the protocol τ from the bulletin board and (using only the circuit $\mathcal{C}$ and the output y) determine if the result is valid or not.

In the case where at least one party is honest, we require the same security guarantees as in normal MPC with a dishonest majority. However, these standard security notions do not give any guarantee in the *fully malicious* setting, i.e. when all parties are corrupted. We tweak the standard notions slightly and ask for an additional property, called *auditable correctness*.

This notion captures the fact that in the fully malicious case, the input cannot be kept secret from $\mathcal{A}$. But we still want to prove that if the computing parties deviate from the protocol, then they will be caught by $\mathcal{P}^A$, who has access to the transcript that is stored on a bulletin board $\mathcal{F}_{\mathsf{Bulletin}}$.

More formally, our definition for auditable correctness is as follows:

Definition 1 (Auditable Correctness). *Let $\mathcal{C}$ be a circuit, $x_1, \ldots, x_m$ be inputs to $\mathcal{C}$, y be a potential output of $\mathcal{C}$ and τ be a protocol transcript for the evaluation of the circuit $\mathcal{C}$. We say that an MPC protocol satisfies* Auditable Correctness *if the following holds: the auditor $\mathcal{P}^A$ with input τ outputs* ACCEPT *y with overwhelming probability if the circuit $\mathcal{C}$ on input $x_1, \ldots, x_m$ produces the output y. At the same time the auditor $\mathcal{P}^A$ will return* REJECT *(except with negligible probability) if τ is not a transcript of an evaluation of $\mathcal{C}$ or if $\mathcal{C}(x_1, \ldots, x_m) \neq y$.*

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

Initialize: On input $(\mathsf{Init}, \mathcal{C}, \mathbb{F})$ from all parties (where $\mathcal{C}$ is a circuit with m inputs and one output, consisting of addition and multiplication gates over $\mathbb{F}$):

(1) Wait until $\mathcal{A}$ sends the sets $\widehat{\mathcal{I}} \subseteq \mathcal{I}$ (corrupted input parties) and $\widehat{\mathcal{P}} \subseteq \mathcal{P}$ (corrupted computing parties).

Input: On input $(\mathsf{Input}, \mathcal{P}_i^I, id_x, x)$ from $\mathcal{P}_i^I$ and $(\mathsf{Input}, \mathcal{P}_i^I, id_x, ?)$ from all parties $\mathcal{P}_j$, with id_x a fresh identifier and $x \in \mathbb{F}$:

(1) Store (id_x, x).
(2) If $|\widehat{\mathcal{P}}| = n$, send $(\mathsf{Input}, \mathcal{P}_i^I, id_x, x)$ to $\mathcal{A}$.

Compute: On input $(\mathsf{Compute})$ from all parties $\mathcal{P}_j$:

(1) If an input gate of $\mathcal{C}$ has no value assigned, stop here.
(2) Compute $y_c = \mathcal{C}(x_1, \ldots, x_m)$.
(3) Send y_c to $\mathcal{A}$ and wait for y^* from $\mathcal{A}$. If $0 < |\widehat{\mathcal{P}}| < n$, the functionality accepts only $y^* \in \{\perp, y_c\}$. If $|\widehat{\mathcal{P}}| = n$, any value $y^* \in \mathbb{F} \cup \{\perp\}$ is accepted.
(4) Send (Output, y^*) to all parties $\mathcal{P}_j$.

Audit: On input (Audit, y) from $\mathcal{P}^A$ (where $y \in \mathbb{F}$), and if **Compute** was executed, the functionality does the following:

```
if $y_c = y^* = y$ then output ACCEPT $y$.
if $y^* = \perp$ then output NO AUDIT POSSIBLE.
if $y_c \neq y^*$ or $y \neq y^*$ then output REJECT.
```

Fig. 1: $\mathcal{F}_{\mathsf{Online}}$: Ideal functionality that describes MPC with audit.

The above definition gives rise to a new ideal functionality which we provide in Fig. 1. To simplify the exposition, $\mathcal{F}_{\mathsf{Online}}$ is only defined for one output value y. This can easily be generalized.

Note that we only defined our $\mathcal{F}_{\mathsf{Online}}$ for deterministic functionalities. The reason for this is that when all parties are corrupted, even the auditor cannot check whether the players *followed the protocol correctly* in the sense of using real random tapes. This can be solved (using standard reductions) by letting the input parties contribute also random tapes and define the randomness used by the functionality as the XOR of those random tapes – but in the extreme case where all the input parties are corrupted this will not help us.

3 Online Phase

Our setup. Let $p \in \mathbb{N}$ be a prime and $\mathbb{G}$ be some Abelian group (in multiplicative notation) of order p where the *Discrete Logarithm Problem*(DLP) is hard to solve (with respect to a given computational security parameter λ). The MPC protocol will evaluate a circuit $\mathcal{C}$ over $\mathbb{F} = \mathbb{Z}_p$ with $p \geq 2^\kappa$ (where κ is a statistical security parameter), whereas we use the group $\mathbb{G}$ to ensure auditability. We let $g, h \in \mathbb{G}$ be two generators of the group $\mathbb{G}$ where h is chosen such that $\log_g(h)$ is not known (e.g. based on some CRS). For two values $x, \tilde{x} \in \mathbb{F}$, we define $pc(x, \tilde{x}) := g^x h^{\tilde{x}}$ where we use $\tilde{x}$ to denote the randomness used in a commitment to the value x.

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

Store: On input $(\mathsf{store}, id, i, msg)$ from $\mathcal{P}_i$, where id was not assigned yet, store (id, i, msg).
Reveal IDs: On input $(\mathsf{reveal\_all})$ from party $\mathcal{P}_i$ reveal all assigned id-values to $\mathcal{P}_i$.
Reveal message: On input $(\mathsf{get\_message}, id)$ from $\mathcal{P}_i$, the functionality checks whether id was assigned already. If so, then it returns (id, j, msg) to $\mathcal{P}_i$. Otherwise it returns $(id, \perp, \perp)$.

Fig. 2: $\mathcal{F}_{\mathsf{Bulletin}}$: Ideal functionality for the bulletin board.

We assume that a secure channel towards the input parties can be established, that a broadcast functionality is available and that we have access to a bulletin board $\mathcal{F}_{\mathsf{Bulletin}}$ (Fig. 2), a commitment functionality $\mathcal{F}_{\mathsf{Commit}}$ and a procedure to jointly produce random values $\mathcal{F}_{\mathsf{Rand}}$ [7]. We use the bulletin board $\mathcal{F}_{\mathsf{Bulletin}}$ to keep track of all those values that are broadcasted. Observe that no information that was posted to $\mathcal{F}_{\mathsf{Bulletin}}$ can ever be changed or erased.

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

Commit: On input $(\mathsf{commit}, v, r, i, j, id_v)$ by $\mathcal{P}_i$, where both v and r are either in $\mathbb{F}$ or $\perp$, and id_v is a unique identifier, it stores (v, r, i, j, id_v) on a list and outputs (i, id_v) to $\mathcal{P}_j$.
Open: On input $(\mathsf{open}, i, j, id_v)$ by $\mathcal{P}_i$, the ideal functionality outputs (v, r, i, j, id_v) to $\mathcal{P}_j$. If $(\mathsf{no\_open}, i, id_v)$ is given by the adversary, and $\mathcal{P}_i \in \widehat{\mathcal{P}}$, the functionality outputs $(\perp, \perp, i, j, id_v)$ to $\mathcal{P}_j$.

Fig. 3: $\mathcal{F}_{\mathsf{Commit}}$: Ideal functionality for commitments.

The $\langle\cdot\rangle_{\mathsf{S}}$-representation of SPDZ All computations during the online phase are done using additively-shared values. In the setting with a dishonest majority, the parties cannot alter such a shared value as it is secured using a secret-shared MAC (with key α). The key α is also additively-shared among the parties, where party $\mathcal{P}_i$ holds share α_i such that $\alpha = \sum_{i=1}^n \alpha_i$.

If a value is put into such form, then we say that it is in $\langle\cdot\rangle_{\mathsf{S}}$-representation.

Definition 2. *Let $x, y, e \in \mathbb{F}$, then the $\langle x\rangle_{\mathsf{S}}$-representation of x is defined as*

$$\langle x\rangle_{\mathsf{S}} = \Big((x_1, \ldots, x_n), (\gamma(x)_1, \ldots, \gamma(x)_n) \Big),$$

[7]The SPDZ protocol already used $\mathcal{F}_{\mathsf{Commit}}$ and the proof of security holds in the ROM, so these are not extra assumptions. We only additionally require the existence of $\mathcal{F}_{\mathsf{Bulletin}}$ and the DLP-hard group $\mathbb{G}$.

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

Let $\mathbb{F}$ be a field such that there exists a PPT TM to efficiently sample values $r \in \mathbb{F}$ uniformly at random.

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

Fig. 4: $\mathcal{F}_{\mathsf{Rand}}$: Functionality to sample randomness.

where $x = \sum_{i=1}^{n} x_i$ *and* $\alpha \cdot x = \sum_{i=1}^{n} \gamma(x)_i$. *Each player* $\mathcal{P}_i$ *will hold his shares* $x_i, \gamma(x)_i$ *of such a representation. Moreover, we define*

$$\begin{aligned}
\langle x \rangle_{\mathsf{S}} + \langle y \rangle_{\mathsf{S}} &= \Big((x_1 + y_1, \ldots, x_n + y_n), (\gamma(x)_1 + \gamma(y)_1, \ldots, \gamma(x)_n + \gamma(y)_n) \Big), \\
e \cdot \langle x \rangle_{\mathsf{S}} &= \Big((e \cdot x_1, \ldots, e \cdot x_n), (e \cdot \gamma(x)_1, \ldots, e \cdot \gamma(x)_n) \Big), \\
e + \langle x \rangle_{\mathsf{S}} &= \Big((x_1 + e, x_2, \ldots, x_n), (\gamma(x)_1 + e \cdot \alpha_1, \ldots, \gamma(x)_n + e \cdot \alpha_n) \Big).
\end{aligned}$$

This representation is linear - if all parties agree upon the (linear) function that should be applied, then they can perform these on the $\langle \cdot \rangle_{\mathsf{S}}$-representations without interaction:

Remark 1. Let $x, y, e \in \mathbb{F}$. We say that $\langle x \rangle_{\mathsf{S}} \hat{=} \langle y \rangle_{\mathsf{S}}$ if the shares of x, y in $\langle x \rangle_{\mathsf{S}}, \langle y \rangle_{\mathsf{S}}$ reconstruct to the same value. Then it holds that

$$\langle x \rangle_{\mathsf{S}} + \langle y \rangle_{\mathsf{S}} \hat{=} \langle x + y \rangle_{\mathsf{S}} \text{ and } e \cdot \langle x \rangle_{\mathsf{S}} \hat{=} \langle e \cdot x \rangle_{\mathsf{S}} \text{ and } e + \langle x \rangle_{\mathsf{S}} \hat{=} \langle e + x \rangle_{\mathsf{S}}.$$

If we later want to reveal a value r inside $\langle x \rangle_{\mathsf{S}}$ such that no party lied about its share, then we have to check the shared MAC. Unfortunately, reconstructing both the secret and the MAC reveals the key and in a distributed protocol, an adversary may be able to forge a different message due to early arrival of messages (we assume rushing adversaries). We therefore introduce a protocol Π_{MacCheck} that verifies the MAC without ever reconstructing the key. It can be found in Fig. 5.

Protocol Π_{MacCheck}

t values $x_1, \ldots, x_t$ have been opened. Each $\mathcal{P}_i$ has a key share α_i and tag shares $\gamma(x_j)_i$ for $j \in [t]$.

(1) The parties use $\mathcal{F}_{\mathsf{Rand}}$ to publicly sample a vector $\boldsymbol{\tau} \xleftarrow{\$} \mathbb{F}^t$.
(2) Each party locally computes $x = \sum_{j=1}^{t} \boldsymbol{\tau}[j] \cdot x_j$.
(3) Each party locally computes $\gamma_i = \sum_{j=1}^{t} \boldsymbol{\tau}[j] \cdot \gamma(x_j)_i$ and $\sigma_i = \gamma_i - \alpha_i \cdot x$.
(4) Each $\mathcal{P}_i$ uses $\mathcal{F}_{\mathsf{Commit}}$ to commit to σ_i as $\langle\langle \sigma_i \rangle\rangle$.
(5) Each $\mathcal{P}_i$ uses $\mathcal{F}_{\mathsf{Commit}}$ to open $\langle\langle \sigma_i \rangle\rangle$ to all parties.
(6) Each party computes and outputs $\sigma = \sum_{i=1}^{n} \sigma_i$.

Fig. 5: Protocol to verify the MACs on opened $\langle \cdot \rangle_{\mathsf{S}}$-representations.

The following can then be shown:

Lemma 1. *Let* $\mathbb{F} = \mathbb{Z}_p$ *for some prime* p. *If all shares were opened correctly and all parties follow the protocol, then* Π_{MacCheck} *will always output* 0. *If one party cheated during the reconstruction of one* x_j *or sent an incorrect tag share, then* Π_{MacCheck} *outputs* $\sigma \neq 0$ *with probability at least* $2/p$.

Proof. See [18, Appendix D.3].

3.1 The $\langle\cdot\rangle_{\mathsf{A}}$-representation

In order to make SPDZ auditable we enhance the way shared values are represented and stored. In a nutshell, we force the computing parties to commit to the inputs, opened values and outputs of the computation. All intermediate steps can then be checked by performing the computation using the data on $\mathcal{F}_{\mathsf{Bulletin}}$. The commitment scheme is information-theoretically hiding, and we will carry both the actual value $\langle x\rangle_{\mathsf{S}}$ as well as the randomness $\langle\tilde{x}\rangle_{\mathsf{S}}$ of the commitment through the whole computation.

The commitment to a value x will be a Pedersen commitment $pc(x,\tilde{x})$ [34]. When we open a $\langle\cdot\rangle_{\mathsf{A}}$-representation, we reconstruct both x and $\tilde{x}$. This way the commitment is also opened (it is already published on $\mathcal{F}_{\mathsf{Bulletin}}$) and everyone can check that it is correct (but the computing parties do not need to do so during the online phase).

Definition 3. *Let $x,\tilde{x}\in\mathbb{F}$ and $g,h\in\mathbb{G}$ where both g,h generate $\mathbb{G}$, then we define the $\langle x\rangle_{\mathsf{A}}$-representation for x as*

$$\langle x\rangle_{\mathsf{A}} = \Big(\langle x\rangle_{\mathsf{S}}, \langle\tilde{x}\rangle_{\mathsf{S}}, pc(x,\tilde{x})\Big),$$

where $\langle x\rangle_{\mathsf{S}}$ is a representation of x as introduced in Def. 2.

Similarly to the linearity of $\langle\cdot\rangle_{\mathsf{S}}$ we can define linear operations on $\langle\cdot\rangle_{\mathsf{A}}$ as follows:

Definition 4. *Let $x,y,\tilde{x},\tilde{y},e\in\mathbb{F}$. Then define*

$$\begin{aligned}\langle x\rangle_{\mathsf{A}}+\langle y\rangle_{\mathsf{A}} &= \Big(\langle x\rangle_{\mathsf{S}}+\langle y\rangle_{\mathsf{S}}, \langle\tilde{x}\rangle_{\mathsf{S}}+\langle\tilde{y}\rangle_{\mathsf{S}}, pc(x,\tilde{x})\cdot pc(y,\tilde{y})\Big),\\ e\cdot\langle x\rangle_{\mathsf{A}} &= \Big(e\cdot\langle x\rangle_{\mathsf{S}}, e\cdot\langle\tilde{x}\rangle_{\mathsf{S}}, pc(x,\tilde{x})^{e}\Big),\\ e+\langle x\rangle_{\mathsf{A}} &= \Big(e+\langle x\rangle_{\mathsf{S}}, \langle\tilde{x}\rangle_{\mathsf{S}}, pc(e,0)\cdot pc(x,\tilde{x})\Big).\end{aligned}$$

We write that two $\langle\cdot\rangle_{\mathsf{A}}$-representations $\langle x\rangle_{\mathsf{A}} = (\langle x\rangle_{\mathsf{S}}, \langle\tilde{x}\rangle_{\mathsf{S}}, c_x)$, $\langle y\rangle_{\mathsf{A}} = (\langle y\rangle_{\mathsf{S}}, \langle\tilde{y}\rangle_{\mathsf{S}}, c_y)$ are identical (up to their MACs), written $\langle x\rangle_{\mathsf{A}} \hat{=} \langle y\rangle_{\mathsf{A}}$, if

$$\langle x\rangle_{\mathsf{S}}\hat{=}\langle y\rangle_{\mathsf{S}} \text{ and } \langle\tilde{x}\rangle_{\mathsf{S}}\hat{=}\langle\tilde{y}\rangle_{\mathsf{S}} \text{ and } c_x = c_y.$$

Remark 2. Let $x,y,e\in\mathbb{F}$. It holds that

$$\langle x\rangle_{\mathsf{A}}+\langle y\rangle_{\mathsf{A}} \hat{=} \langle x+y\rangle_{\mathsf{A}} \text{ and } e\cdot\langle x\rangle_{\mathsf{A}} \hat{=} \langle e\cdot x\rangle_{\mathsf{A}} \text{ and } e+\langle x\rangle_{\mathsf{A}} \hat{=} \langle e+x\rangle_{\mathsf{A}}.$$

In order to multiply two representations, we use Beaver's circuit randomization technique [5], which works because the representation is linear. Interestingly, one does not have to perform the computations on the commitments during the online phase. Instead, *only the $\langle\cdot\rangle_{\mathsf{S}}$-representations are manipulated.*

3.2 Shared randomness from the offline phase

Our online phase relies on the availability of $\langle\cdot\rangle_{\mathsf{A}}$-representations of random values and multiplication triples. In Fig. 6 we define the functionality $\mathcal{F}_{\mathsf{Offline}}$ that describes the behavior and output of the preprocessing. The functionality will, in addition to generating random values and triples, also set up the shared MAC key. In the case that all parties are corrupted, the functionality is allowed to output randomness that is not correct – however, $\mathcal{P}^{A}$ will be able to establish correctness during the audit phase.

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

Initialize: On input $(\mathsf{Init}, \mathbb{F}, \mathbb{G}, \ell)$ from all players, store the SIMD factor ℓ. $\mathcal{A}$ chooses the set of parties $\widehat{\mathcal{P}} \subseteq \mathcal{P}$ he corrupts.

(1) For all $\mathcal{P}_i \in \widehat{\mathcal{P}}$, $\mathcal{A}$ inputs $\alpha_i \in \mathbb{F}$, while for all $\mathcal{P}_i \in \mathcal{P} \setminus \widehat{\mathcal{P}}$, the functionality chooses $\alpha_i \xleftarrow{\$} \mathbb{F}$ uniformly at random.
(2) Set they key $\alpha = \sum_{i=1}^{n} \alpha_i$ and send α_i to $\mathcal{P}_i \in \mathcal{P} \setminus \widehat{\mathcal{P}}$.
(3) Set the flag $\mathsf{cheated} = \bot$.

Audit: On input (Audit), return REJECT if $\mathsf{cheated} = \top$ or if **Initialize**, **Input** or **Triples** was not executed. Else return ACCEPT.

$\mathsf{AuditRep}(\boldsymbol{r}_1, \ldots, \boldsymbol{r}_n, \tilde{\boldsymbol{r}}_1, \ldots, \tilde{\boldsymbol{r}}_n)$**:**

(1) Set $\boldsymbol{r} = \sum_{i=1}^{n} \boldsymbol{r}_i, \tilde{\boldsymbol{r}} = \sum_{i=1}^{n} \tilde{\boldsymbol{r}}_i$.
(2) If $|\widehat{\mathcal{P}}| = n$, $\mathcal{A}$ inputs a vector $\boldsymbol{\Delta}_c \in \mathbb{G}^{\ell}$. If $\boldsymbol{\Delta}_c$ is not the $(1, \ldots, 1)$ vector, set $\mathsf{cheated} = \top$.
If $|\widehat{\mathcal{P}}| < n$ set $\boldsymbol{\Delta}_c$ to the all-ones vector.
(3) Run macros $\langle \boldsymbol{r} \rangle_{\mathsf{S}} \leftarrow \mathsf{ASpdzRep}(\boldsymbol{r}_1, \ldots, \boldsymbol{r}_n)$ and $\langle \tilde{\boldsymbol{r}} \rangle_{\mathsf{S}} \leftarrow \mathsf{ASpdzRep}(\tilde{\boldsymbol{r}}_1, \ldots, \tilde{\boldsymbol{r}}_n)$.
(4) Define $\langle \boldsymbol{r} \rangle_{\mathsf{A}} = (\langle \boldsymbol{r} \rangle_{\mathsf{S}}, \langle \tilde{\boldsymbol{r}} \rangle_{\mathsf{S}}, pc(\boldsymbol{r}, \tilde{\boldsymbol{r}}) \odot \boldsymbol{\Delta}_c)$. Return $\langle \boldsymbol{r} \rangle_{\mathsf{A}}$.

$\mathsf{ASpdzRep}(\boldsymbol{r}_1, \ldots, \boldsymbol{r}_n)$**:**

(1) Set $\boldsymbol{r} = \sum_{i=1}^{n} \boldsymbol{r}_i$.
(2) For $\mathcal{P}_i \in \widehat{\mathcal{P}}$, $\mathcal{A}$ inputs $\boldsymbol{\gamma}(\boldsymbol{r})_i, \boldsymbol{\Delta}_\gamma \in \mathbb{F}^{\ell}$, and for $\mathcal{P}_i \in \mathcal{P} \setminus \widehat{\mathcal{P}}$, choose $\boldsymbol{\gamma}(\boldsymbol{r})_i \xleftarrow{\$} \mathbb{F}^{\ell}$ at random except for $\boldsymbol{\gamma}(\boldsymbol{r})_j$, with j being the smallest index not in $\widehat{\mathcal{P}}$ (if there exists one).
(3) If $|\widehat{\mathcal{P}}| < n$ set

$$\boldsymbol{\gamma}(\boldsymbol{r}) = \alpha \cdot \boldsymbol{r} + \boldsymbol{\Delta}_\gamma \text{ and } \boldsymbol{\gamma}(\boldsymbol{r})_j = \boldsymbol{\gamma}(\boldsymbol{r}) - \sum_{j \neq i=1}^{n} \boldsymbol{\gamma}(\boldsymbol{r})_i,$$

else set $\boldsymbol{\gamma}(\boldsymbol{r}) = \sum_{i=1}^{n} \boldsymbol{\gamma}(\boldsymbol{r})_i$.
(4) Define $\langle \boldsymbol{r} \rangle_{\mathsf{S}} = (\boldsymbol{r}_1, \ldots, \boldsymbol{r}_n, \boldsymbol{\gamma}(\boldsymbol{r})_1, \ldots, \boldsymbol{\gamma}(\boldsymbol{r})_n)$. Return $\langle \boldsymbol{r} \rangle_{\mathsf{S}}$.

Input: This generates ℓ random values for the input.

(1) For each $\mathcal{P}_i \in \mathcal{P} \setminus \widehat{\mathcal{P}}$ choose $\boldsymbol{r}_i, \tilde{\boldsymbol{r}}_i \xleftarrow{\$} \mathbb{F}^{l}$, send these to $\mathcal{P}_i$ and $pc(\boldsymbol{r}_i, \tilde{\boldsymbol{r}}_i)$ to $\mathcal{A}$.
(2) For $\mathcal{P}_i \in \widehat{\mathcal{P}}$, $\mathcal{A}$ inputs $\boldsymbol{r}_i, \tilde{\boldsymbol{r}}_i \in \mathbb{F}^{\ell}$.
(3) Compute and return $\langle \boldsymbol{r} \rangle_{\mathsf{A}} \leftarrow \mathsf{AuditRep}(\boldsymbol{r}_1, \ldots, \boldsymbol{r}_n, \tilde{\boldsymbol{r}}_1, \ldots, \tilde{\boldsymbol{r}}_n)$.

Triples: Generates ℓ multiplication triples.

(1) For $\mathcal{P}_i \in \mathcal{P} \setminus \widehat{\mathcal{P}}$, the functionality samples $\boldsymbol{a}_i, \boldsymbol{b}_i, \tilde{\boldsymbol{a}}_i, \tilde{\boldsymbol{b}}_i \xleftarrow{\$} \mathbb{F}^{\ell}$ at random, sends them to $\mathcal{P}_i$ and $pc(\boldsymbol{a}_i, \tilde{\boldsymbol{a}}_i), pc(\boldsymbol{b}_i, \tilde{\boldsymbol{b}}_i)$ to $\mathcal{A}$.
(2) For $\mathcal{P}_i \in \widehat{\mathcal{P}}$, $\mathcal{A}$ inputs $\boldsymbol{a}_i, \boldsymbol{b}_i, \boldsymbol{c}_i, \tilde{\boldsymbol{a}}_i, \tilde{\boldsymbol{b}}_i, \tilde{\boldsymbol{c}}_i \in \mathbb{F}^{\ell}$. Set $\boldsymbol{a} = \sum_{j=1}^{n} \boldsymbol{a}_j, \boldsymbol{b} = \sum_{j=1}^{n} \boldsymbol{b}_j$.
(3) Let $\mathcal{P}_i$ be an honest party. For each $\mathcal{P}_j \in \mathcal{P} \setminus \widehat{\mathcal{P}}, j \neq i$ sample $\boldsymbol{c}_j, \tilde{\boldsymbol{c}}_j \xleftarrow{\$} \mathbb{F}^{\ell}$. For $\mathcal{P}_i$ set $\boldsymbol{c}_i = \boldsymbol{a} \odot \boldsymbol{b} - \sum_{j \in [n] \setminus \{i\}} \boldsymbol{c}_j$ and $\tilde{\boldsymbol{c}}_i \xleftarrow{\$} \mathbb{F}^{\ell}$.
(4) For each $\mathcal{P}_i \in \mathcal{P} \setminus \widehat{\mathcal{P}}$ send $pc(\boldsymbol{c}_i, \tilde{\boldsymbol{c}}_i)$ to $\mathcal{A}$ and $\boldsymbol{c}_i, \tilde{\boldsymbol{c}}_i$ to $\mathcal{P}_i$. Set $\boldsymbol{c} = \sum_i \boldsymbol{c}_i$.
(5) Run the macros

$$\begin{aligned} \langle \boldsymbol{a} \rangle_{\mathsf{A}} &\leftarrow \mathsf{AuditRep}(\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n, \tilde{\boldsymbol{a}}_1, \ldots, \tilde{\boldsymbol{a}}_n), \\ \langle \boldsymbol{b} \rangle_{\mathsf{A}} &\leftarrow \mathsf{AuditRep}(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n, \tilde{\boldsymbol{b}}_1, \ldots, \tilde{\boldsymbol{b}}_n), \\ \langle \boldsymbol{c} \rangle_{\mathsf{A}} &\leftarrow \mathsf{AuditRep}(\boldsymbol{c}_1, \ldots, \boldsymbol{c}_n, \tilde{\boldsymbol{c}}_1, \ldots, \tilde{\boldsymbol{c}}_n). \end{aligned}$$

(6) Return $\langle \boldsymbol{a} \rangle_{\mathsf{A}}, \langle \boldsymbol{b} \rangle_{\mathsf{A}}, \langle \boldsymbol{c} \rangle_{\mathsf{A}}$.

Fig. 6: $\mathcal{F}_{\mathsf{Offline}}$: Ideal functionality for the offline phase of auditable MPC.

Protocol Π_{Online}

The parties evaluate the circuit $\mathcal{C}$ over $\mathbb{F}$, which has n_I input gates and n_M multiplication gates.

Initialize:

(1) The parties send $(\mathsf{Init}, \mathbb{F}, \mathbb{G}, \ell)$ to $\mathcal{F}_{\mathsf{Offline}}$ and obtain their shares α_i.

(2) Use the random oracle $\mathcal{H}$ with the session ID and the CRS as input to choose two generators $g, h \in \mathbb{G}$.

(3) The parties choose the smallest $n'_I \geq n_I, n'_M \geq n_M$ such that ℓ divides both n'_I, n'_M. Then they and send (Input, n'_I) and (Triple, n'_M) to $\mathcal{F}_{\mathsf{Offline}}$.

Input: $\mathcal{P}_i^I$ inputs a value $x \in \mathbb{F}$. All $\mathcal{P}_j \in \mathcal{P}$ and $\mathcal{P}_i^I$ do the following (using a new random value $\langle r \rangle_{\mathsf{A}}$):

(1) $\langle r \rangle_{\mathsf{A}}$ is privately opened as $r, \tilde{r}$ to $\mathcal{P}_i^I$.

(2) Let c_r be the commitment of $\langle r \rangle_{\mathsf{A}}$ on $\mathcal{F}_{\mathsf{Bulletin}}$. $\mathcal{P}_i^I$ checks that $c_r = pc(r, \tilde{r})$. If not, the protocol is aborted.

(3) $\mathcal{P}_i^I$ broadcasts $m = x - r$ to all $\mathcal{P}_j$ and $\mathcal{F}_{\mathsf{Bulletin}}$.

(4) All players locally compute $\langle x \rangle_{\mathsf{A}} = \langle r \rangle_{\mathsf{A}} + m$. Assign a new id_x to $\langle x \rangle_{\mathsf{A}}$.

Compute: If **Initialize** has been executed and inputs for all input wires of $\mathcal{C}$ have been assigned, evaluate $\mathcal{C}$ gate per gate as follows:

Add: For two values $\langle x \rangle_{\mathsf{A}}, \langle y \rangle_{\mathsf{A}}$ with IDs id_x, id_y:

(1) Let id_z be a fresh ID. Each party locally computes $\langle z \rangle_{\mathsf{A}} = \langle x \rangle_{\mathsf{A}} + \langle y \rangle_{\mathsf{A}}$ and assigns id_z to it. The commitments are excluded from the computation.

Multiply: Multiply two values $\langle x \rangle_{\mathsf{A}}, \langle y \rangle_{\mathsf{A}}$ with IDs id_x, id_y, using the multiplication triple $(\langle a \rangle_{\mathsf{A}}, \langle b \rangle_{\mathsf{A}}, \langle c \rangle_{\mathsf{A}})$. Let id_z be a fresh ID.

(1) The players calculate $\langle \epsilon \rangle_{\mathsf{A}} = \langle x \rangle_{\mathsf{A}} - \langle a \rangle_{\mathsf{A}}, \langle \rho \rangle_{\mathsf{A}} = \langle y \rangle_{\mathsf{A}} - \langle b \rangle_{\mathsf{A}}$. The commitments are excluded from the computation.

(2) The players publicly reconstruct $\epsilon, \rho, \tilde{\epsilon}, \tilde{\rho}$ and send these values to $\mathcal{F}_{\mathsf{Bulletin}}$.

(3) Each player locally calculates $\langle z \rangle_{\mathsf{A}} = \langle c \rangle_{\mathsf{A}} + \rho \langle a \rangle_{\mathsf{A}} + \epsilon \langle b \rangle_{\mathsf{A}} + \epsilon\rho$ and assigns the ID id_z to it. The commitments are excluded from the computation.

Output: The parties open the output $\langle y \rangle_{\mathsf{A}}$. Let $a_1, \ldots, a_t$ be the values opened.

(1) All parties compute $r \leftarrow \Pi_{\mathsf{MacCheck}}(a_1, \ldots, a_t, \tilde{a}_1, \ldots, \tilde{a}_t)$. If $r \neq 0$ then stop.

(2) All parties open the output $\langle y \rangle_{\mathsf{A}}$ towards $\mathcal{F}_{\mathsf{Bulletin}}$.

(3) All parties compute $s \leftarrow \Pi_{\mathsf{MacCheck}}(y, \tilde{y})$ If $s \neq 0$ then stop. Otherwise output y.

Audit:

(1) If the **Output** step was not completed, output NO AUDIT POSSIBLE.

(2) Run **Audit** for $\mathcal{F}_{\mathsf{Offline}}$. If it returns ACCEPT then continue, otherwise output NO AUDIT POSSIBLE.

(3) We follow the computation gates of the evaluated circuit $\mathcal{C}$ in the same order as they were computed. For the i-th gate, do the following:

Input: Let $\langle r \rangle_{\mathsf{A}}$ be the opened value and id_x be the ID of input x. Set $c_{id_x} = pc(m, 0) \cdot c$, where c is the commitment in $\langle r \rangle_{\mathsf{A}}$ and m is the opened difference.

Add: The parties added $\langle x \rangle_{\mathsf{A}}$ with id_x and $\langle y \rangle_{\mathsf{A}}$ with id_y to $\langle z \rangle_{\mathsf{A}}$ with id_z. Set $c_{id_z} = c_{id_x} \cdot c_{id_y}$.

Multiply: The parties multiplied $\langle x \rangle_{\mathsf{A}}$ with id_x and $\langle y \rangle_{\mathsf{A}}$ with id_y (using the auxiliary values $\langle a \rangle_{\mathsf{A}}, \langle b \rangle_{\mathsf{A}}, \langle c \rangle_{\mathsf{A}}, \langle \epsilon \rangle_{\mathsf{A}}, \langle \rho \rangle_{\mathsf{A}}$ with their respective IDs). The output has ID id_z.

(3.1) Set $c_{id_z} = c_{id_c} \cdot c_{id_a}^{\rho} \cdot c_{id_b}^{\epsilon} \cdot pc(\epsilon \cdot \rho, 0)$.

(3.2) Check that $c_{id_x} \cdot c_{id_a}^{-1} = pc(\epsilon, \tilde{\epsilon},)$ and $c_{id_y} \cdot c_{id_b}^{-1} = pc(\rho, \tilde{\rho},)$. If not, output REJECT.

(4) Let y be the output of **Output** and c_y be the commitment for the output value $\langle y \rangle_{\mathsf{A}}$.

If $c_y = pc(y, \tilde{y})$ then output ACCEPT y.

If $c_y \neq pc(y, \tilde{y})$ then output REJECT.

Fig. 7: Π_{Online}: Protocol for the online phase of auditable MPC.

3.3 The online phase

The online phase of our protocol is presented in Fig. 7. To create the transcript, every party puts all values it ever *sends* or *receives* onto $\mathcal{F}_{\mathsf{Bulletin}}$ (except for the private reconstruction of input values)[8].

During the **Initialize** step of Π, the parties set up the secret-shared MAC key α, generate the parameters g, h of the commitment scheme as well as correlated randomness for the computation. Each input party $\mathcal{P}_i^I$ is allowed during **Input** to submit a value to the computation, where a random value is secretly opened to it. It can then check that the commitment on it is correct, and blind its input using this opened value. Observe that this is the only point in the online phase (excluding **Audit**) where a party actually checks a commitment $pc(\cdot,\cdot)$.

Compute uses the linearity of the $\langle\cdot\rangle_{\mathsf{A}}$-representation to perform linear operations on the shared values, and multiplies two representations using the multiplication triples from the preprocessing using the circuit randomization technique. Again, we do not check if reconstructed values actually open the commitments correctly – this is only done by $\mathcal{P}^A$ in the **Audit** phase. In **Output** a result of the computation will be provided, where we check the MACs of the $\langle\cdot\rangle_{\mathsf{S}}$-representations using the protocol in Fig. 5. This is to provide resilience in case of a dishonest majority.

In **Audit** the auditor $\mathcal{P}^A$ will first examine if the output of the offline phase is correct. If so, then he will follow the computation gate by gate as it was done by $\mathcal{P}_1, \ldots, \mathcal{P}_n$, where $\mathcal{P}^A$ checks every opened $\langle\cdot\rangle_{\mathsf{A}}$-representation against the value on $\mathcal{F}_{\mathsf{Bulletin}}$. This will be done until the commitment of $\langle y\rangle_{\mathsf{A}}$, the result of the computation, is reached, which is checked for correctness as well.

4 Security of the Online Phase

We will now prove security for the construction from the previous section in the UC framework, which implies that Π_{Online} fulfills the *auditable correctness* requirement from Def. 1.

Theorem 1. *In the $\mathcal{F}_{\mathsf{Offline}}, \mathcal{F}_{\mathsf{Bulletin}}, \mathcal{F}_{\mathsf{Commit}}$-hybrid model with a random oracle $\mathcal{H}$, the protocol Π_{Online} implements $\mathcal{F}_{\mathsf{Online}}$ with computational security against any static adversary corrupting all parties except $\mathcal{P}^A$ if the DLP is hard in the group $\mathbb{G}$.*

Observe that we will use a programmable RO in the simulator, but this is only for technical reasons as it generates a suitable CRS. This can be replaced by sampling the CRS in the offline phase.

Proof. We prove the above statement by providing two simulators $\mathcal{S}_{\mathsf{OnNormal}}, \mathcal{S}_{\mathsf{OnFull}}$. While the first (Fig. 8) will be used if at least one party is honest, the second (Fig. 9) takes care of the fully malicious setting.

At least one honest party. The simulator runs an instance of Π_{Online} with the players controlled by $\mathcal{A}$ and simulated honest parties. For **Initialize**, **Input**, **Add**, **Multiply** it performs the same steps as in Π_{Online}, only that it uses a fixed input 0 for the simulated honest parties during **Input**. Since every set of at most $n-1$ shares of a value is uniformly random and does not reveal any information about the shared secret, this cannot be distinguished from a real transcript.

During **Output**, we adjust the shares of one simulated honest party to agree with the correct output y from $\mathcal{F}_{\mathsf{Online}}$: the simulator obtained the result y' of the simulated computation, hence it can adjust the share of a simulated honest party. Moreover, it also adjusts the MAC share as depicted in $\mathcal{S}_{\mathsf{OnNormal}}$ using the MAC key α provided by $\mathcal{F}_{\mathsf{Offline}}$. Since for each y there exists only one $\tilde{y}$ that opens the commitment $\langle\!\langle y\rangle\!\rangle$ correctly and we set $y_i', \tilde{y}_i'$ such that the reconstructed value matches it, this is perfectly indistinguishable. This also holds for the $\gamma(\cdot)$-MACs. By the information-theoretic hiding of the $\langle\!\langle\cdot\rangle\!\rangle$-scheme indistinguishability follows. Observe that in the ideal world, the simulator aborts in **Output** if any of the values opened by the dishonest parties was inconsistent, whereas Π_{Online} aborts if Π_{MacCheck} fails. This happens with probability at most $2/p$ and is therefore negligible in κ due to Lemma 1.

[8]This does not break the security, because this is the same information that an $\mathcal{A}$ receives if he corrupts $n-1$ parties.

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

Initialize:

(1) Set up $\mathcal{F}_{\mathsf{Bulletin}}$ and start a local instance Π of Π_{Online} with which the dishonest parties will communicate. Moreover, for all $\mathcal{P}_i \in \mathcal{P} \setminus \widehat{\mathcal{P}}$ simulate an honest party.

(2) Run a copy of $\mathcal{F}_{\mathsf{Offline}}$, with which the dishonest parties and the simulated honest parties communicate through the simulator.

(3) Send $(\mathsf{Init}, \mathcal{C}, \mathbb{F})$ for each $\mathcal{P}_i \in \widehat{\mathcal{P}}$ to $\mathcal{F}_{\mathsf{Online}}$.

(4) Sample a generator $g \in \mathbb{G}$ at random, choose $s \xleftarrow{\$} \mathbb{Z}^*_{|\mathbb{G}|}$ and set $h = g^s$. Set the random oracle $\mathcal{H}$ to output the two generators $g, h \in \mathbb{G}$. Then run this protocol step as in Π_{Online}.

(5) Send $(\mathsf{Init}, \mathbb{F}, \mathbb{G}, l)$ for all simulated $\mathcal{P}_i$. Record the α used by $\mathcal{F}_{\mathsf{Offline}}$.

(6) Send (Input, n'_I) and (Triple, n'_M) as in Π_{Online} for all simulated $\mathcal{P}_i$.

Input: $\mathcal{P}^I_i$ inputs a value x_i.

If $\mathcal{P}^I_i$ is honest then follow Π_{Online} for a default input value $x_i = 0$.

If $\mathcal{P}^I_i$ is dishonest then extract the input value x_i from Π and send $(\mathsf{Input}, \mathcal{P}^I_i, id_x, x_i)$ for $\mathcal{P}^I_i$ and $(\mathsf{Input}, \mathcal{P}^I_j, id_x, ?)$ for all $\mathcal{P}^I_j \in \mathcal{I} \setminus \widehat{\mathcal{I}}$ to $\mathcal{F}_{\mathsf{Online}}$.

Compute: Set $\mathsf{cheated} = \bot$. If **Initialize** has been executed and inputs for all n_I input gates of $\mathcal{C}$ have been provided, evaluate $\mathcal{C}$ gate per gate as follows:

Add: Follow the steps of **Add** in Π_{Online}.

Multiply: Follow the steps of **Multiply** in Π_{Online}. If $\mathcal{A}$ makes the dishonest parties send shares that do not open to the correct opening as expected, set $\mathsf{cheated} = \top$.

Output: Send $(\mathsf{Compute})$ to $\mathcal{F}_{\mathsf{Online}}$. Obtain the output y from $\mathcal{F}_{\mathsf{Online}}$. Simulate Π_{Online} as follows:

(1) Generate correct shares for the simulated honest parties for Π:

(1.1) Let $\mathcal{P}_i$ be a simulated honest party and y' be the output of the simulated protocol. Set $\langle y' \rangle_{\mathsf{A}} = (\langle y' \rangle_{\mathsf{S}}, \langle \tilde{y}' \rangle_{\mathsf{S}}, c = pc(y', \tilde{y}'))$.

(1.2) For $\mathcal{P}_i$ set new shares as follows:

$$y'_i = y'_i + (y - y') \text{ and } \gamma(y)'_i = \gamma(y)'_i + \alpha(y - y').$$

We have $s \neq 0$, so $s^{-1} \bmod p$ exists. Hence we can choose

$$\tilde{y} = (y' - y + s \cdot \tilde{y}')/s \text{ and } \tilde{y}'_i = \tilde{y}'_i + (\tilde{y} - \tilde{y}') \text{ and } \gamma(\tilde{y})'_i = \gamma(\tilde{y})'_i + \alpha \cdot (\tilde{y} - \tilde{y}').$$

(2) Follow the protocol Π_{Online} to check the MACs according to Step (1) of **Output**. If the dishonest parties provide incorrect γ_i in Π_{MacCheck} then set $\mathsf{cheated} = \top$.

(3) If $\mathsf{cheated} = \top$ then let $\mathcal{F}_{\mathsf{Online}}$ deliver $\bot$ to the honest parties and stop.

(4) Send the shares of the simulated honest parties of the output $\langle y' \rangle_{\mathsf{A}}$ to $\mathcal{F}_{\mathsf{Bulletin}}$, i.e. the additive shares $y'_i, \tilde{y}'_i$. If $\mathcal{A}$ does not provide correct shares of $\langle y' \rangle_{\mathsf{A}}$ for all dishonest parties, then set $\mathsf{cheated} = \top$.

(5) Run Π_{MacCheck} as in Π_{Online}. If the dishonest parties send incorrect γ_i then set $\mathsf{cheated} = \top$.

(6) If $\mathsf{cheated} = \top$ then let $\mathcal{F}_{\mathsf{Online}}$ deliver $\bot$ to the honest parties and stop. Else output y.

Audit: Run **Audit** as in Π_{Online} with the malicious players. Then invoke **Audit** in $\mathcal{F}_{\mathsf{Online}}$ and output REJECT if it is the output of $\mathcal{F}_{\mathsf{Online}}$. If not, reveal what $\mathcal{F}_{\mathsf{Online}}$ outputs.

Fig. 8: $\mathcal{S}_{\mathsf{OnNormal}}$: Simulator for the protocol Π_{Online}, honest minority.

If moreover $\mathcal{A}$ decides to stop the execution, then $\mathcal{S}_{\mathsf{OnNormal}}$ will forward this to the ideal functionality and $\mathcal{A}$ will not receive any additional information, as in the real execution. During the **Audit** phase, we also do exactly the same as in the protocol, except when $\mathcal{F}_{\mathsf{Online}}$ outputs REJECT but Π_{Online} outputs ACCEPT y. This means that $\mathcal{A}$ replaced the output y, but Π_{MacCheck} passed successfully. This breaks Lemma 1 as mentioned before.

Fully malicious setting. The intuition behind $\mathcal{S}_{\mathsf{OnFull}}$ is that we let $\mathcal{A}$ send arbitrary messages during the online phase. But since all messages for $\mathcal{F}_{\mathsf{Offline}}$ go through $\mathcal{S}_{\mathsf{OnFull}}$, we extract the used inputs after the fact

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

Initialize:

(1) Set up $\mathcal{F}_{\mathsf{Bulletin}}$ and start a local instance Π of Π_{Online} with which the dishonest parties will communicate. Set the flag $\mathsf{cheated} = \perp$.

(2) Run a copy of $\mathcal{F}_{\mathsf{Offline}}$, with which the dishonest parties communicate through the simulator. Record the α used by $\mathcal{F}_{\mathsf{Offline}}$ during **Initialize**.

Input: $\mathcal{P}_i^I$ inputs a value x_i.

If $\mathcal{P}_i^I$ is honest record it as it is sent by $\mathcal{F}_{\mathsf{Online}}$.

If $\mathcal{P}_i^I$ is dishonest then compute x_i from m on $\mathcal{F}_{\mathsf{Bulletin}}$ and $\langle r \rangle_{\mathsf{A}}$ from the recorded outputs of $\mathcal{F}_{\mathsf{Offline}}$. Then send the input computed from it to $\mathcal{F}_{\mathsf{Online}}$.

Compute: If **Initialize** has been executed and inputs for all input gates of $\mathcal{C}$ have been provided, evaluate $\mathcal{C}$ gate per gate as follows:

Add: Follow the steps of **Add** in Π_{Online}.

Multiply: Follow the steps of **Multiply** in Π_{Online}. If $\mathcal{A}$ makes the dishonest parties send shares that do not open to the correct opening as expected, set $\mathsf{cheated} = \top$.

Output:

(1) Send $(\mathsf{Compute})$ to $\mathcal{F}_{\mathsf{Online}}$.

(2) Follow the protocol Π_{Online} of **Output**. If Π_{Online} aborts or $\mathsf{cheated} = \top$, then set $y^* = \perp$, send it to $\mathcal{F}_{\mathsf{Online}}$ and abort.

(3) If all parties reconstruct an incorrect value y^* then input this value into $\mathcal{F}_{\mathsf{Online}}$.

Audit: Run **Audit** as in Π_{Online} with the malicious players. Then invoke **Audit** in $\mathcal{F}_{\mathsf{Online}}$ and output REJECT if it is the output of $\mathcal{F}_{\mathsf{Online}}$. If not, reveal what $\mathcal{F}_{\mathsf{Online}}$ outputs.

Fig. 9: $\mathcal{S}_{\mathsf{OnFull}}$: Simulator for the protocol Π_{Online}, fully malicious.

which we then can use with $\mathcal{F}_{\mathsf{Online}}$. Observe that, since we cannot guarantee privacy, no inputs must be substituted. In comparison to the honest-minority setting, we let the simulator in the ideal world not abort in **Output** if any of the values sent by the dishonest parties was inconsistent, because $\mathcal{A}$ has the MAC key α anyway and can therefore generate correct $\langle \cdot \rangle_{\mathsf{S}}$-representations.

During **Audit**, we run the protocol Π_{Online} also in the simulator. The difference between both is the output of **Audit** in both worlds. Assume that $\mathcal{F}_{\mathsf{Online}}$ outputs REJECT while Π_{Online} outputs ACCEPT y. Then $\mathcal{A}$ replaced the output y with another value y^* (and also $\tilde{y}^*$) that must open the commitment $\langle\langle y \rangle\rangle$ correctly. But in Step (3) of **Compute**, the simulator already obtained y such that $pc(y, \tilde{y}) = pc(y^*, \tilde{y}^*)$ for some $\tilde{y}$, and one can hence compute $s = \log_g(h)$. Assuming $\mathcal{A}$ achieves this with non-negligible advantage ϵ then we can use $\mathcal{A}$ to compute arbitrary logarithms with essentially the same advantage by adjusting the programmable RO to output the DLP instance we want to break (the reduction may have to use the random self-reducibility of the DLP to generate an instance that $\mathcal{A}$ can break).

5 Offline Phase

We will now provide an implementation of $\mathcal{F}_{\mathsf{Offline}}$, that in particular allows to have an *audit for the offline phase*. For the implementation, we use a somewhat homomorphic encryption scheme (SHE) which we will now introduce.

5.1 Somewhat Homomorphic Encryption

A *threshold somewhat homomorphic encryption* scheme $H = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec}, \oplus, \otimes)$ has a message space $\mathbb{F}$, randomness distribution χ, and we represent ciphertexts known to all parties with the notation

$[\![x]\!] := \mathsf{Enc}_{\mathsf{pk}}(x)$. In addition, H has a predicate

$$\begin{aligned} \mathsf{correct} : \{0,1\}^{n(\lambda)} \times \{0,1\}^{n(\lambda)} \times \{0,1\}^{n(\lambda)} \times \{0,1\}^{n(\lambda)} &\to \{0,1\} \\ (\mathsf{pk}, c, x, r) &\mapsto \mathsf{correct}(\mathsf{pk}, c, x, r), \end{aligned}$$

that maps to 1 if $\mathsf{pk} \overset{\$}{\leftarrow} \mathsf{KG}(1^\lambda), x \in \mathbb{F}, r \overset{\$}{\leftarrow} \chi$ and $c \leftarrow \mathsf{Enc}_{\mathsf{pk}}(x, r)$, but otherwise indicates that c *can be correctly decrypted after certain operations.* This is needed to verify that a ciphertext c, even if generated by a malicious party, fulfills certain criteria. We will later call a ciphertext c correct if the above predicate $\mathsf{correct}$ evaluates to true.

The binary operators $\oplus, \otimes$ then guarantee that, if $\mathsf{correct}$ evaluates to 1 on $[\![x]\!], [\![y]\!]$,

$$\mathsf{Dec}_{\mathsf{sk}}([\![x + y]\!]) = \mathsf{Dec}_{\mathsf{sk}}([\![x]\!]) \oplus \mathsf{Dec}_{\mathsf{sk}}([\![y]\!]) \quad \text{and} \quad \mathsf{Dec}_{\mathsf{sk}}([\![x \cdot y]\!]) = \mathsf{Dec}_{\mathsf{sk}}([\![x]\!]) \otimes \mathsf{Dec}_{\mathsf{sk}}([\![y]\!]).$$

For our purposes, these homomorphic operations only need to support evaluation of circuits with polynomially many additions and multiplicative depth 1, which in practice will be reflected by $\mathsf{correct}$.

Meaningless Keys. The proof of security of the offline phase of [20] relies on an additional property of the cryptosystem, namely the availability of *meaningless keys*. Note that this holds for schemes that are used as H, such as e.g. BGV [9].

Definition 5. *The scheme H has the* meaningless keys property *if there exists an algorithm $\overline{\mathsf{KG}}$ that outputs a meaningless public key, $\overline{\mathsf{pk}}$, such that for $\mathsf{pk} \leftarrow \mathsf{KG}(1^\lambda)$, $\overline{\mathsf{pk}} \leftarrow \overline{\mathsf{KG}}(1^\lambda)$ and any message x:*

$$\mathsf{Enc}(\overline{\mathsf{pk}}, x) \overset{s}{\approx} \mathsf{Enc}(\overline{\mathsf{pk}}, 0)$$
$$\mathsf{pk} \overset{c}{\approx} \overline{\mathsf{pk}},$$

where the distributions are taken over the randomness of $\mathsf{KG}, \overline{\mathsf{KG}}$.

In addition, we require the following interactive protocols that will be used for the preprocessing.

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

Key generation:

(1) Compute $(\mathsf{pk}, \mathsf{sk}) \leftarrow H.\mathsf{KG}(1^\lambda)$.
(2) Let $\mathcal{A}$ choose sk_j for each $\mathcal{P}_j \in \widehat{\mathcal{P}}$.
(3) Generate shares sk_i for all $\mathcal{P}_i \in \mathcal{P} \setminus \widehat{\mathcal{P}}$ consistent with $\mathsf{sk}, (\mathsf{sk}_j)_{j \in \widehat{\mathcal{P}}}$.
(4) Send $(\mathsf{pk}, \mathsf{sk}_i)$ to each honest $\mathcal{P}_i$ and pk to each $\mathcal{P}_j \in \widehat{\mathcal{P}}$. Store $(\mathsf{pk}, \mathsf{sk})$.

Distributed decryption:

(1) Upon receiving $(\mathsf{Decrypt}, \mathsf{pk}, [\![c]\!])$ from all parties check whether there exists a shared key pair $(\mathsf{pk}, \mathsf{sk})$. If not, return $\perp$.
(2) Compute $m \leftarrow H.\mathsf{Dec}_{\mathsf{sk}}([\![c]\!])$ and send m to $\mathcal{A}$. Upon receiving $m^* \in \mathbb{F} \cup \{\perp\}$ from $\mathcal{A}$, send (Result, m^*) to all players.

Designated decryption:

(1) Upon receiving $(\mathsf{Decrypt}, \mathsf{pk}, [\![c]\!], \mathcal{P}_i)$ from all parties, check whether there exists a shared key pair $(\mathsf{pk}, \mathsf{sk})$. If not, return $\perp$. Compute $m \leftarrow H.\mathsf{Dec}_{\mathsf{sk}}([\![c]\!])$.
(2) If $\mathcal{P}_i \in \widehat{\mathcal{P}}$ then send (Result, m) to $\mathcal{P}_i$.
(3) Else, wait for an $m^* \in \mathbb{F} \cup \{\perp\}$ from $\mathcal{A}$. If $m^* \in \mathbb{F}$ send $(\mathsf{Result}, m + m^*)$ to $\mathcal{P}_i$, else send $(\mathsf{Result}, \perp)$.

Fig. 10: $\mathcal{F}_{\mathsf{KeyGenDec}}$: Ideal functionality for distributed key generation and decryption with potential error.

Zero-Knowledge proof of plaintext knowledge. Towards being able to use H in a distributed setting we assume that there exists a protocol Π_{ZKPoPK} called a zero-knowledge proof of plaintext knowledge between a prover $\mathcal{P}_1$ and a verifier $\mathcal{P}_2$ that, for $(\mathsf{pk}, \mathsf{sk}) \overset{\$}{\leftarrow} \mathsf{KG}(1^\lambda)$ has the following properties:

Correctness: If $\mathcal{P}_1$ generates a ciphertext c honestly as $c \leftarrow \mathsf{Enc}_{\mathsf{pk}}(x, r)$ for $x \in \mathbb{F}, r \overset{\$}{\leftarrow} \chi$ and provides these x, r to Π_{KeyGen}, then the protocol outputs 1 with overwhelming probability.

Soundness: If $\mathcal{P}_1$ generated c such that Π_{ZKPoPK} outputs 1 with high probability, then there exists a method to extract (x', r') by rewinding $\mathcal{P}_1$ such that $\mathsf{correct}(\mathsf{pk}, c, x', r') = 1$.

Zero-Knowledge: $\mathcal{P}_2$ learns nothing about x, r from Π_{ZKPoPK}, except for the result of the evaluation of $\mathsf{correct}$.

Such a protocol is called honest-verifier zero-knowledge if it fulfills the above requirements when $\mathcal{P}_2$'s choices are not controlled by $\mathcal{A}$. We will indeed require that Π_{ZKPoPK} is honest-verifier zero-knowledge: in our preprocessing protocols, all parties will sample the verifier's messages with the coin-tossing functionality $\mathcal{F}_{\mathsf{Rand}}$ and the proofs are verified by all parties. This guarantees that the messages sent by the verifier are indeed honestly generated, and at least one party either generates the proof correctly or acts as a verifier.

Distributed key generation and decryption. The distributed key generation protocol outputs a public key pk to all parties, and a secret key share sk_i to each $\mathcal{P}_i$. The distributed decryption protocol then allows the parties to decrypt a public ciphertext so that all parties obtain the output.

The original SPDZ scheme used a distributed decryption protocol as modeled in Fig. 10, and assumed that the key generation was e.g. implemented using another MPC protocol. This distributed decryption protocol has the disadvantage that the resulting m may not be the correct decryption, and a check for correctness of the result must be employed.

Cryptosystems that can be used to instantiate H are for instance the Ring-LWE-based BGV scheme [9,10] as well as the more recent matrix-based cryptosystems like the GSW scheme [25,11].

5.2 Zero-Knowledge proofs

We first give a very simple and inefficient zero-knowledge proof for the following relation

$$R_{CTC} = \left\{ (\boldsymbol{a}, \boldsymbol{w}) \;\middle|\; \begin{array}{c} \boldsymbol{a} = (c_1, c_2, d_1, \ldots, d_\ell, \mathsf{pk}) \wedge \boldsymbol{w} = (\boldsymbol{x}_1, \boldsymbol{x}_2, r_1, r_2) \wedge \\ \boldsymbol{c}_1 = \mathsf{Enc}_{\mathsf{pk}}(\boldsymbol{x}_1, r_1) \wedge \boldsymbol{c}_2 = \mathsf{Enc}_{\mathsf{pk}}(\boldsymbol{x}_2, r_2) \wedge \\ \mathsf{correct}(\mathsf{pk}, c_1, \boldsymbol{x}_1, r_1) \wedge \mathsf{correct}(\mathsf{pk}, c_2, \boldsymbol{x}_2, r_2) \wedge \\ \left\{ d_k = pc(\boldsymbol{x}_1[k], \boldsymbol{x}_2[k]) \right\}_{k \in [\ell]} \end{array} \right\},$$

which can be found in Fig. 11. It combines two Σ protocols that simultaneously prove knowledge of two ciphertexts and shows that their plaintexts are the messages that $\mathcal{P}_1$ committed to.

The protocol Π_{ZKCom} is correct due to the linearity of H and the commitment scheme[9] and because $\mathcal{P}_1$ only outputs transcripts that fulfill $\mathsf{correct}$. Soundness follows due to the standard soundness of Σ protocols which allows us to extract a witness and because $\boldsymbol{e}$ was chosen from a large enough space so it is computationally infeasible for $\mathcal{P}_1$ to forge an accepting transcript. Zero-knowledge follows trivially in the programmable random oracle model.

Proposition 1. *The protocol Π_{ZKCom} is a non-interactive zero-knowledge proof of knowledge for the relation R_{CTC} using the programmable RO $\mathcal{H}$.*

Note that this simple proof can easily be made more efficient using the techniques from [16], where one has to adjust the choice of the challenge $\boldsymbol{e}$ accordingly.

[9]Note that we cheated a little, because $\boldsymbol{m}_i^j$ comes from a larger interval than what the message space of the commitment scheme is. One first has to reduce $\boldsymbol{m}_i^j$ to the appropriate interval, but we left this out for simplicity.

Protocol Π_{ZKCom}

The input to the protocol are two ciphertexts $c_1 = \mathsf{Enc}_{\mathsf{pk}}(\boldsymbol{x}_1, r_1), c_2 = \mathsf{Enc}_{\mathsf{pk}}(\boldsymbol{x}_2, r_2)$ where $\boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathbb{F}^\ell$ as well as $\{d_k\}_{k\in[\ell]}$.

(1) For $j \in [\lambda]$, $\mathcal{P}_1$ samples

$$\boldsymbol{m}_1^j, \boldsymbol{m}_2^j \xleftarrow{\$} E(\mathbb{F}^\ell) \text{ and } s_1^j, s_2^j \xleftarrow{\$} E(\chi) \text{ and } \left\{b_k^j = pc(\boldsymbol{m}_1^j[k], \boldsymbol{m}_2^j[k])\right\}_{k\in[\ell]}.$$

(2) $\mathcal{P}_1$ computes $[\![\boldsymbol{m}_i^j]\!] \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\boldsymbol{m}_i^j, s_i^j)$ for $i \in \{1, 2\}$.

(3) $\mathcal{P}_1$ computes

$$\boldsymbol{e} \leftarrow \mathcal{H}\left(\left\{[\![\boldsymbol{m}_1^j]\!]||[\![\boldsymbol{m}_2^j]\!]||\{b_k^j\}_{k\in[\ell]}\right\}_{j\in[\lambda]}||c_1||c_2||\{d_k\}_{k\in[\ell]}\right)$$

where $\boldsymbol{e} \in \{0,1\}^\lambda$.

(4) For $i \in \{1, 2\}, j \in [\lambda], k \in [\ell]$ $\mathcal{P}_1$ computes

$$\boldsymbol{\alpha}_i^j = \boldsymbol{m}_i^j + \boldsymbol{e}[j] \cdot \boldsymbol{x}_i \text{ and } \beta_i^j = s_i^j + \boldsymbol{e}[j] \cdot r_i.$$

(5) Output

$$\left([\![\boldsymbol{m}_1^j]\!], [\![\boldsymbol{m}_2^j]\!], \left\{b_k^j\right\}_{k\in[\ell]}, \boldsymbol{\alpha}_1^j, \boldsymbol{\alpha}_2^j, \beta_1^j, \beta_2^j\right)_{j\in[\lambda]}$$

if the transcript leaks no information[a] about $\boldsymbol{x}_1, \boldsymbol{x}_2, r_1, r_2$, otherwise go to Step (1).

(6) $\mathcal{P}_2$ computes

$$[\![\boldsymbol{\alpha}_i^j]\!] \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\boldsymbol{\alpha}_i^j, \beta_i^j) \text{ and } \boldsymbol{e}' \leftarrow \mathcal{H}\left(\left\{[\![\boldsymbol{m}_1^j]\!]||[\![\boldsymbol{m}_2^j]\!]||\{b_k^j\}_{k\in[\ell]}\right\}_{j\in[\lambda]}||c_1||c_2||\{d_k\}_{k\in[\ell]}\right).$$

(7) $\mathcal{P}_2$ checks for $i \in \{1, 2\}, j \in [\lambda], k \in [\ell]$ that
  (7.1) $[\![\boldsymbol{\alpha}_i^j]\!] = [\![\boldsymbol{m}_i^j]\!] \oplus \boldsymbol{e}'[j] \cdot c_1$,
  (7.2) $\mathsf{correct}(\mathsf{pk}, [\![\boldsymbol{\alpha}_i^j]\!], \boldsymbol{\alpha}_i^j, \beta_i^j) = 1$,
  (7.3) $pc(\boldsymbol{\alpha}_1^j[k], \boldsymbol{\alpha}_2^j[k]) = b_k^j \cdot d_k^{\boldsymbol{e}'[j]}$.
  If one of the checks fails then $\mathcal{P}_2$ outputs *reject*, otherwise *accept*.

[a]Such a check highly depends on the cryptosystem H and we leave it out here. Such a check should be successful for honest instances with probability $1 - 1/\mathsf{poly}(\lambda)$.

Fig. 11: Π_{ZKCom}: NIZK proof for the relation R_{CTC}.

5.3 Resharing plaintexts among parties

In order to compute the secret-shared MAC, we compute the product of the shared value and the secret MAC key α using H and reshare the result. To perform this *resharing* the protocol Reshare as depicted in Fig. 12 is used.

The following statements about Reshare can easily be verified:

Proposition 2. *Assuming H and $\mathcal{F}_{\mathsf{Bulletin}}$, then* Reshare *on input $[\![\boldsymbol{m}]\!]$ has the following properties in the ROM:*

(1) If each $\mathcal{P}_i$ honestly follows the protocol, then $\mathcal{P}_i$ obtains a share $\boldsymbol{m}_i$ such that $\boldsymbol{m} = \sum_i \boldsymbol{m}_i$. For each $\mathcal{P}_i \notin \widehat{\mathcal{P}}$ the share $\boldsymbol{m}_i$ is uniformly random.

(2) Assuming all the NIZKs are correct but the parties act maliciously, then $[\![\boldsymbol{m}']\!]$ is a correct *H ciphertext such that $\boldsymbol{m}' = \sum_i \boldsymbol{m}_i$ and each $\mathcal{P}_i$ knows the share m_i.*

Both statements hold with probability that is overwhelming in λ.

The procedure in Reshare is sufficient as long as no commitments are involved in the resharing. To add them, we introduce the procedure ComReshare which can also be found in Fig. 12.

Procedure Reshare**:** The input to the procedure is $[\![\boldsymbol{m}]\!]$.

(1) Each $\mathcal{P}_i$ samples $\boldsymbol{r}_i \xleftarrow{\$} \mathbb{F}^\ell$. We set $\boldsymbol{r} = \sum_{j=1}^n \boldsymbol{r}_j$.
(2) Each $\mathcal{P}_i$ computes and broadcasts $[\![\boldsymbol{r}_i]\!] \leftarrow \mathsf{Enc}(\boldsymbol{r}_i)$ to all parties and $\mathcal{F}_{\mathsf{Bulletin}}$ together with a NIZK-version of Π_{ZKPoPK}.
(3) The players compute $[\![\boldsymbol{r}]\!] = \bigoplus_{i=1}^n [\![\boldsymbol{r}_i]\!]$, set $[\![\boldsymbol{m}+\boldsymbol{r}]\!] = [\![\boldsymbol{m}]\!] \oplus [\![\boldsymbol{r}]\!]$ and check the ZKPoPKs. If they are not correct, then they abort.
(4) The players decrypt $[\![\boldsymbol{m}+\boldsymbol{r}]\!]$ to $\boldsymbol{\delta}$ using $\mathcal{F}_{\mathsf{KeyGenDec}}$.
(5) All players set $[\![\boldsymbol{m}']\!] \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\boldsymbol{\delta}, 0) \oplus (\bigoplus_{i=1}^n [\![-\boldsymbol{r}_i]\!])$.
(6) $\mathcal{P}_1$ computes $\boldsymbol{m}_1 = \boldsymbol{\delta} - \boldsymbol{r}_1$ and outputs $([\![\boldsymbol{m}']\!], \boldsymbol{m}_1)$. Each other player $\mathcal{P}_i$ sets $\boldsymbol{m}_i = -\boldsymbol{r}_i$ and outputs $([\![\boldsymbol{m}']\!], \boldsymbol{m}_i)$.

Procedure RandShareCom**:** Sample ℓ random commitments plus ciphertexts encrypting their opening. Let def be a flag.

(1) Each $\mathcal{P}_i$ samples $\boldsymbol{r}_i, \tilde{\boldsymbol{r}}_i \xleftarrow{\$} \mathbb{F}^\ell$. We denote $\boldsymbol{r} = \sum_{j=1}^n \boldsymbol{r}_j$ and $\tilde{\boldsymbol{r}} = \sum_{j=1}^n \tilde{\boldsymbol{r}}_j$.
(2) Depending on the flag def:
    if def $= \bot$**:** Each $\mathcal{P}_i$ computes and broadcasts $[\![\boldsymbol{r}_i]\!] \leftarrow \mathsf{Enc}(\boldsymbol{r}_i), [\![\tilde{\boldsymbol{r}}_i]\!] \leftarrow \mathsf{Enc}(\tilde{\boldsymbol{r}}_i)$ to all parties and $\mathcal{F}_{\mathsf{Bulletin}}$.
    if def $= \top$**:** Each $\mathcal{P}_i$ computes $[\![\boldsymbol{r}_i]\!] \leftarrow \mathsf{Enc}(\boldsymbol{r}_i), [\![\tilde{\boldsymbol{r}}_i]\!] \leftarrow \mathsf{Enc}(\tilde{\boldsymbol{r}}_i)$ and commits to them using $\mathcal{F}_{\mathsf{Commit}}$. In a next step, each $\mathcal{P}_i$ opens the commitment to all parties and $\mathcal{F}_{\mathsf{Bulletin}}$.
(3) For each $k \in [\ell]$, each party $\mathcal{P}_i$ publishes $\langle\!\langle \boldsymbol{r}_i[k] \rangle\!\rangle = pc(\boldsymbol{r}_i[k], \tilde{\boldsymbol{r}}_i[k])$ on $\mathcal{F}_{\mathsf{Bulletin}}$ and together with a proof using Π_{ZKCom} that R_{CTC} holds for

$$a = ([\![\boldsymbol{r}_i]\!], [\![\tilde{\boldsymbol{r}}_i]\!], \langle\!\langle \boldsymbol{r}_i[1] \rangle\!\rangle, \ldots, \langle\!\langle \boldsymbol{r}_i[\ell] \rangle\!\rangle, \mathsf{pk}) .$$

(4) Each $\mathcal{P}_i$ checks that the proofs are valid. If not, then abort.
(5) The players locally compute

$$[\![\boldsymbol{r}]\!] = \bigoplus_{i=1}^n [\![\boldsymbol{r}_i]\!] \text{ and } [\![\tilde{\boldsymbol{r}}]\!] = \bigoplus_{i=1}^n [\![\tilde{\boldsymbol{r}}_i]\!] \text{ and } \left\{ \langle\!\langle \boldsymbol{r}[k] \rangle\!\rangle = \prod\nolimits_{i\in[n]} \langle\!\langle \boldsymbol{r}_i[k] \rangle\!\rangle \right\}_{k\in[\ell]} .$$

(6) Output $[\![\boldsymbol{r}]\!], [\![\tilde{\boldsymbol{r}}]\!], \{\langle\!\langle \boldsymbol{r}[k] \rangle\!\rangle\}_{k\in[\ell]}$.

Procedure ComReshare**:** The input to the procedure is $[\![\boldsymbol{m}]\!]$.

(1) Compute $\big([\![\boldsymbol{r}]\!], [\![\tilde{\boldsymbol{r}}]\!], \{\langle\!\langle \boldsymbol{r}[k] \rangle\!\rangle\}_{k\in[\ell]}\big) \leftarrow \mathsf{RandShareCom}(\top)$.
(2) Set $[\![\boldsymbol{m}+\boldsymbol{r}]\!] = [\![\boldsymbol{m}]\!] \oplus [\![\boldsymbol{r}]\!]$.
(3) The players decrypt $[\![\boldsymbol{m}+\boldsymbol{r}]\!]$ using $\mathcal{F}_{\mathsf{KeyGenDec}}$ to obtain $\boldsymbol{\delta}$.
(4) All players set $[\![\boldsymbol{m}']\!] \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\boldsymbol{\delta}, 0) \oplus (\bigoplus_{i=1}^n [\![-\boldsymbol{r}_i]\!])$.
(5) $\mathcal{P}_1$ sets $\boldsymbol{m}_1 = \boldsymbol{\delta} - \boldsymbol{r}_1$ and each other $\mathcal{P}_i$ sets $\boldsymbol{m}_i = -\boldsymbol{r}_i$. Moreover, each $\mathcal{P}_i$ sets $\tilde{\boldsymbol{m}}_i = -\tilde{\boldsymbol{r}}_i$.
(6) For each $k \in [\ell]$ compute $\langle\!\langle \boldsymbol{m}'[k] \rangle\!\rangle = pc((\boldsymbol{\delta})[k], 0) \cdot \Pi_{i\in[n]} \langle\!\langle \boldsymbol{r}_i[k] \rangle\!\rangle^{-1}$.
(7) Each $\mathcal{P}_i$ outputs $([\![\boldsymbol{m}']\!], [\![\tilde{\boldsymbol{r}}]\!], \boldsymbol{m}_i, \tilde{\boldsymbol{m}}_i, \langle\!\langle \boldsymbol{m}'[1] \rangle\!\rangle, \ldots, \langle\!\langle \boldsymbol{m}'[\ell] \rangle\!\rangle)$.

Fig. 12: Procedures Reshare, RandShareCom and ComReshare that reshare a ciphertext, generate a random ciphertext plus commitment and reshare a ciphertext with commitments.

Proposition 3. *Assuming H, $\mathcal{F}_{\mathsf{Bulletin}}$ and a group $\mathbb{G}$ where the DLP is hard, then* ComReshare *has the following properties in the ROM:*

(1) If each $\mathcal{P}_i$ honestly follows ComReshare*, then $\mathcal{P}_i$ obtains $\boldsymbol{m}_i, \tilde{\boldsymbol{m}}_i$ such that $\boldsymbol{m} = \sum_i \boldsymbol{m}_i$, $\tilde{\boldsymbol{m}} = \sum_i \tilde{\boldsymbol{m}}_i$. For each $\mathcal{P}_i \notin \widehat{\mathcal{P}}$ these shares are uniformly random. Moreover, each $\mathcal{P}_i$ obtains $\langle\!\langle \boldsymbol{m}[k] \rangle\!\rangle = pc(\boldsymbol{m}[k], \tilde{\boldsymbol{m}}[k])$.*

(2) If all NIZKs are correct, $[\![\boldsymbol{m}']\!], [\![\tilde{\boldsymbol{r}}]\!]$ are correct *ciphertexts, each $\mathcal{P}_i$ has shares $\boldsymbol{m}_i, \tilde{\boldsymbol{m}}_i$ such that $\boldsymbol{m}' = \sum_i \boldsymbol{m}_i, \tilde{\boldsymbol{r}} = \sum_i \tilde{\boldsymbol{m}}_i$ and $\langle\!\langle \boldsymbol{m}'[k] \rangle\!\rangle = pc(\boldsymbol{m}'[k], \tilde{\boldsymbol{r}}[k])$.*

Both statements hold with probability that is overwhelming in λ.

5.4 Checking the correctness of the output

Whenever ciphertexts must be decrypted in the protocol the functionality $\mathcal{F}_{\mathsf{KeyGenDec}}$ is used. This means that $\mathcal{A}$ will be able to influence the outcome of the decryption process, and we hence have to check the output for correctness. While (according to $\mathcal{F}_{\mathsf{Offline}}$) this is ok for the secret-shared MACs, the functionality requires that multiplication triples should be correct. A procedure to check this is given in Fig. 13.

Procedure DataCheck

The input to the procedure are triples $t_1, \ldots, t_{2k}$ We put the triples into the checking and evaluation vectors $\boldsymbol{C}$ and $\boldsymbol{O}$. For a vector of triples $\boldsymbol{C}$, we want to access all ith $\langle\cdot\rangle_{\mathsf{A}}$-representations in vector form as $\boldsymbol{C}(i)$.

(1) Let $\boldsymbol{C} := (t_1, \ldots, t_k)$ and $\boldsymbol{O} := (t_{k+1}, \ldots, t_{2k})$.
(2) The parties use $\mathcal{H}$ to generate the joint random vector $\boldsymbol{t} \in \mathbb{F}^k$. As seed $\boldsymbol{s}$ they use the public transcript of GenTriples.
(3) Calculate
$$\boldsymbol{\gamma} = \boldsymbol{t} \odot \boldsymbol{O}(1) - \boldsymbol{C}(1) \text{ and } \boldsymbol{\Delta} = \boldsymbol{O}(2) - \boldsymbol{C}(2),$$
then open $\boldsymbol{\gamma}$ and $\boldsymbol{\Delta}$ towards all players and $\mathcal{F}_{\mathsf{Bulletin}}$.
(4) Each party evaluates
$$\boldsymbol{v} = \boldsymbol{t} \odot \boldsymbol{O}(3) - \boldsymbol{C}(3) - \boldsymbol{\Delta} \odot \boldsymbol{C}(1) - \boldsymbol{\gamma} \odot \boldsymbol{C}(2) - \boldsymbol{\Delta} \odot \boldsymbol{\gamma}$$
and commits to its share of $\boldsymbol{v}$ using $\mathcal{F}_{\mathsf{Commit}}$. Check if the commitments hold. If not, then abort.
(5) Each party broadcasts its opening value of the commitment to its share of $\boldsymbol{v}$ and then locally reconstructs $\boldsymbol{v}$.
(6) If $\boldsymbol{v} = \boldsymbol{0}$ then output $\boldsymbol{O}$ (also check that the commitments open to the correct values). Otherwise abort.

Fig. 13: DataCheck: Procedure to check the validity of triples.

Lemma 2. *In the ROM, the test* DataCheck *is correct and an adversary corrupting up to $n-1$ parties can pass the test* DataCheck *for k triples, out of which at least one is not correct, with probability at most $k/|\mathbb{F}|$. If $|\widehat{\mathcal{P}}| = n$ then this holds if one tests the commitents of the $\langle\cdot\rangle_{\mathsf{A}}$-representations in Step* (6).

Proof. Let us consider two triples $(a, b, c) \in \mathbb{F}^3$ and $(x, y, z) \in \mathbb{F}^3$. For $t \cdot (a \cdot b - c) = (x \cdot y - z)$ with $t \overset{\$}{\leftarrow} \mathbb{F}$, the following cases can happen:

(1) (a, b, c) **correct,** (x, y, z) **not:** the adversary has no chance to win.
(2) (a, b, c) **not correct,** (x, y, z) **correct:** there exists only one $t \in \mathbb{F}$ to satisfy the equation, namely $t = 0$. The adversary can only win with probability $1/|\mathbb{F}|$.
(3) **both not correct:** there is only one $t \in \mathbb{F}$ such that the equation holds, hence winning probability is $1/|\mathbb{F}|$.

If $\mathcal{A}$ cheats during this process and t is chosen uniformly at random, then he can cheat for every pair of triples with probability at most $1/|\mathbb{F}|$ as explained above. The result then follows from a union bound over all k instances.

5.5 The offline phase

The overall protocol Π_{Offline} which describes the full offline phase can be found in Fig. 15. We now give a brief, informal outline of its idea which is based on the procedures that were described before.

During **Initialize** the parties will generate a key pair for the encryption scheme H as well as a random MAC key α (also in encrypted form) and parameters g, h for the commitment scheme. **Input** uses the procedure GenRandom from Fig. 14 which generates some random $\langle\cdot\rangle_{\mathsf{A}}$-representations. Therefore, the parties sample uniformly random encryptions together with commitments to the values. They then reshare the

These procedures generate l input values or triples at a time. $[\![\boldsymbol{\alpha}]\!]$ is a ciphertext where every plaintext item equals the MAC key α.

Procedure $\mathsf{GenRandom}$**:**

(1) Compute $\big([\![\boldsymbol{r}]\!], [\![\tilde{\boldsymbol{r}}]\!], \{\langle\!\langle \boldsymbol{r}[k] \rangle\!\rangle\}_{k\in[\ell]}\big) \leftarrow \mathsf{RandShareCom}(\bot)$.
(2) Compute $\gamma_{\boldsymbol{r},i} \leftarrow \mathsf{Reshare}(e_{\boldsymbol{r}} \otimes [\![\boldsymbol{\alpha}]\!])$, $\gamma(\tilde{\boldsymbol{r}})_i \leftarrow \mathsf{Reshare}([\![\tilde{\boldsymbol{r}}]\!] \otimes [\![\boldsymbol{\alpha}]\!])$.
(3) $\big((\boldsymbol{r}_i[k], \gamma_{\boldsymbol{r},i}[k])\,,\,(\tilde{\boldsymbol{r}}_i[k], \gamma(\tilde{\boldsymbol{r}})_i[k])\,,\,(\langle\!\langle \boldsymbol{r}[k] \rangle\!\rangle)\big)$ are now the components of $\langle \boldsymbol{r}[k] \rangle_{\mathsf{A}}$ for $\mathcal{P}_i$.

Procedure $\mathsf{GenTriples}$**:**

(1) Compute

$$([\![\boldsymbol{a}]\!], [\![\tilde{\boldsymbol{a}}]\!], \{\langle\!\langle \boldsymbol{a}[k] \rangle\!\rangle\}_{k\in[\ell]}) \leftarrow \mathsf{RandShareCom}(\bot),$$
$$([\![\boldsymbol{b}]\!], [\![\tilde{\boldsymbol{b}}]\!], \{\langle\!\langle \boldsymbol{b}[k] \rangle\!\rangle\}_{k\in[\ell]}) \leftarrow \mathsf{RandShareCom}(\bot).$$

(2) The parties compute $e_{\boldsymbol{a}\cdot\boldsymbol{b}} \leftarrow e_{\boldsymbol{a}} \otimes e_{\boldsymbol{b}}$ and invoke $\mathsf{ComReshare}(e_{\boldsymbol{a}\cdot\boldsymbol{b}})$. As a result, each party $\mathcal{P}_i$ obtains shares $\boldsymbol{c}_i, \tilde{\boldsymbol{c}}_i$ and all parties obtain a ciphertext $e_{\boldsymbol{c}}, [\![\tilde{\boldsymbol{c}}]\!]$ as well as commitments $\{\langle\!\langle \boldsymbol{c}[k] \rangle\!\rangle = pc(\boldsymbol{c}[k], \tilde{\boldsymbol{c}}[k])\}_{k\in[\ell]}$.
(3) The parties compute

$$\mathsf{Reshare}(e_{\boldsymbol{a}} \otimes [\![\boldsymbol{\alpha}]\!]) \text{ and } \mathsf{Reshare}([\![\tilde{\boldsymbol{a}}]\!] \otimes [\![\boldsymbol{\alpha}]\!]) \text{ to obtain } \langle \boldsymbol{a} \rangle_{\mathsf{A}},$$
$$\mathsf{Reshare}(e_{\boldsymbol{b}} \otimes [\![\boldsymbol{\alpha}]\!]) \text{ and } \mathsf{Reshare}([\![\tilde{\boldsymbol{b}}]\!] \otimes [\![\boldsymbol{\alpha}]\!]) \text{ to obtain } \langle \boldsymbol{b} \rangle_{\mathsf{A}},$$
$$\mathsf{Reshare}(e_{\boldsymbol{c}} \otimes [\![\boldsymbol{\alpha}]\!]) \text{ and } \mathsf{Reshare}([\![\tilde{\boldsymbol{c}}]\!] \otimes [\![\boldsymbol{\alpha}]\!]) \text{ to obtain } \langle \boldsymbol{c} \rangle_{\mathsf{A}}.$$

(4) Output the triples $\langle \boldsymbol{a} \rangle_{\mathsf{A}}, \langle \boldsymbol{b} \rangle_{\mathsf{A}}, \langle \boldsymbol{c} \rangle_{\mathsf{A}}$.

Fig. 14: Procedures $\mathsf{GenRandom}$ and $\mathsf{GenTriples}$ to generate random $\langle\cdot\rangle_{\mathsf{A}}$-representations and multiplication triples.

product of these values with the MAC key α. In order to generate multiplication triples, **Triples** follows a similar pattern but additionally computes a product of the random encryptions and reshares it with commitments[10]. In comparison to **Input** we here additionally have to check that the generated triples are valid multiplication triples.

Audit similarly as in Π_{Online} follows the computation. In addition it ensures that the following properties hold:

(1) All encrypted values and commitments have zero-knowledge proofs.
(2) All zero-knowledge proofs are correct.
(3) Linear operations on commitments are carried out correctly.
(4) The procedure $\mathsf{DataCheck}$ was executed correctly and the multiplicative property holds.
(5) All opened commitments are indeed correctly opened.

The protocol Π_{MacCheck} is not used in the preprocessing in $\mathsf{DataCheck}$, as we directly check for correct openings using the commitments. This is to simplify the protocol.

6 Security of the Offline Phase

In this section, we will give a proof of security of the offline phase.

Theorem 2. *Let H be a somewhat homomorphic cryptosystem. Then Π_{Offline} implements $\mathcal{F}_{\mathsf{Offline}}$ with computational security against any static adversary corrupting at most all parties in the $(\mathcal{F}_{\mathsf{KeyGenDec}}, \mathcal{F}_{\mathsf{Commit}}, \mathcal{F}_{\mathsf{Bulletin}})$-hybrid model with a RO $\mathcal{H}$ if the DLP is hard in $\mathbb{G}$.*

[10]The MACs have no commitment so the normal $\mathcal{P}_{\text{RESHARE}}$ algorithm is sufficient. In comparison, if we reshare the product in a multiplication triple, then we have to take care that we generate a proper commitment for the output of the procedure.

Protocol Π_{Offline}

We define $\mathbf{1} \in \mathbb{F}^{\ell}$ to be the vector that is 1 in every coordinate.

Initialize:

(1) The parties use $\mathcal{F}_{\mathsf{KeyGenDec}}$ to generate a public key pk and a shared private key sk.
(2) Use the random oracle $\mathcal{H}$ with the session ID and the CRS as input to choose two generators $g, h \in \mathbb{G}$.
(3) Each $\mathcal{P}_i$ generates a private $\alpha_i \in \mathbb{F}$. Set $\alpha = \sum_{j=1}^{n} \alpha_j$.
(4) Each $\mathcal{P}_i$ computes and broadcasts $[\![\boldsymbol{\alpha_i}]\!] = \mathsf{Enc}(\mathbf{1} \cdot \alpha_i)$ to each party and $\mathcal{F}_{\mathsf{Bulletin}}$.
(5) Each player $\mathcal{P}_i$ uses the NIZK version of Π_{ZKPoPK} to prove that $[\![\boldsymbol{\alpha_i}]\!]$ is a $\mathsf{correct}$ ciphertext and a diagonal element.
(6) Each player checks the zero-knowledge proofs from all other parties. If one is not correct, then abort.
(7) All players compute $[\![\boldsymbol{\alpha}]\!] = \bigoplus -i = 1^n [\![\boldsymbol{\alpha_i}]\!]$.

Input: Generate n_I random values, where ℓ divides n_I:

(1) Run $(\langle r_1 \rangle_{\mathsf{A}}, \ldots, \langle r_\ell \rangle_{\mathsf{A}}) \leftarrow \mathsf{GenRandom}(\ell)$ n_I/ℓ times.
(2) Return $\langle r_1 \rangle_{\mathsf{A}}, \ldots, \langle r_{n_I} \rangle_{\mathsf{A}}$.

Triples: Generate n_M triples where ℓ divides n_M:

(1) Run $(t_1, \ldots, t_\ell) \leftarrow \mathsf{GenTriples}(\ell)$ $2 \cdot n_M/\ell$ times.
(2) $(t_1, \ldots, t_{n_M}) \leftarrow \mathsf{DataCheck}(t_1, \ldots, t_{2 \cdot n_M})$.
(3) Return $t_1, \ldots, t_{n_M}$.

Audit: If **Compute** was executed successfully, do the following together with $\mathcal{F}_{\mathsf{Bulletin}}$:

(1) Obtain all ids and messages on $\mathcal{F}_{\mathsf{Bulletin}}$.
(2) For every encryption $[\![i]\!]$ and commitment $\langle\!\langle j \rangle\!\rangle$, check whether there exists a transcript of Π_{ZKCom} or Π_{ZKPoPK} that guarantees its correctness. Otherwise return REJECT.
(3) For every transcript of Π_{ZKCom} or Π_{ZKPoPK}, check whether the values for each instance are on $\mathcal{F}_{\mathsf{Bulletin}}$. Otherwise return REJECT.
(4) Run the verifier part for each transcript of Π_{ZKCom}, Π_{ZKPoPK}. If the verifier rejects, return REJECT.
(5) For each value $\langle a \rangle_{\mathsf{A}}$ that was generated with $\mathsf{GenRandom}, \mathsf{GenTriples}$, check whether its commitment can be obtained from the commitments to the shares as in $\mathsf{GenRandom}, \mathsf{GenTriples}$. If not, return REJECT.
(6) Run $\mathsf{DataCheck}$ on the commitments of the triples simulating the invocation of $\mathcal{H}$. If one of the triples that were returned by **Compute** does not open to 0, return REJECT.
(7) Check for every opened value r with randomness $\tilde{r}$ and commitment c whether $c = pc(r, \tilde{r})$. If not, return REJECT.
(8) Return ACCEPT.

Fig. 15: Π_{Offline}: Protocol that performs the preprocessing for auditable MPC.

Proof. Similar to the online phase we again split the simulator in two: $\mathcal{S}_{\mathsf{OffFull}}$ in Fig. 16 handles the fully malicious case, and $\mathcal{S}_{\mathsf{OffNormal}}$ (Figures 17,18) the setting where $|\widehat{\mathcal{P}}| < n$.

Fully malicious setting. In the fully malicious setting, we do not have to simulate any honest party. Moreover, the only output that $\mathcal{Z}$ ever obtains from $\mathcal{F}_{\mathsf{Offline}}$ is in the **Audit** step as in every other step, no honest party interacts with the functionality (except $\mathcal{P}^A$).

We want to catch the adversary if he does something not according to the protocol, which means that he can be caught during **Audit**. We therefore check if the transcript is well-formed (meaning legitimately comes from a Π_{Offline} interaction). Additionally, $\mathcal{S}$ decrypts every ciphertext and checks whether the ciphertext fulfills $\mathsf{correct}$ or the representation is in R_{CTC} instead of verifying the NIZKs. By the definition of H and Prop. 1 this is computationally indistinguishable from correct proofs using $\Pi_{\mathsf{ZKPoPK}}, \Pi_{\mathsf{ZKCom}}$. It also checks whether the decrypted product of the multiplication triples was indeed decrypted correctly. An abort for wrong decryption of triples will happen in **Audit** due to $\mathsf{DataCheck}$ with all but negligible probability. Hence the output of $\mathcal{S}_{\mathsf{OffFull}}$ is computationally indistinguishable from that of Π_{Offline}.

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

Initialize:

(1) The simulator starts a local copy of $\mathcal{F}_{\mathsf{KeyGenDec}}$, $\mathcal{F}_{\mathsf{Bulletin}}$, $\mathcal{F}_{\mathsf{Commit}}$ as well as the random oracle $\mathcal{H}$. Set the flag $\mathsf{cheated} = \perp$.
(2) Send $(\mathsf{Init}, \mathbb{F}, \mathbb{G}, \ell)$ to $\mathcal{F}_{\mathsf{Offline}}$ in the name of all dishonest parties.
(3) The parties use $\mathcal{F}_{\mathsf{KeyGenDec}}$ as in Π to generate a public key pk and shares of a secret key sk for all parties. The simulator obtains pk, sk.
(4) Let the parties perform Step (2) to Step (6). If the proofs are not correct, stop the execution of **Initialize** here. Otherwise, decrypt all $[\![\boldsymbol{\alpha}_i]\!]$ to obtain α_i.
(5) Send the α_i to $\mathcal{F}_{\mathsf{Offline}}$.

Input:

(1) Wait for the transcript of Π for all instances of **Input** between the dishonest parties to be finished.
(2) If the transcript is not well-formed then abort[a].
(3) For each instance of Π_{ZKPoPK} check whether the ciphertext $[\![x]\!]$ fulfills $\mathsf{correct}$. If not, then set $\mathsf{cheated} = \top$.
(4) For each instance of Π_{ZKCom} check if $[\![\boldsymbol{x}]\!], [\![\tilde{\boldsymbol{x}}]\!], \langle\!\langle \boldsymbol{x} \rangle\!\rangle$ fulfills the R_{CTC} relation. If not, then set $\mathsf{cheated} = \top$.
(5) For each instance of **Input** in the transcript call **Input** on $\mathcal{F}_{\mathsf{Offline}}$. Send uniformly random values as inputs for each $\mathcal{P}_i$. If $\mathsf{cheated} = \top$ then send a random $\boldsymbol{\Delta}_c \neq (1, \ldots, 1)$, else send $\boldsymbol{\Delta}_c = (1, \ldots, 1)$.

Triples:

(1) Wait for the transcript of Π for all instances of **Triples** between the dishonest parties to be finished.
(2) If the transcript is not well-formed then abort.
(3) For each instance of Π_{ZKPoPK} check whether the ciphertext $[\![x]\!]$ fulfills $\mathsf{correct}$. If not, then set $\mathsf{cheated} = \top$.
(4) For each instance of Π_{ZKCom} check whether $[\![\boldsymbol{x}]\!], [\![\tilde{\boldsymbol{x}}]\!], \langle\!\langle \boldsymbol{x} \rangle\!\rangle$ fulfills the R_{CTC} relation. If not, then set $\mathsf{cheated} = \top$.
(5) For each call of $\mathsf{ComReshare}$ check if the correct value was announced as decrypted value of the product. If not, then set $\mathsf{cheated} = \top$.
(6) For each instance of **Triples** in the transcript call **Triples** on $\mathcal{F}_{\mathsf{Offline}}$. Send uniformly random values as inputs for each $\mathcal{P}_i$. If $\mathsf{cheated} = \top$ then send a random $\boldsymbol{\Delta}_c \neq (1, \ldots, 1)$, else send $\boldsymbol{\Delta}_c = (1, \ldots, 1)$.

Audit:

(1) Query $\mathcal{F}_{\mathsf{Offline}}$ with (Audit). Return the value of $\mathcal{F}_{\mathsf{Offline}}$ to the requesting party.

[a]By that we mean that the publicly available ciphertexts and opened values on $\mathcal{F}_{\mathsf{Bulletin}}$ follow from Π_{Offline}.

Fig. 16: $\mathcal{S}_{\mathsf{OffFull}}$: Simulator for the protocol Π_{Offline}, fully malicious case.

At least one honest party. The simulator will generate shares for the triples that are uniformly random, and use the decryption key to fit these shares to the commitments that $\mathcal{F}_{\mathsf{Offline}}$ outputs for the honest parties. Hence, the values of the dishonest parties are consistent with those values that the honest parties obtain and are indistinguishable. If $\mathcal{A}$ cheats during the decryption, then the simulator will always abort in $\mathsf{DataCheck}$, which happens with essentially the same probability as the abort of $\mathsf{DataCheck}$ according to Lemma 2. Observe that if the check fails, then $\mathcal{S}$ makes $\mathcal{F}_{\mathsf{Offline}}$ abort which is consistent with the protocol. The other case in which $\mathcal{S}$ aborts is when a ciphertext is not correct or a tuple $([\![a]\!], [\![\tilde{a}]\!], \langle\!\langle a \rangle\!\rangle)$ does not fulfill R_{CTC}. By the assumption that Π_{ZKPoPK} is an honest-verifier zero-knowledge proof and Prop. 1 the additional aborts do only appear with probability negligible in λ.

The simulator will only output ℓ triples but can easily be generalized to the arbitrary case. As in [20] one can then use the availability of a lossy key generation algorithm $\overline{\mathsf{KG}}$ to show that $\mathcal{S}_{\mathsf{OffFull}}$ can work without using the decryption key. The rewinding of the NIZKs to obtain witnesses in Π_{Offline} is not a problem, since our protocol is not *round-based* (in a multi-round protocol, the extractor has to simulate previous rounds as the oracle query used in the proof may have happened in a previous round) and we can use the extraction due to [26].

Simulator $\mathcal{S}_{\mathsf{OffNormal}}$ (part 1)

Initialize:

(1) The simulator starts a local copy of $\mathcal{F}_{\mathsf{KeyGenDec}}$, $\mathcal{F}_{\mathsf{Bulletin}}$, $\mathcal{F}_{\mathsf{Commit}}$ as well as the random oracle $\mathcal{H}$. Set $\mathsf{cheated} = \perp$.

(2) For each $\mathcal{P}_i \in \mathcal{P} \setminus \widehat{\mathcal{P}}$ simulate an honest party in a simulated protocol instance Π_{Offline} with the dishonest parties.

(3) Send $(\mathsf{Init}, \mathbb{F}, \mathbb{G}, \ell)$ to $\mathcal{F}_{\mathsf{Offline}}$ in the name of all $\mathcal{P}_i \in \widehat{\mathcal{P}}$.

(4) The parties use $\mathcal{F}_{\mathsf{KeyGenDec}}$ as in Π to generate a public key pk and shares of a secret key sk for all parties. The simulator obtains pk, sk.

(5) Let the parties perform Step (2) to Step (6). Decrypt the $[\![\boldsymbol{\alpha}_i]\!]$ from the dishonest parties. If the ciphertexts are not formed according to $\mathsf{correct}$ then abort. Otherwise, use the α_i and compute $\alpha = \sum_i \alpha_i$.

Audit:

(1) Query $\mathcal{F}_{\mathsf{Offline}}$ with (Audit). Return the value of $\mathcal{F}_{\mathsf{Offline}}$ to the requesting party.

SReshare: The simulator runs $\mathcal{P}_{\text{RESHARE}}$ with the dishonest parties, but additionally does the following:

- In Step (2), it decrypts all ciphertexts $e_{\boldsymbol{r},i}$ to obtain $\tilde{\boldsymbol{r}}_i$. It moreover checks whether these fulfill $\mathsf{correct}$, if not then abort.
- In Step (4), it obtains $\boldsymbol{\delta}'$ from the protocol and $\boldsymbol{\delta}$ using the secret key to decrypt $[\![\boldsymbol{m} + \boldsymbol{r}]\!]$. It then saves $\boldsymbol{\Delta}_\gamma = \boldsymbol{\delta}' - \boldsymbol{\delta}$ and $\gamma(\boldsymbol{r})_i = -\boldsymbol{r}_i$ for each $\mathcal{P}_i \in \widehat{\mathcal{P}}$ for later.

SRandShareCom: The simulator runs $\mathsf{RandShareCom}(\perp)$ with the dishonest parties , but additionally does the following:

- In Step (1) obtain the commitment c_i for each $\mathcal{P}_i \in \mathcal{P} \setminus \widehat{\mathcal{P}}$ from $\mathcal{F}_{\mathsf{Offline}}$. Sample each $\boldsymbol{r}_i \overset{\$}{\leftarrow} \mathbb{F}^\ell$ and set $\tilde{\boldsymbol{r}}_i$ such that $c_i = pc(\boldsymbol{r}_i, \tilde{\boldsymbol{r}}_i)$ using the trapdoor.
- In Step (2) decrypt $[\![\boldsymbol{r}_i]\!], [\![\tilde{\boldsymbol{r}}_i]\!]$ obtained from $\mathcal{P}_i \in \widehat{\mathcal{P}}$ and send them to $\mathcal{F}_{\mathsf{Offline}}$.
- In Step (3) check whether the relation R_{CTC} holds. If not, then abort.

Input: For each instance of $\mathsf{GenRandom}$:

(1) Use $\mathsf{SRandShareCom}$ where the c_i come from Step (1) of **Input** in $\mathcal{F}_{\mathsf{Offline}}$.

(2) If $\mathsf{SRandShareCom}$ does not abort, send $\boldsymbol{r}_i, \tilde{\boldsymbol{r}}_i$ for each $\mathcal{P}_i \in \widehat{\mathcal{P}}$ to $\mathcal{F}_{\mathsf{Offline}}$.

(3) Simulate the instances of $\mathsf{ASpdzRep}$ using the above inputs using $\mathsf{SReshare}$. Input the $\boldsymbol{\Delta}_\gamma, \gamma(\boldsymbol{r})_i$ for $\mathcal{P}_i \in \widehat{\mathcal{P}}$ to $\mathcal{F}_{\mathsf{Offline}}$.

Fig. 17: $\mathcal{S}_{\mathsf{OffNormal}}$: Simulator for the protocol Π_{Offline}, honest minority.

7 Summary and Open Problems

In this paper, we described how to formally lift MPC into a setting where all servers are malicious. We outlined how this concept can then be securely realized on top of the SPDZ protocol. Though our approach can also be implemented for other MPC protocols, we focused on SPDZ since, even as an publicly auditable scheme, its online phase is very efficient. We note that our protocol would also work for Boolean circuits, but this would introduce a significant slowdown (since the MACs must then be defined as elements of an extension field over $\mathbb{F}_2$, which leads to a significant overhead). It is an interesting future direction to design an efficient auditable protocol optimized for Boolean circuits or circuits over fields with small characteristic.

With respect to online voting, there exist stronger degrees of auditability than we presented. An example is the notion of *universal verifiability* (see e.g. [36,31]) where the auditor must not know the output of the computation. We also do not provide *accountability* (see e.g. Küsters et al. [39]), and leave it as an open question whether similar, efficient protocols can be achieved in this setting.

We leave a working implementation of our scheme as a future work. As our protocol is very similar in structure to the original SPDZ, it should be possible to implement it easily on top of the existing codebase of [18].

Simulator $\mathcal{S}_{\mathsf{OffNormal}}$ (part 2)

$\mathsf{SComReshare}$**:** The simulator runs $\mathsf{ComReshare}$ with the dishonest parties. If $\mathcal{P}_1 \in \widehat{\mathcal{P}}$ then do the following:

- Commit to uniformly random ciphertexts using $\mathcal{F}_{\mathsf{Commit}}$ in Step (2) of $\mathsf{RandShareCom}$.
- Use the simulated $\mathcal{F}_{\mathsf{Commit}}$ to obtain $\boldsymbol{r}_j, \tilde{\boldsymbol{r}}_j$ for $\mathcal{P}_j \in \widehat{\mathcal{P}}$. Send these to $\mathcal{F}_{\mathsf{Offline}}$ and obtain c_i for the $\mathcal{P}_i \in \mathcal{P} \setminus \widehat{\mathcal{P}}$.
- For each $\mathcal{P}_i \in \mathcal{P} \setminus \widehat{\mathcal{P}}$ set $\tilde{\boldsymbol{r}}_i$ such that $c_i^{-1} = pc(\boldsymbol{r}_i, \tilde{\boldsymbol{r}}_i)$ using the trapdoor.
- Afterwards, open the commitments of each honest $\mathcal{P}_i$ using $\mathcal{F}_{\mathsf{Commit}}$ as encryptions $\boldsymbol{r}_i, \tilde{\boldsymbol{r}}_i$. In Step (3) send commitments c_i^{-1} for each honest $\mathcal{P}_i$.
- Follow the protocol. Compute $\boldsymbol{\delta}' \leftarrow \mathsf{Dec}_{\mathsf{sk}}([\![\boldsymbol{m}+\boldsymbol{r}]\!])$. If $\boldsymbol{\delta}' \neq \boldsymbol{\delta}$ then set $\mathsf{cheated} = \top$.

If $\mathcal{P}_1 \in \mathcal{P} \setminus \widehat{\mathcal{P}}$ then do the following:

- Commit to uniformly random ciphertexts using $\mathcal{F}_{\mathsf{Commit}}$ in Step (2) of $\mathsf{RandShareCom}$.
- Use the simulated $\mathcal{F}_{\mathsf{Commit}}$ to obtain $\boldsymbol{r}_j, \tilde{\boldsymbol{r}}_j$ for $\mathcal{P}_j \in \widehat{\mathcal{P}}$. Send these to $\mathcal{F}_{\mathsf{Offline}}$ and obtain c_i for the $\mathcal{P}_i \in \mathcal{P} \setminus \widehat{\mathcal{P}}$.
- For each $\mathcal{P}_i \in \mathcal{P} \setminus \widehat{\mathcal{P}}, i \neq 1$ set $\tilde{\boldsymbol{r}}_i$ such that $c_i^{-1} = pc(\boldsymbol{r}_i, \tilde{\boldsymbol{r}}_i)$ using the trapdoor. For $\mathcal{P}_1$ choose $\boldsymbol{\delta} \xleftarrow{\$} \mathbb{F}^\ell$ and set $\boldsymbol{r}_1 = \boldsymbol{\delta} - \boldsymbol{m} - (\sum_{j \in [n]\setminus\{1\}} \boldsymbol{r}_j)$ where m is a decryption of the input $[\![\boldsymbol{m}]\!]$ to the procedure. Then set $\tilde{\boldsymbol{r}}_1$ such that $c_1^{-1} = pc(\boldsymbol{r}_1 - \boldsymbol{\delta}, \tilde{\boldsymbol{r}}_1)$ using the trapdoor.
- Afterwards, open the commitments of each honest $\mathcal{P}_i$ using $\mathcal{F}_{\mathsf{Commit}}$ as encryptions $\boldsymbol{r}_i, \tilde{\boldsymbol{r}}_i$. In Step (3) send commitments c_i^{-1} for each honest $\mathcal{P}_i, i \neq 1$ and $c_1^{-1} \cdot pc(\boldsymbol{\delta}, 0)^{-1}$.
- Follow the protocol. Compute $\boldsymbol{\delta}' \leftarrow \mathsf{Dec}_{\mathsf{sk}}([\![\boldsymbol{m}+\boldsymbol{r}]\!])$. If $\boldsymbol{\delta}' \neq \boldsymbol{\delta}$ then set $\mathsf{cheated} = \top$.

Triples: For the first ℓ triples do the following:

(1) In Step (1) of $\mathsf{GenTriples}$ use $\mathsf{SRandShareCom}$ where the c_i come from Step (1) of **Triple** in $\mathcal{F}_{\mathsf{Offline}}$. If $\mathsf{SRandShareCom}$ does abort, then abort.

(2) In Step (2) use $[\![\boldsymbol{m}]\!] = [\![\boldsymbol{a} \cdot \boldsymbol{b}]\!]$ and run $\mathsf{SComReshare}$ where we send $\boldsymbol{a}_i, \boldsymbol{b}_i, \boldsymbol{r}_i, \tilde{\boldsymbol{a}}_i, \tilde{\boldsymbol{b}}_i, \tilde{\boldsymbol{r}}_i$ for each $\mathcal{P}_i \in \widehat{\mathcal{P}}$ to $\mathcal{F}_{\mathsf{Offline}}$. If $\mathsf{SComReshare}$ does abort, then abort.

(3) Simulate the instances of $\mathsf{ASpdzRep}$ for using the above inputs using $\mathsf{SReshare}$. Input the $\boldsymbol{\Delta}_\gamma, \gamma(\boldsymbol{z})_i$ for all 6 instances for $\mathcal{P}_i \in \widehat{\mathcal{P}}$ to $\mathcal{F}_{\mathsf{Offline}}$.

For the second ℓ triples by just run the protocol $\mathsf{GenTriples}$ normally for each simulated honest party, except for abort when a check for $\mathsf{correct}$ or R_{CTC} fails. If in $\mathsf{ComReshare}$ the decrypted value $\boldsymbol{\delta}$ differs from the decryption of $[\![\boldsymbol{m}+\boldsymbol{r}]\!]$ then set $\mathsf{cheated} = \top$.

Now simulate $\mathsf{DataCheck}$ as follows:

(1) Run $\mathsf{DataCheck}$ honestly until Step (6).

(2) Abort in Step (6) if $\mathsf{DataCheck}$ aborts or if $\mathsf{cheated} = \top$.

Fig. 18: $\mathcal{S}_{\mathsf{OffNormal}}$: Simulator for the protocol $\mathit{\Pi}_{\mathsf{Offline}}$, honest minority, continued.

Acknowledgements

We want to thank Benjamin Fuller for informing us about a bug in $\mathcal{S}_{\mathsf{OffFull}}$ in a previous version of this work, and the anonymous reviewers for their helpful comments.

References

1. Ben Adida. Helios: Web-based open-audit voting. In *USENIX Security Symposium*, volume 17, pages 335–348, 2008.
2. William Aiello, Yuval Ishai, and Omer Reingold. Priced oblivious transfer: How to sell digital goods. In *EUROCRYPT*, pages 119–135, 2001.
3. Miklós Ajtai, János Komlós, and Endre Szemerédi. An o(n log n) sorting network. In *STOC*, pages 1–9, 1983.
4. Gilad Asharov and Claudio Orlandi. Calling out cheaters: Covert security with public verifiability. In *ASIACRYPT*, pages 681–698, 2012.
5. 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.
6. Mihir Bellare, Juan A Garay, and Tal Rabin. Fast batch verification for modular exponentiation and digital signatures. In *Advances in Cryptology—EUROCRYPT'98*, pages 236–250. Springer, 1998.

7. 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.
8. Elette Boyle, Sanjam Garg, Abhishek Jain, Yael Tauman Kalai, and Amit Sahai. Secure computation against adaptive auxiliary information. In *CRYPTO (1)*, pages 316–334, 2013.
9. 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.
10. Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-lwe and security for key dependent messages. In *Advances in Cryptology–CRYPTO 2011*, pages 505–524. Springer, 2011.
11. Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) lwe. *SIAM Journal on Computing*, 43(2):831–871, 2014.
12. Ernest F Brickell, Daniel M Gordon, Kevin S McCurley, and David B Wilson. Fast exponentiation with precomputation. In *Advances in Cryptology—EUROCRYPT'92*, pages 200–207. Springer, 1993.
13. David Chaum, Peter YA Ryan, and Steve Schneider. *A practical voter-verifiable election scheme*. Springer, 2005.
14. David L Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.
15. Josh D Cohen and Michael J Fischer. A robust and verifiable cryptographically secure election scheme. In *FOCS*, volume 85, pages 372–382, 1985.
16. Ronald Cramer and Ivan Damgård. On the amortized complexity of zero-knowledge protocols. In *Advances in Cryptology-CRYPTO 2009*, pages 177–191. Springer, 2009.
17. Emiliano De Cristofaro and Gene Tsudik. Experimenting with fast private set intersection. In *TRUST*, pages 55–73, 2012.
18. Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. Practical covertly secure mpc for dishonest majority - or: Breaking the spdz limits. In *ESORICS*, pages 1–18, 2013.
19. Ivan Damgård and Sarah Zakarias. Constant-overhead secure computation of boolean circuits using preprocessing. In *Theory of Cryptography*, pages 621–641. Springer, 2013.
20. 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.
21. Sebastiaan Jacobus Antonius de Hoogh. *Design of Large Scale Applications of Secure Multiparty Computation: Secure Linear Programming*. PhD thesis, Technische Universiteit Eindhoven, 2012.
22. Dario Fiore and Rosario Gennaro. Publicly verifiable delegation of large polynomials and matrix computations, with applications. In *ACM Conference on Computer and Communications Security*, pages 501–512, 2012.
23. Eiichiro Fujisaki and Tatsuaki Okamoto. A practical and provably secure scheme for publicly verifiable secret sharing and its applications. In *Advances in Cryptology—EUROCRYPT'98*, pages 32–46. Springer, 1998.
24. Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct nizks without pcps. In *EUROCRYPT*, pages 626–645, 2013.
25. 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.
26. Jens Groth. Evaluating security of voting schemes in the universal composability framework. In *International Conference on Applied Cryptography and Network Security*, pages 46–60. Springer, 2004.
27. Yan Huang, David Evans, and Jonathan Katz. Private set intersection: Are garbled circuits better than custom protocols? In *NDSS*, 2012.
28. Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer - efficiently. In *Advances in Cryptology - CRYPTO 2008, 28th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2008. Proceedings*, pages 572–591, 2008.
29. Alptekin Küpçü and Anna Lysyanskaya. Optimistic fair exchange with multiple arbiters. In *ESORICS*, pages 488–507, 2010.
30. Chae Hoon Lim and Pil Joong Lee. More flexible exponentiation with precomputation. In *Advances in cryptology—CRYPTO'94*, pages 95–107. Springer, 1994.
31. Tal Moran and Moni Naor. Receipt-free universally-verifiable voting with everlasting privacy. In *CRYPTO*, pages 373–392, 2006.
32. Moni Naor, Benny Pinkas, and Reuban Sumner. Privacy preserving auctions and mechanism design. In *Proceedings of the 1st ACM conference on Electronic commerce*, pages 129–139. ACM, 1999.

33. 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.
34. Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Advances in Cryptology — CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140. Springer, Berlin, Germany, 1992.
35. Kazue Sako. An auction protocol which hides bids of losers. In *Public Key Cryptography*, pages 422–432. Springer, 2000.
36. Kazue Sako and Joe Kilian. Receipt-free mix-type voting scheme. In *Advances in Cryptology—EUROCRYPT'95*, pages 393–403. Springer, 1995.
37. Berry Schoenmakers. A simple publicly verifiable secret sharing scheme and its application to electronic voting. In *Advances in Cryptology—CRYPTO'99*, pages 148–164. Springer, 1999.
38. Markus Stadler. Publicly verifiable secret sharing. In *Advances in Cryptology—EUROCRYPT'96*, pages 190–199. Springer, 1996.
39. Tomasz Truderung, Andreas Vogt, and Ralf Küsters. Accountability: definition and relationship to verifiability. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 526–535. ACM, 2010.

A A Generic Solution

We now argue that it is possible to securely implement $\mathcal{F}_{\mathsf{Online}}$ using generic tools, namely a "strong" semi-honest OT protocol in the sense that the protocol should be secure even if the adversary tampers with the corrupted parties internal tapes (but follows the protocol honestly), and universally composable non-interactive zero-knowledge proofs of knowledge(UC-NIZKPoKs) in the CRS model.

First of all, note that UC-NIZKPoKs trivially implement an auditable functionality: if the CRS and the proof are posted on the bulletin board, then the auditor (i.e., anyone) can run the verifier algorithm and *double-check* the output of the verifier.

Several notions of "strong" semi-honest protocols have been used in recent works – see Remark 1 in [28] or the notion of "semi-malicious" in [8]. In all notions different requirements of security still hold when the adversary can tamper with the randomness of otherwise semi-honest parties.

In our setting, we need an OT protocol that is still *secure* even if the adversary tampers with the random tape of one of the parties, and in addition the protocol should still be *correct* even if the adversary tampers with the random tape of *both* parties. Here *security* is defined as the usual notion of indistinguishability of the joint distribution of the view of the corrupted party and the outputs of all parties (including the honest ones) between a real execution of the protocol and a simulated one. The *correctness* requirements can similarly be defined, but we only require that indistinguishability should hold w.r.t. the output of the computation.

Note that in the case where there is at least one honest party, any semi-honest protocol can be turned into one that gives full security (not only correctness) when the adversary tampers with the randomness of the corrupted parties. The transformation goes as follows: at the beginning of the protocol $\mathcal{P}_i$ receives a random string from all other parties and redefines his random tape as the XOR of its original random tape and the strings obtained externally. As long as one party is honest, $\mathcal{P}_i$'s random tape will be uniformly distributed. However it is easy to see that this transformation does not work when all parties are corrupted.

Fortunately many "natural" OT protocols, such as [2], are still correct even when the adversary tampers with the randomness of all parties. Then we can construct an "auditable" GMW-protocol against active adversaries using such an OT protocol and NIZKPoK.

The protocol proceeds as follows: the input parties $\mathcal{P}^I_1, \dots, \mathcal{P}^I_m$ share their inputs using an $n-1$-out-of-n secret sharing scheme and produce commitments to all of their shares. They publish the commitments on the bulletin board and send one share to each server $\mathcal{P}_j$. Those commitments should be binding even if all parties (including the input parties) are corrupted. This can be achieved by using e.g. a commitment scheme where the receiver does not send any message to the sender.

Now the computing parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$ engage in an execution of the GMW-protocol using the strongly-correct OT and prove that all their messages are well formed using the NIZKPoK. If there is at least one honest party, this protocol can be shown to be secure following the GMW-protocol (the only step missing is the "coin-flipping into the well" but this is taken care by the fact that the OT protocol enjoys "strong" security against semi-honest corruptions). In the audit phase $\mathcal{P}^A$ checks all the NIZKs on the bulletin board and accepts y if they do and rejects if any NIZK verification fails. As the OT protocol is guaranteed to be correct even when all parties use bad randomness but follow the protocol, the auditor only outputs ACCEPT y if this is the correct output.

B Efficiency

The practical efficiency of the offline and audit phase of our protocol crucially depends on how fast commitments can be computed and checked. We now discuss the efficiency of our approach and present a few optimizations.

B.1 Asymptotic efficiency

In terms of asymptotic efficiency, Π_{Online} is as efficient as the SPDZ protocol. The number of local field operations and sent values increases by a modest factor of two, plus some additional work for each input-providing party (to check whether the commitment is correct).

To be more precise, we have to distinguish between the field operations in $\mathbb{F}$ and the group operations in $\mathbb{G}$. In the standard setting, where each party provides $O(1)$ inputs and $O(1)$ output values are jointly computed, and where the number of gates in our circuit is upper-bounded by $|\mathcal{C}|$, all operations of the online phase (**Input**, **Add**, **Multiply**, **Output**) together can be performed by each player doing at most $O(n \cdot |\mathcal{C}|)$ field operations. Assuming that we use Pedersen commitments to implement $\mathcal{F}_{\mathsf{Commit}}$ in practice, we obtain an extra $O(n \cdot \log p)$ group operations during **Input** and **Output**. In terms of network load, each party sends or receives $O(n \cdot |\mathcal{C}|)$ field elements over the network during the **Input** phase and while the computation is carried out, and $O(n)$ elements from $\mathbb{F}$ and $\mathbb{G}$ during **Output**.

Concerning the **Audit** phase of the protocol (we exclude $\mathcal{F}_{\mathsf{Offline}}$ from the discussion), we observe that the strategy of it is to *follow the computation with the commitments.* The number of operations in $\mathbb{F}$ is $O(n \cdot |\mathcal{C}|)$, which is comparable to the online phase. In addition, the algorithm performs the gate operations on commitments and checks whether every opening of a commitment was correct – this in total requires $O((n + |\mathcal{C}|) \cdot \log p)$ group operations.

B.2 Towards a faster offline phase

Generating the commitments for $\langle\cdot\rangle_{\mathsf{A}}$-representations can be made faster using preprocessing as in [12,30]. Moreover, it is possible to reduce the total number of commitments (introducing a moderate slowdown during the online phase) as follows: instead of computing *one commitment per value*, one can also use s pairwise distinct generators $g_1, \ldots, g_s \in \mathbb{F}$ together with just one randomness parameter, where generator g_i is used to commit to the ith value.

A representation $(x_1, \ldots, x_t, \tilde{x}, g_1^{x_1} \cdots g_t^{x_t} h^{\tilde{x}})$ of t values in parallel is component-wise linear, and multiplications can also be performed as before (now for multiple elements in parallel). We observe that the computation of a commitment with many generators can be substantially faster than computing all commitments individually. This optimization, similar to [19], works for a large class of circuits. In order to use this optimization, one also has to precompute *permutations between the representations* which must then be used during the online phase. This leads to a moderate slowdown during the evaluation.

B.3 Tweaks for the audit phase

The audit process, as explained in the protocol Π_{Online}, basically consists of (i) performing linear operations on commitments and (ii) checking whether they open to the correct values. Whereas we see no approach to speed up the first part, we will address the second one using a well-known technique from [6].

Let $c_1, \ldots, c_t \in \mathbb{G}$ be the commitments and let $x_1, \ldots, x_t, \tilde{x}_1, \ldots, \tilde{x}_t$ be the values that should open them. We want to establish that $\forall i \in [t] : c_i = g^{x_i} h^{\tilde{x}_i}$. The idea is to compute a random linear combination of all commitments, and thus to check all of them at once. Therefore, choose the coefficients of the random combination from the interval $0, \ldots, 2^\kappa - 1$. Now computing such a random linear combination will yield a false positive with probability $\approx 2^{-\kappa}$, but we can adjust the error probability and make it independent of the field description size $\log |\mathbb{F}|$. This also yields less computational overhead, as we only have to raise group elements to at most 2^κth powers. The algorithm looks as follows:

(1) Sample $\boldsymbol{a} \overset{\$}{\leftarrow} \{0, \ldots, 2^\kappa - 1\}^t$ uniformly at random.
(2) Check that $\prod_i c_i^{\boldsymbol{a}[i]} = \prod_i (g^{x_i} h^{\tilde{x}_i})^{\boldsymbol{a}[i]} = g^{\sum_i \boldsymbol{a}[i] x_i} h^{\sum_i \boldsymbol{a}[i] \tilde{x}_i}$.

Bellare et al. show in [6] that this algorithm fails to correctly verify with probability $2^{-\kappa}$.