TaaS: Commodity MPC via Triples-as-a-Service

Nigel P. Smart[1,2][0000−0003−3567−3304] and Titouan Tanguy[1][0000−0002−7965−620X]

[1] imec-COSIC, KU Leuven, Leuven, Belgium.
[2] University of Bristol, Bristol, UK.
titouan.tanguy@kuleuven.be, nigel.smart@kuleuven.be

Abstract. We propose a mechanism for an m-party dishonest majority Multi-Party Computation (MPC) protocol to obtain the required pre-processing data (called Beaver Triples), from a subset of a set of cloud service providers; providing a form of TaaS (Triples-as-a-Service). The service providers used by the MPC computing parties can be selected dynamically at the point of the MPC computation being run, and the interaction between the MPC parties and the TaaS parties is via a single round of communication, logged on a public ledger. The TaaS is itself instantiated as an MPC protocol which produces the triples for a different access structure. Thus our protocol also acts as a translation mechanism between the secret sharing used by one MPC protocol and the other.

1 Introduction

Secure Multi-Party Computation (MPC) enables a set of parties to securely compute a function on their private inputs, revealing no more than what is revealed by the output of the function. Such computations are however often expensive. In the late 1990's Beaver [3] had already thought about the complexity that stems from general cryptographic computation, and to remedy this problem he defined what he called commodity-based cryptography, related to the notion of beacons introduced by Rabin [19]. The high level idea of commodity-based cryptography is to have commodity servers establishing shared resources for a group of computing parties. The commodity servers do not take part in the actual computation of the cryptographic primitive, but instead provide a resource which reduces the computational costs of the users of this commodity. In Beaver's commodity paradigm, the commodity servers need have no knowledge of each other, only that some parties asked them for resources. These commodity servers would then respond to the parties request with a single message containing correlated randomness independent of the parties inputs. In [3] Beaver proposes a protocol to achieve 2-party OT in the commodity setting, assuming passive corruption of one client and a minority of the commodity servers. Thus in modern parlance the commodity servers act like a set of cloud service providers which provide RaaS (Randomness-as-a-Service) to a set of computing parties.

MPC is not new, it dates back to Yao's famous protocol [23] for solving the millionaire's problem, but was until very recently considered totally inefficient for practical applications. Over the last ten years there has been considerable work in bridging the gap between theory and practice. A notable variant of MPC which is particularly efficient, for many parties, is that of secret sharing based MPC. The rise of secret sharing based MPC can be linked with the *pre-processing model*, which was another introduction of Beaver [2], and which was first efficiently implemented in the VIFF protocol [9]. In the *pre-processing model*, the protocol is split up into two distinct sub-protocols. In what we call an offline phase (or pre-processing), the parties interact with each other to emulate a trusted dealer that securely distributes correlated randomness. The offline phase is independent from the inputs of the parties and as such can be computed at any point prior the evaluation of the circuit. The correlated randomness produced during the pre-processing is then consumed in

the so-called online phase. The online phase actually computes the function the parties agreed on. The advantage of considering MPC in the *pre-processing model*, much alike the RaaS setting, is that we can outsource all the cryptographically intensive computation to the offline phase. This allows us to have a very fast online phase, which usually consists essentially of information theoretic computation.

Early work, such as VIFF [9] considered the case of a semi-honest adversary, with a honest majority. That is an adversary that does not deviate from the protocol and can corrupt only a minority of parties. More recent work are able to defend against a much stronger and arguably more realistic malicious adversary, which can arbitrarily deviate from the protocol, with dishonest majority. The first modern work in secret sharing MPC against a malicious adversary that corrupts a majority of the parties is due to BDOZ [4]. However, since then the SPDZ [11] family of protocols has been able to achieve better results, replacing the pairwise MACs used in BDOZ by introducing a unique secretly shared MAC key amongst all the players. One of the major drawbacks of all those protocols is that the offline phase, which produces correlated randomness, is computationally heavy.

The correlated randomness produced by the offline phase, is (generally) so-called Beaver triples. These are sharings of two random values a and b, along with their shared product $c = a \cdot b$. There are mainly two methods in the literature for generating such triples: homomorphic encryption, as presented in the original SPDZ protocol or Oblivious Transfer (OT) extensions proposed in the MASCOT [16] protocol, which improves on the work of Frederiksen et al [13]. Both these methods require very expensive public key cryptography, making the offline phase of SPDZ-like protocols order of magnitudes slower than the online one. We propose to produce these triples used by the computing parties as a third party Triples-as-a-Service (TaaS).

In [21] a similar methodology for outsourcing triple production was provided for the SPDZ family of protocols. However, in their work the computing parties have to be fixed and constant throughout the execution of the outsourcing protocol, thus the model does not directly map onto the commodity service idea of Beaver, nor is it suited for TaaS for cloud providers to sell. Other work has been done to outsource some parts of MPC protocols, with various goals in mind. Some tackle the problem of outsourcing garbled circuits generation [5], or their evaluation [14,6]. Others limit themselves to semi-honest adversaries [12,20] or work in the specific two party case [7,18]. Eventually there are also works which leverage the functionality of FHE to achieve outsourced MPC [1]. In our work we generalise the method of [21] to be closer to the original commodity model of Beaver. In particular we allow the commodity servers to do most of the computation regardless of which computing parties that will ask for their help. Thus, in our approach of commodity MPC, the set of computing parties can be dynamic. In addition the computing parties can also dynamically select which of the commodity servers (subject to some minor conditions) will be selected to produce the correlated randomness. Thus the resources produced by the commodity servers more closely match the definition of commodities as envisioned by Beaver. As a consequence, in our approach, the commodities can be seen as a service available in the cloud.

Contribution: In this work we improve on the outsourcing model of [21] to better match the communication model envisioned in [3]. We present a method to outsource the pre-processing for the SPDZ family of protocols to a set of service providers. The main difference from [21] is that in our work we require the set of service providers to be able to communicate with each other only up to the commodity request. In addition the commodity servers are assumed to only have an honest majority, or more generally to follow a $\mathcal{Q}_2$ access structure, as opposed to a dishonest majority (a

$\mathcal{Q}_2$ access structure is one in which no union of two unqualified subsets can form the whole set). In addition we will require an immutable, append only public ledger to log the transactions.

After receiving the request for correlated randomness from the group of computing parties via the ledger (one could think of using a smart contract on a public blockchain to achieve this), the service providers will only perform local computation and point-to-point communication with the later group to respond to these requests. The computing parties (the clients in our outsourcing model) can select which subset of commodity servers they wish to obtain data from.

We will denote by **SP** the set of n service providers that run (say) in the cloud, we assume these come defined with an access structure S which we assume is $\mathcal{Q}_2$. The set **SP** is the service provider in our TaaS application. There is in addition a set **CP** of m computing parties who wish to perform a secure computation. From **SP** a set $\mathbf{SP}^r$ of r service providers are selected by the computing parties **CP**. The set $\mathbf{SP}^r$ must be selected so that the access structure S restricted to $\mathbf{SP}^r$ is also $\mathcal{Q}_2$.

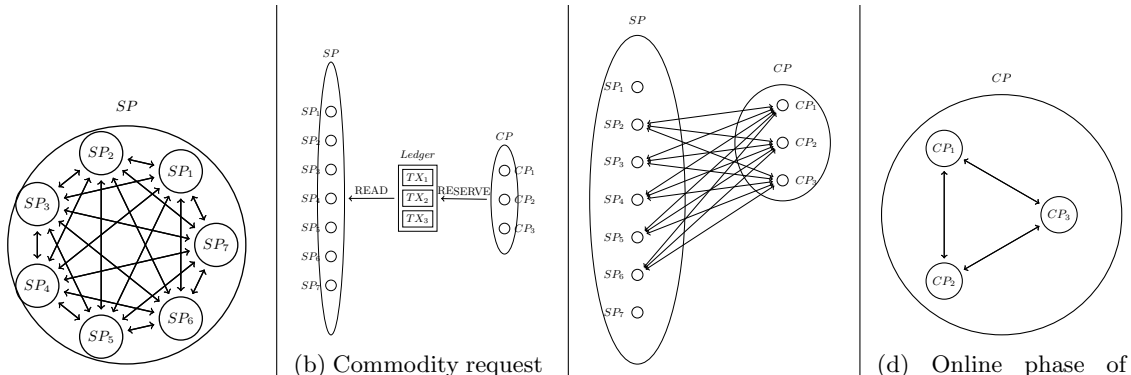
A $\mathcal{Q}_2$ sharing for the set **SP** instead of a full threshold sharing allows for the set of computing parties to interact with only a subset of the service providers while still being able to securely retrieve the required commodities. In addition, this stronger requirement allows the service providers to benefit from a somewhat cheaper MPC protocol. We can also argue that the computing parties could easily agree on some big corporation that provide web services to not be malicious when they act as service providers.

In our setting we let the adversary actively corrupt at most an unqualified set of parties from the service providers and $m-1$ computing parties at the same time. In theory the choice of the access structure for **SP** and **CP** can be either $\mathcal{Q}_2$ or full threshold as long as it implies an MPC protocol. However we note that having a full threshold, SPDZ-like, MPC protocol for **SP** makes the *offline* generation of commodity data much more expensive. Moreover, one will have to take into account that a SPDZ-like sharing involves a share of a MAC alongside every shared values. Meaning that the re-sharing procedure will also be more involved. Regarding the online phase where only the parties in **CP** are involved, we believe that with access to a pool of correlated randomness, the choice of a full threshold MPC scheme is more meaningful. Indeed, the online phase of the SPDZ protocol is almost as efficient as the one induced by a $\mathcal{Q}_2$ access structure, but provides much more confidence when distrusting parties are involved. Therefore, as described above, our focus will be on a $\mathcal{Q}_2$ access structure for **SP** and a full threshold for **CP**. In practice, our setting also gives us the advantage that the computing parties can choose the minimal subset of service providers that respect the $\mathcal{Q}_2$ property and is geographically the closest to optimize the latency of their communication.

With this notation, we now distinguish four different phases for our commodity MPC model as presented in Figure 1:

1. First the service providers **SP** pre-compute Beaver triples.
2. Then the computing parties **CP** send a commodity request via a ledger to **SP**.
3. The service providers answer to this request by sharing authenticated triples to the computing parties; with no interaction between the parties in **SP**.
4. Eventually the computing parties run the computationally cheap online phase of the SPDZ protocol.

In this model, the last three phases respect Beaver's commodity model exposed in [3] and only the first step deviates by requiring the commodity servers to communicate with each other.



(a) Pre-processing between parties in **SP**

(b) Commodity request from **CP** to **SP** via a ledger

(c) Reshares triples from **SP** to **CP**

(d) Online phase of SPDZ between parties in **CP**

Fig. 1: Overview of the four different phases of our commodity MPC

In this paper we will utilize two distinct linear secret-sharing schemes (LSSS). The first one is a multiplicative secret sharing scheme over n parties implementing a $\mathcal{Q}_2$ complete monotone access structure denoted by $\langle\cdot\rangle_{n,S}$, the second one is a full threshold additive sharing over m parties denoted by $[\![\cdot]\!]_m$. The sharing $[\![\cdot]\!]_m$ is authenticated in that each share consists of an additive sharing of the shared value x, as well as an additive sharing of the MAC-value $\alpha \cdot x$, for some global (hidden) MAC key α. Both secret sharing schemes give rise to MPC protocols which are actively secure with abort, the former, leveraging the multiplicative property, uses classic protocols (see [22] for a modern presentation), and the latter using the SPDZ protocol [11]. The $\langle\cdot\rangle_{n,S}$ LSSS will be used for the parties in **SP** and the $[\![\cdot]\!]_m$ LSSS will be used for the parties in **CP**.

Our computing parties **CP**, we assume, want to run a dishonest majority MPC protocol, based (say) on SPDZ. Thus they have an efficient online protocol which can evaluate any function, as long as they have access to a functionality providing the authenticated (i.e. MAC'd) Beaver triples $\mathcal{F}_{\mathsf{Prep},[\![\cdot]\!]_m}^{\mathbf{CP},\mathcal{A}}$; namely the SPDZ offline phase for the LSSS $[\![\cdot]\!]_m$. The computing parties **CP** want to outsource this pre-processing to a set of commodity servers **SP**. Here we assume that the commodity servers **SP** operate in an honest majority setting, with an access structure S which is $\mathcal{Q}_2$. The computing parties select a subset $\mathbf{SP}^r$ of the **SP** that also satisfies the $\mathcal{Q}_2$ property, and reserve a set of authenticated Beaver triples via a transaction on the ledger. Note that the commodity servers have no knowledge of the MAC key held by the computing parties, and that each request to the commodity servers can be from a different set of computing parties.

The commodity servers produce their own offline data, which is essentially Beaver triples with respect to the $\mathcal{Q}_2$ multiplicative secret sharing scheme, via access to an offline functionality $\mathcal{F}_{\mathsf{Prep},\langle\cdot\rangle_{\mathsf{n,S}}}^{\mathbf{SP},\mathcal{A}}$. This is a functionality which produces Beaver triples with respect to the $\langle\cdot\rangle_{n,S}$ LSSS. The precise nature of how $\mathcal{F}_{\mathsf{Prep},\langle\cdot\rangle_{\mathsf{n,S}}}^{\mathbf{SP},\mathcal{A}}$ is implemented will not bother us (but see for example [22]). What is of concern to us, is how the commodity servers respond to the request from the computing parties, namely a protocol $\Pi_{\mathsf{Prep}}^{\mathbf{SP}\to\mathbf{CP},\mathcal{A}}$ which maps the output of $\mathcal{F}_{\mathsf{Prep},\langle\cdot\rangle_{\mathsf{n,S}}}^{\mathbf{SP},\mathcal{A}}$ to the output of $\mathcal{F}_{\mathsf{Prep},[\![\cdot]\!]_{\mathsf{m}}}^{\mathbf{CP},\mathcal{A}}$.

Our work thus bridges the gap between step 1a and step 1d of Figure 1 by proposing a secure protocol for step 1c. Informally, the service providers compute a lot of triples interactively before reading a request on the ledger from a computing group **CP**. With the commodity request, the selected subset in **SP** receives a $\langle\cdot\rangle_{r,S}$ re-sharing of the $[\cdot]_m$-shared MAC key held by the computing

parties. By using standard *re-sharing* techniques, the parties in the subset of **SP** are able to translate their sharing of correlated randomness from their honest majority sharing scheme to the full threshold one of **CP**.

Thanks to the $\langle\cdot\rangle_{r,S}$-sharing of the $[\cdot]_m$-sharing of the MAC key received along with the commodity request and to the cost of one multiplication (and hence one $\langle\cdot\rangle_{r,S}$ Beaver-triple) per field element that is *re-shared* from **SP** to **CP**, parties in **SP** are also able to create the sharing of the MAC-value required by the online phase of the SPDZ [11] protocol.

We note that unlike previous work, our protocol permits the parties in **SP** to create the sharing of the MAC-value of any value with respect to a MAC key held by the parties in **CP** without requiring any interaction after the commodity request. Therefore the novelty of this work is that phases Figure 1b-1c match Beaver's commodity paradigm, allowing the exchange of commodities in only two communication rounds between **SP**, the *Ledger* and **CP**, and no interaction between the **SP** after the offline step. Unlike Beaver's model, the parties in **SP** do need to communicate in the first phase, and we also require a *Ledger*, for which a blockchain seems to be a good candidate.

We end this introduction with a quick overview of what follows: In Section 2 we briefly introduce the notion of Linear Secret Sharing Schemes and describe the offline phase of SPDZ that we want to emulate via our commodity protocol. We then introduce our notations. In Section 3 we explain the main steps of our protocol and state our main theorem. We continue by formally defining $\Pi^{R\to Q,\mathcal{A}}_{\mathsf{Prep}}$ and give a simulator that proves our theorem in the UC framework. We finish by Section 4 in which we give some conclusions and further thoughts.

2 Preliminaries

In this section we first briefly introduce the concept of multiplicative Linear Secret Sharing Schemes (LSSS), and the MPC protocols associated with them. In particular we discuss protocols in the SPDZ family, i.e. ones based on an offline phase which produces Beaver triples, in more detail. We finish the section by introducing an ideal functionality for a ledger, which we use to keep a log on every transactions that happened.

2.1 Linear Secret Sharing Schemes

Recall, we use $\langle\cdot\rangle_{n,S}$ to denote a mulitplicative LSSS with a $\mathcal{Q}_2$ access structure S over n parties, and $[\![\cdot]\!]_m$ to denote a full threshold mutliplicative LSSS over m parties. The $\mathcal{Q}_2$ sharing $\langle\cdot\rangle_{n,S}$ will be used as the basis of the sharing for the commodity servers **SP**, whilst the full threshold sharing $[\![\cdot]\!]_m$ will be used as the sharing for the computing parties **CP**. We let $\mathbf{SP} = \{\mathsf{SP}_1, \ldots, \mathsf{SP}_n\}$ and $\mathbf{CP} = \{\mathsf{CP}_1, \ldots, \mathsf{CP}_m\}$.

The $\langle\cdot\rangle_{n,S}$-sharing: To define the $\langle\cdot\rangle_{n,S}$ we must first introduce the notion of a complete monotone access structure, and what it means to be $\mathcal{Q}_2$.

Definition 2.1. *Let $\mathcal{P} = \{P_1, \ldots, P_n\}$ be a set of parties, $\Gamma \in 2^{\mathcal{P}}$ the monotonically increasing set of qualified set and $\Delta \in 2^{\mathcal{P}}$ the monotonically decreasing set of unqualified set. If $\Gamma \cap \Delta = \emptyset$ then (Γ, Δ) is a monotone access structure. If in addition it holds that $\Delta = 2^{\mathcal{P}} \setminus \Gamma$, then the access structure is said to be complete.*

In the following we will only consider complete monotone access structure, and will refer to those simply as access structure. We remark that such an access structure is completely defined by $\mathcal{P}$ and either one of the qualified set or unqualified set. Therefore we will often talk about Γ access structure or Δ access structure to refer to the (Γ, Δ) access structure. For an access structure to be $\mathcal{Q}_2$ we now require an additional property.

Definition 2.2. *An access structure (Γ, Δ) is $\mathcal{Q}_2$ if for all $U_1, U_2 \in \Delta$ we have $U_1 \cup U_2 \subsetneq \mathcal{P}$*

Definition 2.3. *A $\mathcal{Q}_2$ (share-reconstructable) sharing of $a \in \mathbb{F}_p$, with $\mathbb{F}_p$ a field of size p for p a prime power, and S a $\mathcal{Q}_2$ access structure on n parties, is defined as $\langle a \rangle_{n,S}$ such that any qualified set of parties Q can reconstruct the secret a from a linear combination of their shares.*

In the rest of the paper when talking about LSSS we implicitly mean multiplicative LSSS, define as in 2.4.

Definition 2.4. *A LSSS is said to be multiplicative if given two sharings of two secrets s and s', the product $s \cdot s'$ is a linear sum of the Schur product (the local tensor product) of the shares of the secrets.*

To abstract away from any specific scheme, it is easier to think about $\mathcal{Q}2$ LSSS through Monotone Span Programs (MSPs). This model of computation was first described by Karchmer and Wigderson [15], and has been shown to induce LSSSs. We will only informally describe how to use a MSP to derive a LSSS, and refer the reader to [22] for more details. Informally a MSP is defined by $(\mathbb{F}_p, M, \mathbf{k}, \iota)$ where $\mathbb{F}_p$ is a field, $M \in \mathbb{F}_p^{l \times k}$ a full-rank matrix, $\mathbf{k} \in \mathbb{F}_p^k$ a non-zero vector and ι is a labeling function of rows of M to parties in $\mathcal{P}$. Now to share a secret s, one needs to sample $\mathbf{v} \leftarrow \mathbb{F}_p^k$ subject to $\langle \mathbf{v}, \mathbf{k} \rangle = s$. Then define the vector $\mathbf{s} = M \cdot \mathbf{v}$ and for all $i \in \{1, \ldots, l\}$ let party $\iota(i)$ have $\mathbf{s}[i]$. We note that the definition of a MSP implies that for any qualified set, i.e. any subset $Q \in \Gamma$, there exists a recombination vector $\boldsymbol{\lambda}_Q$ such that $\langle \boldsymbol{\lambda}_Q, \mathbf{s}_Q \rangle = s$, where $\mathbf{s}_Q$ denotes the entries of the vector $\mathbf{s}$ held by parties in Q.

In the following we will denote by $\langle a \rangle_{n,S} = (a_1, \ldots, a_l)$ a $\mathcal{Q}_2$ sharing of a. We define $a_i = \langle a \rangle_{n,S}^i$. We also associate such sharing with the labeling function $\iota : \{1, \ldots, l\} \to \mathcal{P}$ such that the share a_i of a is held by the party $\iota(i)$. We note that ι need not to be bijective, because it is not always injective. Therefore, when we refer to $\iota^{-1}(\mathsf{SP}_i)$ for a party $\mathsf{SP}_i \in \mathbf{SP}$, we refer to the rows of the matrix M *owned* by this party.

An example of such scheme, that we use as an example throughout this paper, is the Shamir sharing, for which the MPC protocol of [22] makes use of the fact that if one receives $t+1$ shares then not only can one reconstruct the secret a, but one can also detect (with high probability if p is large) if an adversarial set has given one invalid share. This fact is used to produce an actively secure with abort MPC protocol for honest majority. Thus, in some sense, the Shamir LSSS is self authenticating. The MSP representation of the (t, n) Shamir sharing is $(\mathbb{F}_p, M, \mathbf{k}, \iota)$ where ι is the function that maps row i to player P_i, $\mathbf{k} = (1, 0, \ldots, 0) \in \mathbb{F}_p^{t+1}$ and

$$M = \begin{pmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & 2 & 4 & \ldots & 2^t \\ 1 & 3 & 9 & \ldots & 3^t \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & n & n^2 & \ldots & n^t \end{pmatrix}$$

The $[\![\cdot]\!]_m$-sharing: To define the $[\![\cdot]\!]_m$ notation we first introduce a basic full threshold sharing, with no authentication.

Definition 2.5. *A full threshold m-party additive secret sharing of $a \in \mathbb{F}_p$, with $\mathbb{F}_p$ a finite field of size p a prime power, is defined as $[a]_m = (a_1, \ldots, a_m)$, with $\forall i \in \{1, \ldots, m\}$, $a_i \in \mathbb{F}_p$, such that $a = \sum_{i=1}^{m} a_i$, with $\mathsf{CP}_i \in \boldsymbol{CP}$ only knowing a_i.*

Unlike the Shamir scheme the LSSS given by $[\cdot]_m$ is not self authenticating. Thus in the SPDZ papers [11,10] the authors introduce an authenticated full threshold LSSS.

Definition 2.6. *An authenticated full threshold m-party additive secret sharing of $a \in \mathbb{F}_p$, with $\mathbb{F}_p$ a finite field of size p a prime power, is defined as $[\![a]\!]_m = ([a]_m, [\alpha \cdot a]_m)$, where $\alpha \in \mathbb{F}_p$ is a global secret value which is $[\cdot]_m$-shared.*

The authentication of a $[\![a]\!]_m$-sharing $([a]_m, [\alpha \cdot a]_m)$ given an opening of the $[a]_m$ value is not immediate, unlike in the case of Shamir sharings. However, a protocol (called $\mathsf{MACCheck}$ and given in [10]) allows one to verify such openings, without needing to open either $[\alpha \cdot a]_m$ or $[\alpha]_m$.

2.2 Pseudo-Random Secret Sharing

We will make use of the functionality $\mathcal{F}^{\mathcal{P},\mathcal{A}}_{\mathsf{PRSS},\langle\cdot\rangle_{\mathsf{n,S}}}$ given in Figure 2, which we take from [22]. In the case of Shamir sharing (for smallish values of t and n) one can obtain this functionality efficiently without interaction, using well known techniques dating back to [8]. For other access structures, or larger values of t and n, one can easily obtain this functionality using interaction.

Ideal Functionality $\mathcal{F}^{\mathcal{P},\mathcal{A}}_{\mathsf{PRSS},\langle\cdot\rangle_{\mathsf{n,S}}}$

On input $PRSS(count)$ from all parties, if the counter value is the same for all parties and has not been used before, the functionality samples $a \leftarrow \mathbb{F}_p$, computes share vector $\langle a\rangle_{n,S}$ and sends $\langle a\rangle^k_{n,S}$ to $P_{\iota(k)}$.

Figure 2: Ideal Functionality $\mathcal{F}_{\mathsf{PRSS},\langle\cdot\rangle_{\mathsf{n,S}}}$

2.3 SPDZ-Like MPC Protocols

We assume that we aim to produce MPC protocols for arithmetic circuits over $\mathbb{F}_p$, for a prime p. To simplify our presentation we require p to be large enough such that $1/p$ is a negligible function of the security parameter λ. Improvements in the protocols we use can alleviate this constraint, but we keep to the case of large p for ease of exposition.

Our goal is to produce an MPC protocol for a set of parties $\mathcal{P}$, where an adversary controls a subset $\mathcal{A}$. Such a protocol is given by the ideal functionality $\mathcal{F}_{\mathsf{MPC}}$ presented in Figure 3. We will be examining MPC over different access structures. In particular the m computing parties wish to execute an MPC protocol for the full threshold case, i.e. the full set of parties are given by $\mathcal{P} = \mathbf{CP}$, and the adversary controls a subset $\mathcal{A}$ such that $\mathcal{A} \cap \mathcal{P} \neq \mathcal{P}$. In the case of the n commodity servers they are essentially executing an MPC protocol (albeit only an offline phase) for a set of parties $\mathcal{P} = \mathbf{SP}$, and the adversary controls an unqualified subset $\mathcal{A} \in \Delta$.

In [11,10] it is shown how to realise the functionality $\mathcal{F}_{\mathsf{MPC}}$ in the $\mathcal{F}^{\mathcal{P},\mathcal{A}}_{\mathsf{Prep},[\![\cdot]\!]_{\mathsf{m}}}$-hybrid model in the case where $\mathcal{A} \cap \mathcal{P} \neq \mathcal{P}$, using the secret sharing scheme $[\![\cdot]\!]_m$. See Figure 4 for the functionality

Ideal Functionality $\mathcal{F}_{\mathsf{MPC}}^{\mathcal{P},\mathcal{A}}$

Initialise: On input $(Initialise, \mathbb{F}_p)$ from all parties in $\mathcal{P}$, store $\mathbb{F}_p$.

Input: On input $(Input, i, id, x)$ from party i and $(Input, i, id)$ from all other parties, with id a fresh identifier and $x \in \mathbb{F}_p$, store (id, x).

Add: On command (Add, id_1, id_2, id_3) from all parties in $\mathcal{P}$ (where id_1 and id_2 are present in memory), retrieve (id_1, x) and (id_2, y) and store $(id_3, x + y)$.

Multiply: On command $(Multiply, id_1, id_2, id_3)$ from all parties in $\mathcal{P}$ (where id_1 and id_2 are present in memory), retrieve (id_1, x) and (id_2, y) and store $(id_3, x \cdot y)$.

Output: On input $(Output, id)$ from all honest parties (where id is present in memory), retrieve (id, z), output z to the adversary. If the adversary responds OK then output the value z to all parties, otherwise output ABORT to all parties.

Figure 3: Ideal MPC Functionality $\mathcal{F}_{\mathsf{MPC}}$

Ideal Functionality $\mathcal{F}_{\mathsf{Prep},[\![\cdot]\!]_{\mathsf{m}}}^{\mathcal{P},\mathcal{A}}$

Initialise: On input ($Initialise$, p, T, R_1, ..., R_m) from all players and the adversary:

1. The functionality samples $\alpha \leftarrow\!\!\$\, \mathbb{F}_p$ to be the global MAC key.
2. The functionality receives for each corrupted player $i \in \mathcal{A}$ a share α_i .
3. The functionality samples at random α_i for each $i \notin \mathcal{A}$ s.t. $\sum_{i \in \mathcal{P}} \alpha_i = \alpha$.
4. The functionality sends α_i to party i, for all $i \in \mathcal{P}$.

Macro: $\mathsf{Angle}(x)$:

1. The functionality accepts $(\{x_i, \gamma(x)_i\}_{i \in \mathcal{A}})$ from the adversary.
2. The functionality waits for the adversary to send a signal either ABORT or $\mathsf{PROCEED}$. If ABORT, it aborts, otherwise it continues.
3. The functionality samples at random $\{x_i, \gamma(x)_i\}_{i \notin \mathcal{A}}$ s.t. $\sum_{i \in \mathcal{P}} x_i = x$ and $\sum_{i \in \mathcal{P}} \gamma(x)_i = \alpha \cdot x$.

Preprocessing:

- For i in $\mathcal{P}$
  1. If $i \notin \mathcal{A}$ the functionality samples $r^{(k)} \leftarrow\!\!\$\, \mathbb{F}_p$ for $k \in R_i$. Otherwise $i \in \mathcal{A}$ so the functionality accepts $r^{(k)}$ for $k \in R_i$ from the adversary.
- For all $i \in \mathcal{P}$, for $k \in R_i$:
  1. The functionality runs $\mathsf{Angle}(r^{(k)})$.
  2. For each $j \in \mathcal{P}$, the functionality sends party j the pair $(r_j^{(k)}, \gamma(r^{(k)})_j)$.
  3. Additionaly, to party i, the functionality sends $r^{(k)}$.
- For $k \in T$
  1. The functionality samples $a^k, b^k \leftarrow\!\!\$\, \mathbb{F}_p$ and computes $c^k = a^k \cdot b^k$.
  2. The functionality calls $\mathsf{Angle}(a^k), \mathsf{Angle}(b^k), \mathsf{Angle}(c^k)$.
  3. The functionality outputs $((a_i^k, \gamma(a^k)_i), (b_i^k, \gamma(b^k)_i), (c_i^k, \gamma(c^k)_i))$ to party i.

Figure 4: The Ideal Functionality $\mathcal{F}_{\mathsf{Prep},[\![\cdot]\!]_{\mathsf{m}}}^{\mathcal{P},\mathcal{A}}$ for the LSSS $[\![\cdot]\!]_m$

$\mathcal{F}^{\mathcal{P},\mathcal{A}}_{\mathsf{Prep},[\![\cdot]\!]_m}$ in this case. The functionality is a little different than the one presented in [21] as we do not allow the adversary to introduce an error in the *Angle* macro. Instead, as in [16], the functionality generates the triples *optimistically* and allows the adversary to abort the protocol before sending out the shares. We also require the parties to send in a single message the set of handles T' on the triple items and the sets of handles $R_1, \ldots, R_m$ on the input masks that each of the m parties will need. During the *Initialise* part of the pre-processing phase, the parties agree on a global MAC key α such that each party $i \in \mathcal{P}$ only knows a share α_i of it, subject to $\sum_{i\in\mathcal{P}} \alpha_i = \alpha$. Then, each value x additively shared by the parties in the *Preprocessing* part of the offline phase is accompanied by an additively shared information theoretic MAC $\gamma(x) = \alpha \cdot x$. Therefore, at the end of the offline phase, for all x produced in the *Preprocessing*, party $i \in \mathcal{P}$ holds a tuple $(x_i, \gamma(x)_i)$ subject to $x = \sum_{i\in\mathcal{P}} x_i$ and $\alpha \cdot x = \sum_{i\in\mathcal{P}} \gamma(x)_i$. That is, for all x, a $[\![x]\!]_m$ sharing is produced. We note that such a sharing trivially allows for linear operation without interaction, and hence additions are *free* and the Beaver multiplication [2] trick can be used to multiply two secret-shared values by consuming a pre-processed triple. The strength of the SPDZ protocol resides in the fact that at the end of the online phase, the parties can use the MAC scheme to verify that the adversary did not deviate from the protocol, and abort the protocol in case of malicious activities.

Ideal functionality $\mathcal{F}^{\mathcal{P},\mathcal{A}}_{\mathsf{Prep},\langle\cdot\rangle_{n,S}}$

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

1. (a) Sample $a, b \leftarrow \mathbb{F}_p$ and compute share vectors $\langle a\rangle_{n,S}$ and $\langle b\rangle_{n,S}$.
   (b) Send $(\langle a\rangle^{\iota^{-1}(\mathcal{A})}_{n,S}, \langle b\rangle^{\iota^{-1}(\mathcal{A})}_{n,S})$ to the adversary.
   (c) Receive a subvector of shares $\langle \widetilde{c}\rangle^{\iota^{-1}(\mathcal{A})}_{n,S}$ from the adversary.
   (d) Compute a sharing $\langle c\rangle_{n,S} = \langle a \cdot b\rangle_{n,S}$ such that $\langle c\rangle^{\iota^{-1}(\mathcal{A})}_{n,S} = \langle \widetilde{c}\rangle^{\iota^{-1}(\mathcal{A})}_{n,S}$. If no such vector $\mathbf{c}$ exists, set an internal flag ABORT to true and continue.
2. Wait for a message OK or ABORT from the adversary.
3. If the response is OK and the internal flag ABORT has not been set to true, for each honest $P_i \in \mathcal{P}$, send $(\langle a\rangle^{\iota^{-1}(i)}_{n,S}, \langle b\rangle^{\iota^{-1}(i)}_{n,S}, \langle c\rangle^{\iota-1(i)}_{n,S})$, otherwise output the message ABORT to all honest parties and abort.

Figure 5: The Ideal Functionality $\mathcal{F}^{\mathcal{P},\mathcal{A}}_{\mathsf{Prep},\langle\cdot\rangle_{n,S}}$ for $\mathcal{Q}_2$ sharing $\langle\cdot\rangle_{n,S}$

In Figure 5, taken from [22], we present the equivalent offline phase for the multiplicative $\mathcal{Q}_2$ secret sharing system. We use the notation $\mathcal{F}^{\mathcal{P},\mathcal{A}}_{\mathsf{Prep},\langle\cdot\rangle_{n,S}}$ to denote this offline functionality. It is shown in [22] how to realise the functionality $\mathcal{F}_{\mathsf{MPC}}$ in the $\mathcal{F}^{\mathcal{P},\mathcal{A}}_{\mathsf{Prep},\langle\cdot\rangle_{n,S}}$-hybrid model in the case where $\mathcal{A} \in \Delta$, but this time using the secret sharing scheme $\langle\cdot\rangle_{n,S}$.

2.4 The Ledger

To make sure the service providers do not reshare twice the same commodities, every transaction has to be logged on a ledger. To do this we introduce the ideal functionality $\mathcal{F}_{\mathsf{Ledger}}$ in Figure 6. The ledger stores which commodities are requested for each transactions, making sure that each commodity is reserved only once. The use of a ledger allows us to have an agreement in **SP** on which commodities have already been used without proceeding to a Byzantine Agreement after each requests. The latter solution would break the communication model we try to achieve. An implementation of such a ledger could be done via a smart contract executed over a public blockchain.

In addition the ledger also stores the encrypted re-sharing of the MAC key destined for the service providers in $\mathbf{SP}^r$.

Ideal Functionality $\mathcal{F}_{\mathsf{Ledger}}$

The functionality emulates a publicly readable, immutable, append only ledger.
The functionality starts with two empty sets $T = R = \emptyset$.

Reserve. Upon receiving (RESERVE, T', R_1, ..., R_m, $\{Enc_{sk_{\iota(j)}}(\langle \alpha_i \rangle^j_{r,S})\}_{j \in \iota^{-1}(\mathbf{SP}^r)}$, $\mathbf{SP}^r$, $\mathbf{CP}$) from all $\mathsf{CP}_i \in \mathbf{CP}$,

1. If $T \cap T' \neq \emptyset$, or $\forall i, j \in \mathbf{CP}, i \neq j, R'_i \cap R'_j \neq \emptyset$ or $\forall i \in \mathbf{CP}, R \cap R_i \neq \emptyset$ send FAIL to every parties in $\mathbf{CP}$ that have already sent a message.
2. Otherwise $T = T \cup T'$ and $R = R \cup_{i=1}^{m} R_i$. Plus the functionality stores the request on the ledger.

Read. Upon receiving $(\mathsf{READ}, \mathsf{SP}_i)$ from $\mathsf{SP}_i \in \mathbf{SP}$, reply with all entries of ledger in which SP_i is involved.

Figure 6: Ideal Ledger Functionality $\mathcal{F}_{\mathsf{Ledger}}$

3 Commodity Offline Data

Our goal is to provide a protocol which allows the computing parties to obtain the Beaver triples needed for secure computation in an outsourced manner from a set of commodity servers. To make things more concrete we utilizes the two sets of parties defined above, the service providers **SP** and the computing parties **CP**. In practice, the service providers could be computationally powerful machines hosted in the cloud, producing Beaver triples as a service (TaaS) for low energy devices. We consider an active adversary $\mathcal{A}$ which can corrupt an unqualified set $U \in \Delta$ of the n parties in **SP AND** $m-1$ parties in **CP**.

The idea for our commodity MPC is to have parties in **SP**, which can run in the cloud, to execute the costly offline phase, generating pre-processing data in the $\langle . \rangle_{n,S}$-scheme. For this the parties in **SP** utilize the functionality $\mathcal{F}^{\mathbf{SP},\mathcal{A}\cap\mathbf{SP}}_{\mathsf{Prep},\langle\cdot\rangle_{\mathsf{n,S}}}$. The service providers then execute our re-sharing protocol below to transform the data from a $\langle . \rangle_{n,S}$-sharing to a $[.]_m$-sharing, and add the correct MAC to it to make it a $[\![.]\!]_m$-sharing. The computing parties will then be able to use the data in an online phase of a SPDZ-like protocol. Thus the output of the protocol will be equivalent to the computing parties executing the functionality $\mathcal{F}^{\mathbf{CP},\mathcal{A}\cap\mathbf{CP}}_{\mathsf{Prep},[\![\cdot]\!]_{\mathsf{m}}}$. The offline data is therefore seen as commodities, which shall be computed independently of the parties that are going to use it. To be sure that the same data is not used more than once by **SP**, we use the public ledger to log every request made from the sets **CP**s to **SP**. In practice this ledger could be implemented via a blockchain.

Before proceeding with our protocol we present some observations:

- The computing parties are independent of the commodity parties (unlike the case considered in [21]), because the pre-processing of triples can be done without knowing the MAC key held by **CP**.
- On each application of the protocol with the commodity servers **SP**, there could be a new distinct set of computing parties **CP**, and hence a distinct MAC key α. Indeed even the same set of computing parties can utilize a different MAC key on each execution.

- Contrary to Beaver's idea of commodity cryptography in our model the commodity servers **SP** do interact with each other. However, as opposed to the outsourcing protocol of [21], the interaction amongst the commodity servers can all be done before the request of the computing parties. Thus the interaction between the parties in **SP** can itself be done in an offline manner.

Our protocol, which we call $\Pi_{\mathsf{Prep}}^{\mathbf{SP}\to\mathbf{CP},\mathcal{A}}$, will allow a set of computing parties **CP** to select a subgroup $\mathbf{SP}^r$ of the set **SP** which is minimal to satisfy the $\mathcal{Q}_2$ property and the geographically closest to them to optimize latency. The computing parties will then *only* interact with the commodity servers in $\mathbf{SP}^r$ so as to realise the functionality $\mathcal{F}_{\mathsf{Prep},[\![\cdot]\!]_\mathsf{m}}^{\mathbf{CP},\mathbf{CP}\cap\mathcal{A}}$ in the $\mathcal{F}_{\mathsf{Prep},\langle\cdot\rangle_{\mathsf{n,S}}}^{\mathbf{SP},\mathbf{SP}\cap\mathcal{A}}, \mathcal{F}_{\mathsf{PRSS},\langle\cdot\rangle_{\mathsf{n,S}}}^{\mathbf{CP},\mathbf{CP}\cap\mathcal{A}}$-hybrid model.

3.1 Overview of the protocol

In this section we first briefly describe the main steps of our protocol. We then formally define $\Pi_{Prep}^{SP\to\mathbf{CP},\mathcal{A}}$ and we prove its security, defined by theorem 3.1.

Theorem 3.1. *The protocol $\Pi_{Prep}^{\boldsymbol{SP}\to\boldsymbol{CP},\mathcal{A}}$ securely realizes the ideal functionality $\mathcal{F}_{\mathsf{Prep},[\![\cdot]\!]_\mathsf{m}}^{\boldsymbol{CP},\boldsymbol{CP}\cap\mathcal{A}}$ in the $(\mathcal{F}_{\mathsf{Prep},\langle\cdot\rangle_{\mathsf{n,S}}}^{\boldsymbol{SP},\boldsymbol{SP}\cap\mathcal{A}}, \mathcal{F}_{\mathsf{PRSS},\langle\cdot\rangle_{\mathsf{n,S}}}^{\boldsymbol{SP},\boldsymbol{SP}\cap\mathcal{A}}, \mathcal{F}_{\mathsf{Ledger}})$-hybrid model in the presence of a static, active adversary $\mathcal{A}$ that can corrupt an unqualified set of parties U in **SP** and $m-1$ parties in **CP**.*

Fig 7 outlines the main steps of our Triple as a Service protocol. Initially we only describe the interactions taking place between **SP** and a single set of **CP**. Therefore in this overview we do not describe the mechanism that prevents a set of commodities to be used more than once, and we only introduce it later, through calls to the $\mathcal{F}_{\mathsf{Ledger}}$ in the formal description of our protocol.

First, independent from the potential sets of computing parties, the service providers continuously generate Beaver triples in the $\langle.\rangle_{n,S}$ scheme. For each element of each triple, the service providers also need to pre-compute an additional triple associated to it, that we denote for e.g. $\langle a\rangle_{n,S}$ by $(\langle a_a\rangle_{n,S}, \langle b_a\rangle_{n,S}, \langle c_a\rangle_{n,S})$.

Then when a group of computing parties wants to begin a computation, they generate an additively shared MAC key $[\alpha]_m$, and agree on a restricted set of service providers $\mathbf{SP}^r \subset \mathbf{SP}$ they will interact with. With the commodity request sent from **CP** to $\mathbf{SP}^r$ (through the ledger), each $\mathsf{CP}_i \in \mathbf{CP}$ locally executes **TransformMac** on its share α_i of the MAC key, resulting in $\langle\alpha_i\rangle_{r,S}$. CP_i then sends the shares of its share of α to parties in $\mathbf{SP}^r$ according to the labeling function ι. Using only linear operations, the parties in $\mathbf{SP}^r$ can now compute a sharing $\langle\alpha\rangle_{r,S}$ of α. In the following we demonstrate how the service providers now have the ability to authenticate and reshare their precomputed triples non-interactively, to the cost of one additional triple per resharing.

To do so on e.g. a, all parties in $\mathbf{SP}^r$ locally execute **TransformShare** on their $\langle a\rangle_{n,S}$ sharing of a and also on their $\langle c_a\rangle_{n,S}$ sharing of c_a. They then send the resulting $[a]_m$ and $[c_a]_m$ appropriately to parties in **CP**. The service providers also broadcast their shares of $\langle a-a_a\rangle_{n,S}$ and $\langle\alpha-b_a\rangle_{r,S}$ to all the computing parties. The broadcast allows all $\mathsf{CP}_i \in \mathbf{CP}$ to securely reconstruct the values $a-a_a$ and $\alpha-b_a$ by locally executing **CheckOpening**. Using the classic Beaver multiplication trick, it is now possible for the parties in **CP** to locally compute a share of the MAC value of a. At the end of this protocol, the computing parties hold $[\![a]\!]_m$, an authenticated sharing of a under their global MAC key α.

We now present the different subprotocols used above in our outline of the commodity protocol, and apply them to our (t,r) Shamir sharing as an example.

First, Fig 8 shows how the parties in **CP** can securely transmit a sharing of their global MAC key α to parties in $\mathbf{SP}^r$. We note that in the detailed protocol described later, this subroutine is slightly modified to take into account the call to the $\mathcal{F}_{\mathsf{Ledger}}$ functionality.

In our example of a (t, r) Shamir sharing, Fig 8 works as follow:
We note that for all $\mathsf{CP}_i \in \mathbf{CP}$ we have $P_{\alpha_i}(0) = \alpha_i$, therefore $\sum_{i=1}^m P_{\alpha_i}(0) = \sum_{i=1}^m \alpha_i = \alpha$. Plus $\sum_{i=1}^m P_{\alpha_i}$ is a sum of m degree t polynomials, therefore a degree t polynomial. By definition $(\sum_{i=1}^m P_{\alpha_i}(1), \ldots, \sum_{i=1}^m P_{\alpha_i}(r))$ is then a (t, r)-Shamir sharing of α.

For the second step, in Fig 9 we describe the re-sharing from $\mathbf{SP}^r$ to **CP**. For our Shamir sharing example Fig 9 transforms a $\langle\cdot\rangle_{t,r}$ to a $[\![\cdot]\!]_m$ sharing. Given $\mathbf{SP}^r_{t+1} \subset \mathbf{SP}^r$ of size $t+1$, we have:

$$\begin{aligned}\sum_{k=1}^{m} \sum_{j \in \mathbf{SP}^r_{t+1}} a_{j,k} &= \sum_{j \in \mathbf{SP}^r_{t+1}} \sum_{k=1}^{m} a_{j,k} \\ &= \sum_{j \in \mathbf{SP}^r_{t+1}} P_a(j) \cdot \Delta_j(0) \text{ with } \Delta_j \text{ the Lagrange coefs.} \\ &= a\end{aligned}$$

Therefore by definition $(\sum_{j \in \mathbf{SP}^r_{t+1}} a_{j,1}, \ldots, \sum_{j \in \mathbf{SP}^r_{t+1}} a_{j,m})$ is a $[.]_m$ sharing of a.

Finally, we have the opening subprotocol defined in Fig 10. This protocol is executed by the parties in **CP** to check correctness of the broadcast shares, and retrieve the secret value from them. Since we have a qualified set of honest parties in $\mathbf{SP}^r$, we know that among the r shares that are broadcast, enough shares come from honest service providers. Therefore if the subroutine does not output $\perp$, all the service providers must have broadcast their correct share of a. We observe that in our example with a Shamir sharing, we can do this efficiently using a parity check matrix.

Informally we can see that our commodity protocol is secure because the adversary can only tamper with the shares of a and c_a. Indeed, the **checkOpening** subroutine prevents the adversary to tamper with $a - a_a$ and $\alpha - b_a$. If the adversary decides to cheat by introducing errors $[a]_m \to [a + \epsilon_1]_m$ **and** $[c_a]_m \to [c_a + \epsilon_2]_m$, the parties end up with the result of Eq 1 instead of αa.

$$\begin{aligned}&-(a - a_a)(\alpha - b_a) + (c_a + \epsilon_2) \\ &+(\alpha - b_a)(a + \epsilon_1) + (a - a_a)\alpha \\ =\ &-a\alpha + ab_a + a_a\alpha - a_a b_a + (c_a + \epsilon_2) \\ &+\alpha(a + \epsilon_1) - b_a(a + \epsilon_1) + a\alpha - a_a\alpha \\ &= (a + \epsilon_1)\alpha - b_a\epsilon_1 + \epsilon_2 \qquad (1)\end{aligned}$$

To pass the MAC check, the adversary needs $\epsilon_2 = b_a\epsilon_1$, which with b_a unkown, results in guessing with probability $\lambda = \frac{1}{p}$ for a finite field of size p.

3.2 Definition and security of $\Pi^{\mathbf{SP}\to\mathbf{CP},\mathcal{A}}_{Prep}$

In this section we prove that $\Pi^{\mathbf{SP}\to\mathbf{CP},\mathcal{A}}_{Prep}$ described in Fig 11 implements $\mathcal{F}^{\mathbf{CP},\mathbf{CP}\cap\mathcal{A}}_{\mathsf{Prep},[\![\cdot]\!]_m}$ described in Fig 4 securely in the $(\mathcal{F}^{\mathbf{SP},\mathbf{SP}\cap\mathcal{A}}_{\mathsf{Prep},\langle\cdot\rangle_{n,S}}, \mathcal{F}^{\mathbf{SP},\mathbf{SP}\cap\mathcal{A}}_{\mathsf{PRSS},\langle\cdot\rangle_{n,S}}, \mathcal{F}_{\mathsf{Ledger}})$-hybrid model. The first two ideal functionalities are defined in [22] and all three are described respectively in Fig 5, 2 and 6 above. To prove theorem 3.1, we now analyze why the simulator described in Fig 12-13 gives the correct view to the environment.

Offline : In the offline phase, the simulator behavior is straightforward. The simulator only emulates the hybrid functionalities for the offline phase of the $\langle\cdot\rangle_{n,S}$ scheme. This allows the simulator to know all the $\langle\cdot\rangle_{n,S}$ shares held by the adversary for the triples generated by $\mathcal{F}_{\mathsf{Prep},\langle\cdot\rangle_{\mathsf{n,S}}}^{\mathbf{SP},\mathbf{SP}\cap\mathcal{A}}$, and random elements generated by $\mathcal{F}_{\mathsf{PRSS},\langle\cdot\rangle_{\mathsf{n,S}}}^{\mathbf{SP},\mathbf{SP}\cap\mathcal{A}}$.

Initialise : In the initialise step, the simulator extracts the $\widetilde{\alpha_i}$ for $i \in \mathbf{CP} \cap \mathcal{A}$ from the sharing it receives from the adversary. This extraction requires that enough shares are seen by the simulator, meaning that $\mathbf{SP}^r \setminus \mathcal{A}$ must be a qualified set. The simulator also sets an abort flag to true if the re-sharing to the honest parties is not consistent, which would be detected later in the **CheckOpening**. The simulator creates dummy sharing of $\widehat{\alpha_i}$ for $i \in \mathbf{CP} \setminus \mathcal{A}$ and from it, it computes $\langle\widehat{\alpha_i}\rangle_{r,S}$. Then forwards the $\widetilde{\alpha_i}$ of the adversary to the functionality $\mathcal{F}_{\mathsf{Prep},[\![\cdot]\!]_{\mathsf{m}}}^{\mathbf{CP},\mathbf{CP}\cap\mathcal{A}}$, and keeps in memory $\langle\widehat{\alpha_i}\rangle_{r,S}$ and $\langle\widetilde{\alpha_i}\rangle_{r,S}$ to be able to respond to read request to the ledger. The functionality will sample $\alpha \leftarrow\!\!\$\, \mathbb{F}_p$ and α_i for $i \in \mathbf{CP} \setminus \mathcal{A}$ such that $\alpha = \sum_{i\in\mathbf{CP}\setminus\mathcal{A}} \alpha_i + \sum_{i\in\mathbf{CP}\cap\mathcal{A}} \widetilde{\alpha_i}$. We note that in the end the environment has access to α , the α_i for $i \in \mathbf{CP} \setminus \mathcal{A}$, the $\widetilde{\alpha_i}$ for $i \in \mathbf{CP} \cap \mathcal{A}$, $\langle\widetilde{\alpha_i}\rangle_{r,S}^j$ for $i \in \mathbf{CP} \cap \mathcal{A}$ and $\langle\widehat{\alpha_i}\rangle_{r,S}^j$ for $i \in \mathbf{CP} \setminus \mathcal{A}$ for all $\iota(j) \in \mathbf{SP}^r \cap \mathcal{A}$. But because $\mathbf{SP}^r \cap \mathcal{A} \in \Delta$ and because the environment does not have access to the internal states of the honest parties, it is not able to check for correctness of the dummy $\langle\widetilde{\alpha_i}\rangle_{r,S}$ against the α_i for $i \in \mathbf{CP} \setminus \mathcal{A}$. Therefore, the *dummy* $\langle\widetilde{\alpha_i}\rangle_{r,S}$ made by the simulator looks random to the environment, as would the sharing of the *real* α_i.

SFeedAndMAC : Each parties in $\mathbf{SP}^r \cap \mathcal{A}$ need to $[.]_m$ re-share their shares for the value x and also for the c_x term of the associated triple to parties in $\mathbf{CP} \setminus \mathcal{A}$. As in [21] we note that having a corrupt party in $\mathbf{CP} \cap \mathcal{A}$ to receive a share of the re-share from a party in $\mathbf{SP} \cap \mathcal{A}$ and then introduce an error is equivalent to have the party in $\mathbf{SP} \cap \mathcal{A}$ to send a different re-share to the party in $\mathbf{CP} \cap \mathcal{A}$. Therefore the simulator samples the shares destined to the corrupt parties $\mathbf{CP} \cap \mathcal{A}$ as if no error was introduced. This is made possible because the adversary knows from the **Offline** step what are the shares held by $\mathbf{SP}^r \cap \mathcal{A}$. By doing so, the simulator produces re-sharings that do not contain any error and are consistent to what the environment expects. That means that the simulator receives $\{\widetilde{x_i^j}\}_{i\in\mathbf{CP}\setminus\mathcal{A}}$ for the re-sharing of $\langle x\rangle_{n,S}^j$ from the adversary which controls the shares indexed by $j \in \iota^{-1}(\mathbf{SP}^r \cap \mathcal{A})$, and the simulator samples $\{\widehat{x_i^j}\}_{i\in\mathbf{CP}\cap\mathcal{A}}$ subject to $\sum_{i\in\mathbf{CP}\setminus\mathcal{A}} \widetilde{x_i^j} + \sum_{i\in\mathbf{CP}\cap\mathcal{A}} \widehat{x_i^j} = \langle x\rangle_{n,S}^j$. The proof is very similar to [21]. The idea is that when the adversary re-shares to the honest parties, it does not commit to any error, because the error is only defined when all the shares are set, during the computing phase of **SP**. Thus at the end the view for the environment is as it expects, containing the correct error, introduced by the adversary when it sets the shares for $\mathbf{CP} \cap \mathcal{A}$ for itself. Namely, at some point the adversary will decide on $\{\widetilde{x_i^j}\}_{i\in\mathbf{CP}\cap\mathcal{A}}$ for $j \in \iota^{-1}(\mathbf{SP} \cap \mathcal{A})$ in addition to the $\{\widetilde{x_i^j}\}_{i\in\mathbf{CP}\setminus\mathcal{A}}$ that it sent to the simulator. It is only at this point that the error introduced by the adversary is really fixed. In consequence, at the end of the protocol the environment will end up with the faulty shares fixed by the adversary for itself. Thus fixing the error in combination with the shares forwarded to the ideal functionality. We note that the environment lacks access to the internal values of the honest parties. Therefore, even if the simulator does not know the values sampled by the functionality $\mathcal{F}_{\mathsf{Prep},[\![\cdot]\!]_{\mathsf{m}}}^{\mathbf{CP},\mathbf{CP}\cap\mathcal{A}}$, nor does the

environment. This means that the access structures implies that there are always enough undefined shares to enable the simulation of the offline phase during the computing phase of **CP**. In particular the re-sharing of *dummy* values done in step (3) of the macro in the simulator looks random to the environment, as would the re-sharing of the *real* values. In the last communication phase of the macro, the simulator receives from the adversary the shares of the two substractions. Thanks to the MAC extraction during the **Initialise** step, the simulator checks them for correctness and abort if need be to emulate the **OpenCheck** subroutine.

Preprocessing Apart from the call to SFeedAndMAC which security is discussed above, the simulator only has to make sure that the r^k values that parties in $\mathbf{CP} \cap \mathcal{A}$ should receive corresponds to the sharing the honest parties have. That is why, when the r^k is destined to a party controlled by the adversary r^k is sent to the $\mathcal{F}_{\mathsf{Prep},[\![\cdot]\!]_\mathsf{m}}^{\mathbf{CP},\mathbf{CP}\cap\mathcal{A}}$ in step (2), and then revealed to the adversarial party in step (4).

3.3 Performance

We have designed our protocol such that the computationally expensive offline phase of the SPDZ protocol can be outsourced to a set of service providers. By analyzing the above protocol, we observe that the cost for parties in **CP** is relatively low, as they only need to compute one encryption per parties in $\mathbf{SP}^r$ (which sums up to r public-key encryptions), write their request on the ledger and receive 12 field elements per triples and 4 field elements per input mask. Those figures are to compare with the offline phase of SPDZ [17]. For a prime of bit length 64 and two parties [17] advertises a communication of 9 kbit per triples, whereas in theory our protocol only requires $12 \cdot 2 \cdot 64 = 1536$ bits of communication. We also note that the use of a ledger is costly in terms of latency, from a few seconds to a few minutes depending on the underlying agreement procedure, but cheap in terms of computation. Whereas the FHE operations (or OTs) require some computational power. Therefore we argue that our protocol provides an interesting trade-off in terms of communication and computational power, for low-power devices.

4 Conclusion

In this paper we have shown that Beaver's vision of commodity cryptography exposed in [3] is still relevant in today's cryptography. Especially now, with the trend on computation on encrypted data which requires computationally heavy public key cryptography. But also the growth of cloud computing, with service providers that have access to extensive computing power.

We have defined and proved secure a new protocol which can help a group of low-energy computing devices to achieve secure computation, with the help of the cloud. In fact, our protocol allows for a set of computing parties to outsource the offline phase of the SPDZ [11] protocol. Doing so, the computing parties are left with computing the online phase, which is mainly information theoretic primitives, making it computationally cheap to run.

Our communication model does not fall exactly into Beaver's vision of commodity cryptography. However, after the initial communication amongst service providers to produce raw triples, they need not to communicate. Which makes the actual re-sharing and MAC-ing procedure to match the communication model envisioned by Beaver. Plus this work improves on previous work where

the service providers needed to communicate throughout the protocol, and induces an overhead of only four triples to re-share one to the computing parties.

Because we use a public, immutable ledger for logging all requests for commodities, it might be of interest in the future to look at how one could leverage the functionalities offered by smart contracts to make TaaS available on a blockchain.

Acknowledgements

This work has been supported in part by ERC Advanced Grant ERC-2015-AdG-IMPaCT, by the Defense Advanced Research Projects Agency (DARPA) and Space and Naval Warfare Systems Center, Pacific (SSC Pacific) under contract No. N66001-15-C-4070 and FA8750-19-C-0502, by the Office of the Director of National Intelligence (ODNI), Intelligence Advanced Research Projects Activity (IARPA) via Contract No. 2019-1902070006, and by the FWO under an Odysseus project GOH9718N. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the ERC, United States Air Force, IARPA, DARPA or FWO.

References

1. Asharov, G., Jain, A., López-Alt, A., Tromer, E., Vaikuntanathan, V., Wichs, D.: Multiparty computation with low communication, computation and interaction via threshold FHE. In: Pointcheval, D., Johansson, T. (eds.) Advances in Cryptology – EUROCRYPT 2012. Lecture Notes in Computer Science, vol. 7237, pp. 483–501. Springer, Heidelberg, Germany, Cambridge, UK (Apr 15–19, 2012)
2. Beaver, D.: Correlated pseudorandomness and the complexity of private computations. In: 28th Annual ACM Symposium on Theory of Computing. pp. 479–488. ACM Press, Philadephia, PA, USA (May 22–24, 1996)
3. Beaver, D.: Commodity-based cryptography (extended abstract). In: 29th Annual ACM Symposium on Theory of Computing. pp. 446–455. ACM Press, El Paso, TX, USA (May 4–6, 1997)
4. Bendlin, R., Damgård, I., Orlandi, C., Zakarias, S.: Semi-homomorphic encryption and multiparty computation. In: Paterson, K.G. (ed.) Advances in Cryptology – EUROCRYPT 2011. Lecture Notes in Computer Science, vol. 6632, pp. 169–188. Springer, Heidelberg, Germany, Tallinn, Estonia (May 15–19, 2011)
5. Carter, H., Lever, C., Traynor, P.: Whitewash: Outsourcing garbled circuit generation for mobile devices. Cryptology ePrint Archive, Report 2014/224 (2014), `http://eprint.iacr.org/2014/224`
6. Carter, H., Mood, B., Traynor, P., Butler, K.R.B.: Secure outsourced garbled circuit evaluation for mobile devices. In: King, S.T. (ed.) USENIX Security 2013: 22nd USENIX Security Symposium. pp. 289–304. USENIX Association, Washington, DC, USA (Aug 14–16, 2013)
7. Carter, H., Mood, B., Traynor, P., Butler, K.R.B.: Outsourcing secure two-party computation as a black box. In: Reiter, M., Naccache, D. (eds.) CANS 15: 14th International Conference on Cryptology and Network Security. pp. 214–222. Lecture Notes in Computer Science, Springer, Heidelberg, Germany, Marrakesh, Morocco (Dec 10–12, 2015)
8. Cramer, R., Damgård, I., Ishai, Y.: Share conversion, pseudorandom secret-sharing and applications to secure computation. In: Kilian, J. (ed.) TCC 2005: 2nd Theory of Cryptography Conference. Lecture Notes in Computer Science, vol. 3378, pp. 342–362. Springer, Heidelberg, Germany, Cambridge, MA, USA (Feb 10–12, 2005)
9. Damgård, I., Geisler, M., Krøigaard, M., Nielsen, J.B.: Asynchronous multiparty computation: Theory and implementation. In: Jarecki, S., Tsudik, G. (eds.) PKC 2009: 12th International Conference on Theory and Practice of Public Key Cryptography. Lecture Notes in Computer Science, vol. 5443, pp. 160–179. Springer, Heidelberg, Germany, Irvine, CA, USA (Mar 18–20, 2009)
10. Damgård, I., Keller, M., Larraia, E., Pastro, V., Scholl, P., Smart, N.P.: Practical covertly secure MPC for dishonest majority - or: Breaking the SPDZ limits. In: Crampton, J., Jajodia, S., Mayes, K. (eds.) ESORICS 2013: 18th European Symposium on Research in Computer Security. Lecture Notes in Computer Science, vol. 8134, pp. 1–18. Springer, Heidelberg, Germany, Egham, UK (Sep 9–13, 2013)

11. Damgård, I., Pastro, V., Smart, N.P., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. In: Safavi-Naini, R., Canetti, R. (eds.) Advances in Cryptology – CRYPTO 2012. Lecture Notes in Computer Science, vol. 7417, pp. 643–662. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 19–23, 2012)
12. Demmler, D., Schneider, T., Zohner, M.: Ad-hoc secure two-party computation on mobile devices using hardware tokens. In: Fu, K., Jung, J. (eds.) USENIX Security 2014: 23rd USENIX Security Symposium. pp. 893–908. USENIX Association, San Diego, CA, USA (Aug 20–22, 2014)
13. Frederiksen, T.K., Keller, M., Orsini, E., Scholl, P.: A unified approach to MPC with preprocessing using OT. In: Iwata, T., Cheon, J.H. (eds.) Advances in Cryptology – ASIACRYPT 2015, Part I. Lecture Notes in Computer Science, vol. 9452, pp. 711–735. Springer, Heidelberg, Germany, Auckland, New Zealand (Nov 30 – Dec 3, 2015)
14. Kamara, S., Mohassel, P., Riva, B.: Salus: a system for server-aided secure function evaluation. In: Yu, T., Danezis, G., Gligor, V.D. (eds.) ACM CCS 2012: 19th Conference on Computer and Communications Security. pp. 797–808. ACM Press, Raleigh, NC, USA (Oct 16–18, 2012)
15. Karchmer, M., Wigderson, A.: On span programs. In: Proceedings of Structures in Complexity Theory. pp. 102–111 (1993)
16. Keller, M., Orsini, E., Scholl, P.: MASCOT: Faster malicious arithmetic secure computation with oblivious transfer. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 2016: 23rd Conference on Computer and Communications Security. pp. 830–842. ACM Press, Vienna, Austria (Oct 24–28, 2016)
17. Keller, M., Pastro, V., Rotaru, D.: Overdrive: Making SPDZ great again. In: Nielsen, J.B., Rijmen, V. (eds.) Advances in Cryptology – EUROCRYPT 2018, Part III. Lecture Notes in Computer Science, vol. 10822, pp. 158–189. Springer, Heidelberg, Germany, Tel Aviv, Israel (Apr 29 – May 3, 2018)
18. Mohassel, P., Orobets, O., Riva, B.: Efficient server-aided 2PC for mobile phones. Proceedings on Privacy Enhancing Technologies 2016(2), 82–99 (Apr 2016)
19. Rabin, M.O.: Transaction protection by beacons. J. Comput. Syst. Sci. 27(2), 256–267 (1983), https://doi.org/10.1016/0022-0000(83)90042-9
20. Riazi, M.S., Weinert, C., Tkachenko, O., Songhori, E.M., Schneider, T., Koushanfar, F.: Chameleon: A hybrid secure computation framework for machine learning applications. In: Kim, J., Ahn, G.J., Kim, S., Kim, Y., López, J., Kim, T. (eds.) ASIACCS 18: 13th ACM Symposium on Information, Computer and Communications Security. pp. 707–721. ACM Press, Incheon, Republic of Korea (Apr 2–6, 2018)
21. Scholl, P., Smart, N.P., Wood, T.: When it's all just too much: Outsourcing MPC-preprocessing. In: O'Neill, M. (ed.) 16th IMA International Conference on Cryptography and Coding. Lecture Notes in Computer Science, vol. 10655, pp. 77–99. Springer, Heidelberg, Germany, Oxford, UK (Dec 12–14, 2017)
22. Smart, N.P., Wood, T.: Error detection in monotone span programs with application to communication-efficient multi-party computation. In: Matsui, M. (ed.) Topics in Cryptology – CT-RSA 2019. Lecture Notes in Computer Science, vol. 11405, pp. 210–229. Springer, Heidelberg, Germany, San Francisco, CA, USA (Mar 4–8, 2019)
23. Yao, A.C.C.: Protocols for secure computations (extended abstract). In: 23rd Annual Symposium on Foundations of Computer Science. pp. 160–164. IEEE Computer Society Press, Chicago, Illinois (Nov 3–5, 1982)

Commodity MPC

	SP		**CP**
1 :			
2 :	$(\langle a\rangle_{n,S}, \langle b\rangle_{n,S}, \langle c\rangle_{n,S}) \leftarrow \mathcal{F}^{\mathbf{SP},\mathcal{A}\cap\mathbf{SP}}_{\mathsf{Prep},\langle\cdot\rangle_{\mathsf{n,S}}}$		
3 :	$(\langle a_a\rangle_{n,S}, \langle b_a\rangle_{n,S}, \langle c_a\rangle_{n,S}) \leftarrow \mathcal{F}^{\mathbf{SP},\mathcal{A}\cap\mathbf{SP}}_{\mathsf{Prep},\langle\cdot\rangle_{\mathsf{n,S}}}$		
4 :	$(\langle a_b\rangle_{n,S}, \langle b_b\rangle_{n,S}, \langle c_b\rangle_{n,S}) \leftarrow \mathcal{F}^{\mathbf{SP},\mathcal{A}\cap\mathbf{SP}}_{\mathsf{Prep},\langle\cdot\rangle_{\mathsf{n,S}}}$		
5 :	$(\langle a_c\rangle_{n,S}, \langle b_c\rangle_{n,S}, \langle c_c\rangle_{n,S}) \leftarrow \mathcal{F}^{\mathbf{SP},\mathcal{A}\cap\mathbf{SP}}_{\mathsf{Prep},\langle\cdot\rangle_{\mathsf{n,S}}}$		
	[Lines 6-23] SP need not interacting with each other		
6 :			
7 :			$[\alpha]_m \leftarrow GenMacKey()$
8 :			Select $\mathbf{SP}^r = \{SP_1, ..., SP_r\}$
9 :		Select $\mathbf{SP}^r$ ← **Broadcast**	
	From here we consider only $\mathbf{SP}^r$ on the left		
10 :			
11 :			$\langle\alpha\rangle_{r,=}$**TransformMAC**$([\alpha]_m)$
12 :		$\langle\alpha\rangle_{r,S}$ ← **Point-to-Point**	
	Repeat following for $\langle b\rangle_{n,S}$ and $\langle c\rangle_{n,S}$		
13 :			
14 :	$[a]_m =$ **TransformShare**$(\langle a\rangle_{n,S})$		
15 :	$[c_a]_m =$ **TransformShare**$(\langle c_a\rangle_{n,S})$		
16 :		$[a]_m$ → **Point-to-Point**	
17 :		$[c_a]_m$ → **Point-to-Point**	
18 :		$\langle(a - a_a)\rangle_{n,S}$ → **Broadcast**	
19 :		$\langle(\alpha - b_a)\rangle_{n,S}$ → **Broadcast**	
20 :			**CheckOpening**$(\langle(a - a_a)\rangle_{n,S})$
21 :			**CheckOpening**$(\langle(\alpha - b_a)\rangle_{r,S})$
22 :			$[\alpha a]_m = -(a - a_a)(\alpha - b_a) + [c_a]_m +$
23 :			$(\alpha - b_a)[a]_m + (a - a_a)[\alpha]_m$
	End repeat		
24 :			
25 :			

Fig. 7: Protocol for commodity MPC

TransformMAC (IN: $[\alpha]_m$), (OUT: $\langle\alpha\rangle_{r,S}$)

For each $\mathsf{CP}_i \in \mathbf{CP}$:

1. CP_i creates a $\langle\alpha_i\rangle_{r,S} = (\alpha_{i,1}, \ldots, \alpha_{i,k})$ sharing of its share α_i of the MAC key.
2. CP_i sends $\alpha_{i,j}$ to $\iota(j) \cap \mathbf{SP}^r$.

The parties in $\mathbf{SP}^r$ set $\langle\alpha\rangle_{r,S} = \sum_{i=1}^{m} \langle\alpha_i\rangle_{r,S}$

Figure 8: Subroutine TransformMAC

TransformShare (IN: $\langle a\rangle_{n,S}$), (OUT: $[a]_m$)

For each $j \in \iota^{-1}(\mathbf{SP}^r)$:

1. Party $\iota(j)$ samples $(a_1^j, \ldots, a_m^j) \leftarrow \mathbb{F}_p^m$ s.t. $\sum_{\mathsf{CP}_i \in \mathbf{CP}} a_i^j = \langle a\rangle_{n,S}^j$
2. Party $\iota(j)$ sends a_i^j to each $CP_i \in CP$

Every $\mathsf{CP}_i \in \mathbf{CP}$ sets $[a]_m^i = \sum_{j \in \iota^{-1}(\mathbf{SP}^r)} \beta_j a_i^j$.

Note: From the definition of the $\langle . \rangle_{n,S}$ sharing, there exists a linear combination of the shares of $\langle a\rangle_{n,S}$ which sums up to a. We call $\beta_1, \ldots, \beta_k$ the public coefficients of this linear combination.

Figure 9: Subroutine TranformShare

CheckOpening (IN: $\langle a\rangle_{n,S}$), (OUT: a or $\perp$)

Note that in this subroutine, every party in ***CP*** *has access to the shares of* $\langle a\rangle_{n,S}$ *which belong to parties in* $\boldsymbol{SP}^r$.
For each $\mathsf{CP}_i \in \mathbf{CP}$

1. If the shares of $\langle a\rangle_{n,S}^{\iota^{-1}(\mathbf{SP}^r)}$ are consistent output $a = \langle \mathbf{a}_{\mathbf{SP}^r}, \boldsymbol{\lambda}_{\mathbf{SP}^r}\rangle = \sum_{j \in \iota^{-1}(\mathbf{SP}^r)} \beta_j \langle a\rangle_{n,S}^j$
2. Else otput $\perp$.

Figure 10: Subroutine CheckOpening

Protocol $\Pi_{Prep}^{\mathbf{SP}\to\mathbf{CP},\mathcal{A}}$

Offline: Parties in **SP** precompute $4n_T + n_R$ triples and n_R random shares:

- For i in $\{1,\ldots,n_T\}$
  1. The parties in **SP** call $\mathcal{F}_{\mathsf{Prep},\langle\cdot\rangle_{n,S}}^{\mathbf{SP},\mathbf{SP}\cap\mathcal{A}}$ to obtain $(\langle a^i\rangle_{n,S}, \langle b^i\rangle_{n,S}, \langle c^i\rangle_{n,S})$.
  2. The parties in **SP** call $\mathcal{F}_{\mathsf{Prep},\langle\cdot\rangle_{n,S}}^{\mathbf{SP},\mathbf{SP}\cap\mathcal{A}}$ to obtain $(\langle a_a^i\rangle_{n,S}, \langle b_a^i\rangle_{n,S}, \langle c_a^i\rangle_{n,S})$.
  3. The parties in **SP** call $\mathcal{F}_{\mathsf{Prep},\langle\cdot\rangle_{n,S}}^{\mathbf{SP},\mathbf{SP}\cap\mathcal{A}}$ to obtain $(\langle a_b^i\rangle_{n,S}, \langle b_b^i\rangle_{n,S}, \langle c_b^i\rangle_{n,S})$.
  4. The parties in **SP** call $\mathcal{F}_{\mathsf{Prep},\langle\cdot\rangle_{n,S}}^{\mathbf{SP},\mathbf{SP}\cap\mathcal{A}}$ to obtain $(\langle a_c^i\rangle_{n,S}, \langle b_c^i\rangle_{n,S}, \langle c_c^i\rangle_{n,S})$.
- For i in $\{1,\ldots,n_R\}$
  1. The parties in **SP** call $\mathcal{F}_{\mathsf{PRSS},\langle\cdot\rangle_{n,S}}^{\mathbf{SP},\mathbf{SP}\cap\mathcal{A}}$ to obtain $\langle r^i\rangle_{n,S}$.
  2. The parties in **SP** call $\mathcal{F}_{\mathsf{Prep},\langle\cdot\rangle_{n,S}}^{\mathbf{SP},\mathbf{SP}\cap\mathcal{A}}$ to obtain $(\langle a_r^i\rangle_{n,S}, \langle b_r^i\rangle_{n,S}, \langle c_r^i\rangle_{n,S})$.

Initialise: All sets of computing parties **CP** do:
On input (*Initialise*, p, T', R_1, ..., R_m) from all parties in **CP**:

1. Each party $\mathsf{CP}_i \in \mathbf{CP}$ samples $\alpha_i \leftarrow\!\!\$\, \mathbb{F}_p$.
2. For each $\mathsf{CP}_i \in \mathbf{CP}$, Party CP_i creates $\langle\alpha_i\rangle_{r,S}$.
3. For each $\mathsf{CP}_i \in \mathbf{CP}$, Party CP_i computes $\{Enc_{sk_{\iota(j)}}(\langle\alpha_i\rangle_{r,S}^j)\}_{j\in\iota^{-1}(\mathbf{SP}^r)}$.
4. For each $\mathsf{CP}_i \in \mathbf{CP}$, Party CP_i sends ($\mathsf{RESERVE}$, T', R_1, ..., R_m, $\{Enc_{sk_{\iota(j)}}(\langle\alpha_i\rangle_{r,S}^j)\}_{j\in\iota^{-1}(\mathbf{SP}^r)}$, $\mathbf{SP}^r$, **CP**) to $\mathcal{F}_{\mathsf{Ledger}}$. If it fails, the parties in **CP** abort.

Macro: FeedAndMAC($\langle x\rangle_{n,S}, (\langle a_x\rangle_{n,S}, \langle b_x\rangle_{n,S}, \langle c_x\rangle_{n,S})$):

1. The parties execute **TransformShare**($\langle x\rangle_{n,s}$) and **TransformShare**($\langle c_x\rangle_{n,s}$).
2. For each $j \in \iota^{-1}(\mathbf{SP}^r)$, Party $\iota(j)$ sends $(\langle x\rangle_{n,S}^j - \langle a_x\rangle_{n,S}^j)$ and $(\langle\alpha\rangle_{r,S}^j - \langle b_x\rangle_{n,S}^j)$ to every parties in **CP** (broadcast).
3. For each $\mathsf{CP}_i \in \mathbf{CP}$, Party CP_i executes **checkOpening**($\{\langle x\rangle_{n,S}^j - \langle a_x\rangle_{n,S}^j)\}_{j\in\iota^{-1}(\mathbf{SP}^r)}$) and **checkOpening**($\{(\langle\alpha\rangle_{r,S}^j - \langle b_x\rangle_{n,S}^j)\}_{j\in\iota^{-1}(\mathbf{SP}^r)}$). If one of the opening return $\perp$ the parties abort.
4. For each $\mathsf{CP}_i \in \mathbf{CP}$, Party CP_i sets $\gamma(x)^i = -(x-a_x)(\alpha-b_x)\delta_{i,1} + c_x^i + (x-a_x)\alpha_i + (\alpha-b_x)x^i$ *(with $\delta_{i,j} = (i \stackrel{?}{=} j)$ the Kronecker delta).*

Preprocessing: All parties $\mathsf{SP}_i \in \mathbf{SP}$ send ($\mathsf{READ}, \mathsf{SP}_i$) to $\mathcal{F}_{\mathsf{Ledger}}$ and for every set **CP** they are involved with:

- For $\mathsf{CP}_i \in \mathbf{CP}$
  1. For j in R_i
     (a) Run FeedAndMAC($\langle r^j\rangle_{n,S}, (\langle a_r^j\rangle_{n,S}, \langle b_r^j\rangle_{n,S}, \langle c_r^j\rangle_{n,S})$.
     (b) For each $k \in \iota^{-1}(\mathbf{SP}^r)$, Party $\iota(k)$ sends $\langle r^j\rangle_{n,S}^k$ to Party $\mathsf{CP}_i \in \mathbf{CP}$.
     (c) Party $\mathsf{CP}_i \in \mathbf{CP}$ executes **checkOpening**($\langle r^j\rangle_{n,S}$) and gets r^j.
- For i in T'
  1. run FeedAndMAC($\langle a^i\rangle_{n,S}, (\langle a_a^i\rangle_{n,S}, \langle b_a^i\rangle_{n,S}, \langle c_a^i\rangle_{n,S})$)
  2. do the same for $\langle b^i\rangle_{n,S}$ and $\langle c^i\rangle_{n,S}$.

Figure 11: Protocol $\Pi_{Prep}^{\mathbf{SP}\to\mathbf{CP},\mathcal{A}}$

Simulator $\mathcal{S}_{Prep}^{\mathbf{CP},\mathbf{CP}\cap\mathcal{A}}$ (1/2)

*The set of corrupt parties is set to be the set of corrupt parties in **CP** with the corrupt parties in **SP**.*

Setup: We assume a set-up phase where the simulator provides the adversary with public keys to emulate the secure communication channels to parties in the sets $\mathbf{SP} \setminus \mathcal{A}$ and $\mathbf{CP} \setminus \mathcal{A}$.

Offline: The simulator receives multiple calls to the two functionalities $\mathcal{F}_{\mathsf{Prep},\langle\cdot\rangle_{\mathsf{n,S}}}^{\mathbf{SP},\mathbf{SP}\cap\mathcal{A}}$ and $\mathcal{F}_{\mathsf{PRSS},\langle\cdot\rangle_{\mathsf{n,S}}}^{\mathbf{SP},\mathbf{SP}\cap\mathcal{A}}$ from $\mathbf{SP} \cap \mathcal{A}$. For each call the simulator emulates the functionality.

1. i^{th} call to $\mathcal{F}_{\mathsf{Prep},\langle\cdot\rangle_{\mathsf{n,S}}}^{\mathbf{SP},\mathbf{SP}\cap\mathcal{A}}$
   - The simulator samples $a^i, b^i \leftarrow\$ \mathbb{F}_p$ and sends $\langle a^i\rangle_{n,S}^{\iota^{-1}(\mathbf{SP}\cap\mathcal{A})}$ and $\langle b^i\rangle_{n,S}^{\iota^{-1}(\mathbf{SP}\cap\mathcal{A})}$ to the adversary.
   - The simulator receives $\langle\widetilde{c^i}\rangle_{n,S}^{\iota^{-1}(\mathbf{SP}\cap\mathcal{A})}$ from the adversary, and if possible computes $\langle c^i\rangle_{n,S} = \langle a^i \cdot b^i\rangle_{n,S}$ such that $\langle\widetilde{c^i}\rangle_{n,S}^{\iota^{-1}(\mathbf{SP}\cap\mathcal{A})} = \langle c^i\rangle_{n,S}^{\iota^{-1}(\mathbf{SP}\cap\mathcal{A})}$
   - Otherwise, set abort flag.
2. i^{th} call to $\mathcal{F}_{\mathsf{PRSS},\langle\cdot\rangle_{\mathsf{n,S}}}^{\mathbf{SP},\mathbf{SP}\cap\mathcal{A}}$
   - Simulator receives $PRSS(count)$ from $\mathcal{A}$. If $count \neq i$ set abort flag
   - Sample $a^i \leftarrow\$ \mathbb{F}_p$, compute $\langle a^i\rangle_{n,S}$ and send $\langle a^i\rangle_{n,S}^{\iota^{-1}(\mathbf{SP}\cap\mathcal{A})}$ to the adversary.

Initialise: The simulator receives ($Initialise$, p, T', R_1, …, R_m from the adversary:

1. Simulator forwards the message to $\mathcal{F}_{\mathsf{Prep},[\![\cdot]\!]_{\mathsf{m}}}^{\mathbf{CP},\mathbf{CP}\cap\mathcal{A}}$
2. Simulator receives ($\mathsf{RESERVE}$, T', R_1, …, R_m, $\{Enc_{sk_{\iota(j)}}(\langle\widetilde{\alpha_i}\rangle_{r,S}^j)\}_{j\in\iota^{-1}(\mathbf{SP}^r)}$, $\mathbf{SP}^r$, $\mathbf{CP}$) from $\mathbf{CP} \cap \mathcal{A}$. It emulates the $\mathcal{F}_{\mathsf{Ledger}}$ functionality, and returns FAIL if needed.
3. The Simulator is able to decrypt the $\langle\widetilde{\alpha_i}\rangle_{r,S}^j$ destined to honest $\mathbf{SP}^r \setminus \mathcal{A}$, and therefore can reconstruct $\widetilde{\alpha_i}$ for $\mathsf{CP}_i \in \mathbf{CP} \cap \mathcal{A}$ if the sharing is correct or set an abort flag otherwise.
4. Simulator also samples *dummy* $\widehat{\alpha_i} \leftarrow\$ \mathbb{F}_p$ for $\mathsf{CP}_i \in \mathbf{CP} \setminus \mathcal{A}$, and computes $\langle\widehat{\alpha_i}\rangle_{r,S}$ for those.
5. Simulator sends $\{\widetilde{\alpha_i}\}_{\mathsf{CP}_i\in\mathbf{CP}\cap\mathcal{A}}$ to $\mathcal{F}_{\mathsf{Prep},[\![\cdot]\!]_{\mathsf{m}}}^{\mathbf{CP},\mathbf{CP}\cap\mathcal{A}}$ and keeps both $\langle\widetilde{\alpha_i}\rangle_{r,S}^j$ $\langle\widehat{\alpha_i}\rangle_{r,S}^j$ as the ledger would for $j \in \iota^{-1}(SP^r)$.

Figure 12: Simulator (1/2)

Simulator $\mathcal{S}_{Prep}^{\mathbf{CP},\mathbf{CP}\cap\mathcal{A}}$ (2/2)

Macro: SFeedAndMAC($\langle x\rangle_{n,S}, (\langle a_x\rangle_{n,S}, \langle b_x\rangle_{n,S}, \langle c_x\rangle_{n,S})$)

1. The simulator receives $\{\widetilde{x_i^j}\}_{\mathsf{CP}_i\in\mathbf{CP}\setminus\mathcal{A}}$ and $\{\widetilde{c_i^j}\}_{\mathsf{CP}_i\in\mathbf{CP}\setminus\mathcal{A}}$ from the adversary for $j \in \iota^{-1}(\mathbf{SP}^r \cap \mathcal{A})$
   - The simulator samples *dummy* $\{\widehat{x_i^j}\}_{\mathsf{CP}_i\in\mathbf{CP}\cap\mathcal{A}}$ for $j \in \iota^{-1}(\mathbf{SP}^r\cap\mathcal{A})$ s.t. $\sum_{\mathsf{CP}_i\in\mathbf{CP}\cap\mathcal{A}} \widehat{x_i^j} + \sum_{\mathsf{CP}_i\in\mathbf{CP}\setminus\mathcal{A}} \widetilde{x_i^j} = \langle x\rangle_{r,S}^j$.
   - The simulator repeats the previous step for c
2. The simulator samples $(\widehat{x_1^j}, \ldots, \widehat{x_m^j})$ for $j \in \iota^{-1}(\mathbf{SP}^r \setminus \mathcal{A})$ s.t. $\sum_{\mathsf{CP}_i\in\mathbf{CP}} \widehat{x_i^j} = \langle x\rangle_{r,S}^j$ and does the same for $(\widehat{c_{x,1}^j}, \ldots, \widehat{c_{x,Q}^j})$.
3. For each $\mathsf{CP}_i \in \mathbf{CP}\cap\mathcal{A}$ the simulator sends $\{x_i^j\}_{j\in\iota^{-1}(\mathbf{SP}^r)}$ and $\{c_{x,i}^j\}_{j\in\iota^{-1}(\mathbf{SP}^r)}$ to CP_i
4. For each $j \in \iota^{-1}(\mathbf{SP}^r\cap\mathcal{A})$ the simulator receives $(\langle x\rangle_{n,S}^j - \langle a_x\rangle_{n,S}^j)$ and $(\langle\alpha\rangle_{r,S}^j - \langle b_x\rangle_{n,S}^j)$. The simulator transfers these values to $\mathsf{CP}_i \in \mathbf{CP}\cap\mathcal{A}$ along with the one simulated for $j \in \iota^{-1}(\mathbf{SP}^r\setminus\mathcal{A})$. If the abort flag was set during initialize or if values are wrong, abort.
5. The simulator sets $\gamma(x)^i = -(x-a_x)(\alpha-b_x)\delta_{1,i} + \sum_{\iota(j)\in\mathbf{SP}^r} c_{x,i}^j + (x-a_x)\alpha_i + (\alpha-b_x)\sum_{j\in\iota^{-1}(\mathbf{SP}^r)} x_i^j$ for $i \in \mathbf{CP}\cap\mathcal{A}$ and sends $\{\sum_{j\in\iota^{-1}(\mathbf{SP}^r)} x_i^j, \gamma(x)^i : \text{for } i \in \mathbf{CP}\cap\mathcal{A}\}$ to $\mathcal{F}_{\mathsf{Prep},[\![\cdot]\!]_\mathsf{m}}^{\mathbf{CP},\mathbf{CP}\cap\mathcal{A}}$

Preprocessing: The simulator receives $(\mathsf{READ}, \mathsf{SP}_i)$ from parties in $\mathbf{SP}$, and responds appropriately with adversary and *dummy* data to each request, as would the $\mathcal{F}_{\mathsf{Ledger}}$ functionality. When it has receive $(\mathsf{READ}, \mathsf{SP}_i)$ from all the SP_i parties of a RESERVE request, the simulator does the following:

- For CP_i in $\mathbf{CP}$
  1. For k in R_i.
  2. If CP_i is in $\mathcal{A}$, the simulator sends r^k to the functionality $\mathcal{F}_{\mathsf{Prep},[\![\cdot]\!]_\mathsf{m}}^{\mathbf{CP},\mathbf{CP}\cap\mathcal{A}}$.
  3. The simulator runs SFeedAndMAC($\langle r^k\rangle_{n,S}$, ($\langle a_{r^k}\rangle_{n,S}$,$\langle b_{r^k}\rangle_{n,S}$, $\langle c_{r^k}\rangle_{n,S}$)).
  4. If CP_i is in $\mathcal{A}$, the simulator sends r^k to CP_i.
- For k in T'
  1. The simulator runs SFeedAndMAC($\langle a^k\rangle_{n,S}$, ($\langle a_{a^k}\rangle_{n,S}$,$\langle b_{a^k}\rangle_{n,S}$, $\langle c_{a^k}\rangle_{n,S}$).
  2. Do the same for b^k and c^k.

Figure 13: Simulator (2/2)