When It's All Just Too Much: Outsourcing MPC-Preprocessing

Peter Scholl[1], Nigel P. Smart[2], and Tim Wood[2]

[1] Dept. Computer Science, Aarhus University, Denmark
[2] Dept. Computer Science, University of Bristol, United Kingdom

Abstract. Many modern actively secure multi-party computation protocols make use of a function- and input-independent pre-processing phase. This pre-processing phase is tasked with producing some form of correlated randomness and distributing it to the parties. Whilst the "online" phase of such protocols is exceedingly fast, the bottleneck comes in the pre-processing phase. In this paper we examine situations in which the computing parties in the online phase may want to outsource the pre-processing phase to another set of parties, or to a sub-committee. We examine how this can be done, and also describe situations where this may be a benefit.

1 Introduction

Secure multi-party computation (MPC) is the idea of allowing multiple parties to compute on their combined inputs in a "secure" manner. We use the word secure to mean that the interaction provides no party with any information on the secret inputs of the other parties, bar what can be learned from the output (a property called *privacy* or *secrecy*). In this paper we will focus on protocols which can tolerate a majority of the parties being corrupted. In such a situation we know there is no hope that the honest parties can always obtain the correct output, so we usually require that either the honest parties obtain the correct result, or they abort (with overwhelming probability in the security parameter, λ).

For a long time, MPC remained a theoretical exercise and implementations were impractical. However, much work has recently been undertaken on developing practical MPC protocols in the so-called *pre-processing model*. In this model, the protocol is split up into an offline (a.k.a. pre-processing) phase and an online phase. In the offline phase, the parties execute a protocol which emulates a *trusted dealer* who distributes "raw material" (pre-processed data) to parties; this data is then used up in the online phase as the circuit is evaluated. The advantage of doing this is that the pre-processing involves expensive public key operations which can be isolated to the pre-processing phase. In addition, pre-processed data can be made independent of both the inputs and the circuit, so it can be computed at any point prior to the evaluation of the circuit. The online phase is then executed with (essentially) information theoretic primitives, and is thus very fast.

This protocol idea goes back to Beaver [Bea96]. It was first used in a practical (and implemented) MPC system in the VIFF protocol [DGKN09], which was a protocol system built for the case of honest majority MPC. Modern dishonest majority MPC protocols make use of information theoretic MACs to achieve active security, an idea which stems from [RBO89] In the last five years, combining the pre-processed triple idea of Beaver with these protocols has resulted in a step change in what can be implemented efficiently by MPC protocols.

The first protocol in this area was BDOZ [BDOZ11], which demonstrated that if the number of parties was constant and the parties had access to a functionality which would provide the pre-processed data then the overhead of computing an arithmetic circuit over a large finite field securely is only a constant factor times the work required to compute it in the clear. The SPDZ [DPSZ12] protocol showed that the mere constant factor overhead encountered in the BDOZ protocol holds for any number of parties. Further improvements were presented in [DKL$^+$13] to the SPDZ protocol. In the BDOZ and SPDZ protocols, the pre-processing is produced using forms of homomorphic encryption, and so the protocols are more suited to MPC over a large finite field. In TinyOT [NNOB12], similar results in the two-party case for Boolean circuits were given, where the pre-processing was implemented using oblivious transfer (OT) extension. In [LOS14,BLN$^+$15], the TinyOT protocol was extended to the multi-party case, and the online phase was made consistent (in

terms of computational pattern) with that of the SPDZ protocol from [DKL+13]. Further unification of these protocol families occured with the replacement of the homomorphic encryption based pre-processing phase of SPDZ with an OT based pre-processing [KOS16], forming what is known as the MASCOT protocol. To simplify exposition, since all of these protocols are essentially the same at a high level, in this paper we shall refer to the collective as the "SPDZ family".

As already remarked, the SPDZ family of protocols has an efficient online phase; indeed, the online phase has a number of interesting properties:

- **Computational Efficiency:** Since the online phase is made up of information theoretic primitives, the basic arithmetic operations are incredibly simple, requiring only a constant multiplicative factor increase in the number of operations when compared to evaluating the function in the clear. Before every output operation, the execution of a PRF is also required for MAC checking, but for a large computation this is negligible when measuring performance.
- **Communication Efficiency:** The basic protocol requires interaction for each multiplication operation[1]. This interaction need only be conducted over authenticated channels, rather than private channels, and the communication required grows linearly in the number of players.
- **Deterministic:** Given the correlated randomness from the offline phase, the function to be computed, and the parties' inputs, the online phase is essentially deterministic. Only a small amount of "random data" per party is needed to ensure that dishonest parties are detected in the MAC checking protocol. Indeed this random data can be created in the offline phase and then stored for later use.

The simplicity and efficiency of the online phase, however, comes with a penalty in the offline phase. Using either method (i.e. the homomorphic encryption of SPDZ or the OT method of MASCOT) to generate the pre-processed data, the offline phase requires expensive public key machinery, and in practice is a couple of orders of magnitude slower than the online phase. In some instances, while the online phase is computationally cheap enough to be executed by a relatively low powered computing device, the same device would not be sufficiently powerful to perform the associated offline phase efficiently. This can cause a problem when there are parties in a network with very different computing power. Similarly the offline phase requires the transmission of a larger amount of data per multiplication gate than the online phase. Again, this can be a problem in practice if certain parties are on a slow part of the network.

The offline phase also requires each party to input a large amount of randomness, and it is well known that one of the major challenges of running any cryptography in the real world is the generation of randomness. Small hardware devices may not have the capability of producing random values easily, as they usually have very limited access to good sources of entropy: for example, devices such as mobile phones and tablets still have problems with good entropy sources. Moreover, it does not suffice simply to be able to generate pseudo-random numbers: in many cryptographic applications (including MPC) it is necessary that it be "high quality" randomness. This has led to high-end applications requiring expensive dedicated hardware to generate entropy; however, such dedicated hardware may not be available to all computers in a network. Thus, even in the case of high-end servers executing the MPC protocol, it may easily not be the case that all have access to a sufficient entropy source.

For these reasons, we propose a method of outsourcing the offline pre-processing for the SPDZ family of protocols to a different set of parties. We will let Q denote the set of n_q parties who are to run the online phase; the set Q will outsource the computation of the pre-processing to a set of parties R of size n_r. This set R may be a strict subset of Q, or they could be a completely different set all together, or even a mix of parties who will later be involved in computation and parties who will not. The idea is that Q is unable to execute the pre-processing as an n_q-party protocol, due to some limitation of resources (computation, bandwidth, or randomness, for example), whereas R is "more able" to execute the pre-processing as an n_r-party protocol. Our protocol to perform this outsourcing will also aim to minimise the communication needed to transfer the pre-processing data from the set R to the set Q.

[1] For simplicity of expression we assume the MPC functionality is evaluating an arithmetic circuit over a finite field. This is purely for exposition: in practice the usual MPC tricks to remove the need for circuit based computation will be used.

Of course, for this to make sense it is important that the set Q trust the set R to perform this task, and that the protocol respect this trust relationship. In particular, our protocol will assume an adversary which can corrupt a majority of parties in Q and a majority of parties in R, but that the adversary can neither corrupt all parties in R nor all parties in Q: indeed, in such a situation we clearly would not even be *expected* to guarantee any security. In particular, this means that each honest party in Q believes that there is at least one honest party in R, but they may not know which one is honest.

The fact that the parties do not know which parties in the other network are honest has security implications for the way pre-processing is passed from one network to the other. The naïve method of sending on the pre-processed data (in the case of $n_r \leq n_q$) would be to partition Q into n_r subsets, and then for each party in R to send their data to one set in the partition; it turns out that this method is insecure (using our redistribution procedure), though it only requires minor modification to make it secure. Our protocol creates a cover of Q, $\{Q_i\}_{i \in R}$, using $|R|$ sets, not necessarily disjoint, and associates each subset with a party in R; namely, party $i \in R$ is assigned the set Q_i. The association merely defines the network of secure channels by which secret-shared data amongst the parties in R is reshared amongst the parties in Q. Note that there is no assumption of trust of parties in Q for parties in R they are associated to (i.e. with respect to the cover): the only assumption of trust is that at least one party in each of R and Q is honest. Our protocol will be secure if there is at least one pair (i, Q_i) for which $i \in R$ is an honest party in R and Q_i contains at least one honest party from Q. This raises (at least) three potential ways for the subsets to arise:

- If $R \subseteq Q$ then for each $i \in R$, we can just ensure that Q_i contains i. Then since R and Q each contain an honest party, there must be at least one pair containing (the same) honest party.
- It may be the case that every party in Q trusts at least one party in R already. In this case, our cover, $\{Q_i\}_{i \in R}$, can be produced by letting parties in Q elect which parties in R they want to be associated with. Security will follow because in particular, at least one honest party in R believes there is at least one honest party in Q.
- If no prior trust relation is known then the cover must be defined either deterministically or probabilistically. If deterministically, to satisfy the requirement above we must choose $Q_i = Q$ for all i. This guarantees a pair (i, Q_i) as described above, but results in an inefficient network topology (since each party in R needs a secure channel to each party in Q). Alternatively, we make a probabilistic assignment and derive bounds on n_q and n_r which ensure that the assignment preserves security with overwhelming probability: see Section 4 for details.

The first case above is a reasonably likely scenario. Consider an (n, t)-threshold access structure, in which any set of $t+1$ parties contains an honest party. In this case any set of $t+1$ parties can form the network R and undertake the pre-processing. To pass the data on, these parties need to be associated to the remaining $n-t-1$ parties. Thus each party in R must send to $(n-t-1)/(t+1)$ parties on average. For example, if $n = 20$ and $t = 14$, then any 15 parties perform the pre-processing and each sends to all of the remaining 5 parties. For a multiplication performed in the production of a single triple in MASCOT amongst 20 parties, assuming a full-threshold access structure, the required communication is essentially $20 \times 19 = 380$ oblivious transfers (OTs). If we no longer assume full-threshold and instead suppose that any set of 15 parties contains an honest party, we need only $15 \times 14 = 210$ OTs plus $15 \times 5 = 75$ field elements to be sent per triple. In light of the real-world applications of MPC in which full-threshold is sometimes too strong an assumption, and the fact that the number of OTs required for a multiplication is $O(n^2)$, any reduction in the assumed fraction of corruptions t/n provides significant improvements in communication efficiency via our protocol, since we require only $O(t^2)$ OTs plus $O(t \cdot (n-t))$ field elements transmitted.

In the case where we use a probabilistic assignment, and where the parties generating the pre-processing are not later involved in the computation, our protocol is less efficient. For example, using our protocol and the probabilistic algorithm we describe later, if there are 5 parties in R of which at most 2 are corrupt, and 50 parties in Q of which at most 25 are corrupt (and R and Q are disjoint), each party in R need only send to 23 parties in Q for the cover to be statistically secure (in the sense that the adversary cannot with with probability greater than $1 - 2^{-80}$), instead of the 25 required for information-theoretic security.

Besides the ability of the protocol we describe to enable localising the generation of pre-processed data, another potential application of the protocol is to increase the number of parties involved in a given instance

of the SPDZ protocol dynamically (i.e. during the online phase). For example, suppose a set of parties already running an instance of the SPDZ protocol want to (efficiently and securely) allow another set of parties to join them during a reactive computation. It may make more sense to transform the already pre-processed data (or even just a few pre-processed values) via our protocol to a form that is amenable for use by a larger number of parties, and then distribute it to the parties who want to join in on the computation, instead of requiring that the parties halt the computation and then engage in a new round of pre-processing. This would only make sense if the parties joining the computation trusted at least one of the pre-existing parties, which is likely to be the case in any reasonable application of this use-case. The set of parties already performing the computation becomes the set R, and so we are in the first use-case above.

At its heart our technique can be described as follows. We let $\mathcal{F}_{\text{Prep}}^{\mathcal{P},\mathcal{A}}$ denote the SPDZ offline functionality for a set of parties $\mathcal{P}$ of size n with set of corrupt parties $\mathcal{A}$. Suppose now that we have a set of parties indexed by the set $[n]$ and (not necessarily disjoint) subsets $R, Q \subset [n]$ so that $R \cup Q = [n]$, and a subset $A \subset R \cup Q$ indexing corrupt parties. We then define a cover $\{Q_i\}_{i \in R}$ of Q such that there is at least one pair (i, Q_i) for which $i \in R$ is an honest party in R, and Q_i contains at least one honest party from Q. The cover provides a description of the required network: each party in Q_i must be connected by a *secure* channel to the associated party i in R. Just as (i, Q_i) associated a subset $Q_i \subset Q$ to a party $i \in R$, we also let $R_j \subset R$ be the set of parties in R associated to a party $j \in Q$. We then extend A to a set $\overline{A} \subset R \cup Q$ by setting $\overline{A} = A \cup \{j \in Q : R_j \subset A\}$. The set $\overline{A}$ contains all corrupt parties and additionally what we refer to as *effectively corrupt honest parties* with respect to the online phase of the protocol. In brief, these are parties whose pre-processed data is entirely determined by the adversary – while these parties execute the online protocol honestly, the deterministic dependence on pre-processed data means the adversary can decide what values these parties hold for their shares. Our protocol realises the functionality $\mathcal{F}_{\text{Prep}}^{Q,Q\cap\overline{A}}$ in the $\mathcal{F}_{\text{Prep}}^{R,R\cap\overline{A}}$-hybrid model.

The main idea of the protocol is conceptually quite simple, and is essentially a standard "re-sharing" technique similar to [BOGW88]. The main novelty is in showing that this can be efficiently applied to the SPDZ protocol, without the need for any expensive zero-knowledge proofs. In doing this, the difficulty comes in proving that the protocol is actually secure in the UC framework, and also in creating and analysing an (efficient) algorithm for assigning a cover to the network so that the adversary can only win with negligible probability in the security parameter in the case where we randomly assign the covers.

Related work. There is a long line of works on scalable secure computation with a large number of parties [DI06,HN06,DKMS14,BCP15] (to name a few), which use similar techniques to ours. These works often divide the parties into random *committees* (or *quorums*) to distribute the workload of the computation. Most of these papers target asymptotic efficiency, and strong models such as adaptive security, asynchronicity and RAM computation. This gives interesting theoretical results, but the practicality of these techniques has not been demonstrated. In contrast, our work focuses on applying simple techniques to to modern, practical MPC protocols. Furthermore, we give a concrete analysis and examples of parameters that can be used for different numbers of parties in real-world settings, at a given security level.

2 Preliminaries

In this section, we describe the notation used in subsequent sections, formally define *secure cover*, and give an overview of the SPDZ protocol, and the offline phase in particular.

General Concepts and Notation: Parties in the network are indexed by $[n] = \{1, ..., n\}$, where n is the total number of parties. We consider the complete network of parties as the union of two parts, which we call R and Q (so each is a subset of n and they are not necessarily disjoint). To avoid confusion, we will index parties in R by the letter i, and parties in Q by the letter j. We let n_r (resp. n_q) denote the number of parties in R (resp. Q). We let $A \subset R \cup Q$ denote the indexing set of corrupt parties in the complete network, and $\overline{A}$ denote the superset of A which possibly contains additional honest parties in Q, called *effectively corrupt*

honest parties from the introduction. We assume there is a complete network of authenticated channels amongst the parties in R, and similarly amongst the parties in Q. We define a *secure cover* $\{Q_i\}_{i\in R}$ of Q by R in the following way:

Definition 1. *Let $[n]$ be the indexing set of a set of parties in a given network and suppose we are also given subsets $R, Q \subset [n]$ of sizes n_r and n_q respectively. Each party in the network is either corrupt or honest. We call a set $\{Q_i\}_{i\in R}$ of (non-empty but not necessarily disjoint) subsets of Q a* secure cover *if the following hold:*

- *All parties in Q_i are connected to player $i \in R$ via a* secure *channel.*
- *The subsets cover Q, i.e. $Q = \bigcup_{i\in R} Q_i$.*
- *There is* at least one *pair (i, Q_i) where $i \in R$ is an honest party in R, and Q_i contains* at least one *honest party from Q.*

We will also let R_j denote the set of parties in R which are connected to party $j \in Q$. Note that $\{R_j\}_{j\in Q}$ is necessarily a cover of R since $Q_i \neq \varnothing$ for all i, so each i is in at least one R_j. We will use λ to denote the security parameter, and we will say an event occurs with overwhelming probability in the security parameter λ if it occurs with probability at least $1 - 2^{\lambda}$. We denote by $\mathbb{F}_q$ the finite field of order q, a (large) prime power. A function $\nu \in \mathbb{Z}[x]$ is called negligible if for every polynomial $p \in \mathbb{Z}[x]$, there exists a $C \in \mathbb{Z}$ such that $\nu(x) \leq 1/p(x)$ for all $x > C$. We write $\alpha \leftarrow \mathbb{F}_q$ to mean that α is sampled uniformly at random from the field $\mathbb{F}_q$. We denote by $\lceil a \rceil$ the smallest integer $b \in \mathbb{Z}$ such that $b \geq a$.

In Section 6, we discuss the different network topologies of secure channels between our parties in R and parties in Q. In particular, we explore the different ways by which to define the cover $\{Q_i\}_{i\in R}$, taking into account, for example, the fact that the Q_i's are not necessarily all the same size. Section 4 then builds on these considerations by providing concrete methods of creating the cover and analysing the resulting protocols. This involves, for example, examining how the likelihood of the cover being secure changes (if we define it probabilistically) as we change the value of ℓ if we require that all parties in R send to the same number ℓ of parties in Q.

In Appendix A, we give a brief overview of the Universally Composability (UC) framework, which is the model in which we give the proof of our main theorem. The power of UC is well demonstrated in the pre-processing model, since it allows the functionality to be split up into separate independent parts and their corresponding individual protocols to be proved secure independently, such that they remain secure even when run concurrently or sequentially. In this model, we define some functionality $\mathcal{F}_{\text{Prep}}$ for the pre-processing and a separate functionality $\mathcal{F}_{\text{Online}}$ for the online phase. A protocol is designed for each, Π_{Prep} and Π_{Online}, the protocol Π_{Prep} is shown to implement $\mathcal{F}_{\text{Prep}}$ securely, and finally Π_{Online} is shown to implement $\mathcal{F}_{\text{Online}}$ securely in the $\mathcal{F}_{\text{Prep}}$-hybrid model. This is particularly useful in our situation where we only want to change how pre-processing is done since we only need to revamp the pre-processing, and can leave the online phase unchanged, avoiding the need to reprove security.

SPDZ Overview: In general, computation will be done over a finite field $\mathbb{F} = \mathbb{F}_q$ where q is a (large) prime power. The protocol called MACCheck in the SPDZ paper [DPSZ12] requires that the field be large enough to make MAC forgery unfeasible by pure guessing. In particular this means that $1/q$ must be negligible in λ. For smaller finite fields, and in particular the important case of binary circuits, adaptions to the MACCheck protocol can be made; see [LOS14], for example. For this paper we will assume the simpler case of large q for ease of exposition. The SPDZ MPC protocol allows parties to compute an arithmetic circuit on their combined secret input. More specifically, for an arbitrary set of parties $\mathcal{P}$ and a subset set of corrupt parties $\mathcal{A} \subset \mathcal{P}$, the SPDZ protocol implements the functionality $\mathcal{F}_{\text{MPC}}^{\mathcal{P},\mathcal{A}}$ described in Figure 1, provided $\mathcal{P} \setminus \mathcal{A} \neq \varnothing$.

The main motivation for this paper is that the "standard" protocols which implement $\mathcal{F}_{\text{MPC}}^{\mathcal{P},\mathcal{A}}$ in the pre-processing model (for some set of parties $\mathcal{P}$ and corrupt parties $\mathcal{A}$) require a lot of work by the parties in $\mathcal{P}$ during pre-processing. Our goal is to implement $\mathcal{F}_{\text{MPC}}^{\mathcal{P},\mathcal{A}}$ using a (possibly larger) set of parties in which some specified set of parties execute the expensive pre-processing part of the protocol and only the parties in $\mathcal{P}$ who are interested in the computation itself execute the cheap online part of the protocol. In our terminology,

The Functionality $\mathcal{F}_{\text{MPC}}^{\mathcal{P},\mathcal{A}}$.

The superscript $\mathcal{P}$ denotes the set of parties involved in the protocol, and $\mathcal{A} \subsetneq \mathcal{P}$ is the set of corrupt parties.

Initialise: On input $(\mathsf{Initialise}, \mathbb{F})$ from all parties in $\mathcal{P}$, store $\mathbb{F}$.
Input: On input $(\mathsf{Input}, i, \mathsf{id}, x)$ from party i and $(\mathsf{Input}, i, \mathsf{id})$ from all other parties, with id a fresh identifier and $x \in \mathbb{F}$, store (id, x).
Add: On command $(\mathsf{Add}, \mathsf{id}_1, \mathsf{id}_2, \mathsf{id}_3)$ 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 $(\mathsf{id}_3, x + y)$.
Multiply: On command $(\mathsf{Multiply}, \mathsf{id}_1, \mathsf{id}_2, \mathsf{id}_3)$ 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 $(\mathsf{id}_3, x \cdot y)$.
Output: On input $(\mathsf{Output}, \mathsf{id})$ from all honest parties (where id is present in memory), retrieve (id, z), output z to the adversary. If the adversary responds with OK then output the value z to all parties, otherwise output Abort to all parties.

Figure 1. The Functionality $\mathcal{F}_{\text{MPC}}^{\mathcal{P},\mathcal{A}}$.

the parties in Q outsource the pre-processing to a set of parties R (which possibly includes some parties in R) and then compute using the data.

We will elaborate a little here; in what follows we use the notation and functionalities of the latest version of the SPDZ protocol, based on OT, called MASCOT [KOS16]. We will describe the SPDZ offline functionality $\mathcal{F}_{\text{Prep}}^{\mathcal{P},\mathcal{A}}$ and online protocol $\Pi_{\text{Online}}^{\mathcal{P},\mathcal{A}}$ for an arbitrary set of parties $\mathcal{P}$; at the end of this section, we give the conversion protocol $\Pi_{\text{Prep}}^{R \to Q, \overline{A}}$. In the initialisation stage, the parties sample (and keep private) random shares α_i, one for each party, whose sum is taken to be a global (secret) MAC key α, i.e. $\alpha = \sum_{i \in \mathcal{P}} \alpha_i$.

A value $x \in \mathbb{F}_q$ is secret shared among the parties in $\mathcal{P}$ by sampling $(x_i)_{i \in \mathcal{P}} \leftarrow \mathbb{F}_q^{|\mathcal{P}|}$ subject to $x = \sum_{i \in \mathcal{P}} x_i$, with party i holding the value x_i. In addition, we sample $(\gamma(x)_i)_{i \in \mathcal{P}} \leftarrow \mathbb{F}_q^{|\mathcal{P}|}$ subject to $\sum_{i \in \mathcal{P}} \gamma(x)_i = \alpha \cdot x$ and party i holding the share $\gamma(x)_i$. Thus $\gamma(x)_i$ is a sharing of the MAC $\gamma(x) := \alpha \cdot x$ of x. We write the following to denote that x is a secret value, where party $i \in \mathcal{P}$ holds x_i and $\gamma(x)_i$.

$$\langle x \rangle := ((x_i)_{i \in \mathcal{P}}, (\gamma(x)_i)_{i \in \mathcal{P}}))$$

Since this sharing scheme is linear, linear operations on secret values comes "for free", in the sense that adding secret values or multiplying them by a public constant requires no communication. Crucially, since the MAC is linear, the same operations applied to the corresponding MAC shares will result in MACs on the result of the said linear computation.

Unfortunately, multiplication of secret values requires a little more work, and is the reason data must be generated offline. At its heart SPDZ uses Beaver's method [Bea96] to multiply secret-shared values, which we outline here. In the offline phase, a large number of multiplication triples are generated, which are triples $(\langle a \rangle, \langle b \rangle, \langle c \rangle)$ such that $c = a \cdot b$. Note that while other forms of pre-processing can help in various computations, such as shared squares and shared bits, in this paper we focus on the basic form of pre-processing and leave the interested reader to consult [DKL+13] and [KSS13]. To multiply secret-shared elements $\langle x \rangle$ and $\langle y \rangle$ in the online phase, we take a triple $(\langle a \rangle, \langle b \rangle, \langle c \rangle)$ and partially open $\langle x \rangle - \langle a \rangle$ and $\langle x \rangle - \langle b \rangle$ to obtain $\varepsilon := \langle x \rangle - \langle a \rangle$ and $\delta := \langle y \rangle - \langle b \rangle$. By "partially open $\langle x \rangle - \langle a \rangle$", we mean that each party i sends the value $x_i - a_i$ to every other party, but does not send the corresponding MAC share. Then

$$\langle z \rangle = \langle c \rangle + \varepsilon \cdot \langle b \rangle + \delta \cdot \langle a \rangle + \varepsilon \cdot \delta$$

is a correct secret sharing of $z = x \cdot y$, and since the triple is never opened, no information about x or y is revealed. A similar use of pre-processed data is used for the parties to enter their inputs into the computation.

As remarked earlier our paper is focused on turning SPDZ preprocessing produced by one set of parties into preprocessing for another set of parties. Thus we do not discuss the online phase in detail. There is a minor tweak to the proof of security of the online phase, due to our minor tweak to the preprocessing functionality. For the interested reader we include the details in Section 5.

SPDZ Preprocessing: To formalise things a little more, we now discuss the functionality $\mathcal{F}_{\text{Prep}}^{\mathcal{P},\mathcal{A}}$, given in Figure 2, which implements the necessary pre-processing. The superscripts denote parameters of the functionality, where $\mathcal{P}$ denotes the indexing set of parties involved in the computation, and $\mathcal{A} \subset \mathcal{P}$ is a set of parties in $\mathcal{P}$ under the control of the adversary. As explained in the introduction, if we have a set of parties $\mathcal{P}$ and our cover produces *effectively corrupt honest parties*, which are those nominally honest parties which receive reshares from only corrupt parties, the set $\mathcal{A}$ will include these parties. If the parties generate the pre-processing themselves and do not make use of our protocol, this set is exactly the set of corrupt parties; when the pre-processing is outsourced, then we have to worry about (the possibility of) effectively corrupt parties.

The Offline Functionality $\mathcal{F}_{\text{Prep}}^{\mathcal{P},\mathcal{A}}$ for SPDZ.

The set $\mathcal{A} \subset \mathcal{P}$ indexes the corrupt parties in $\mathcal{P}$.

Initialise: On input $(\mathsf{Initialise}, q)$ from all players and the adversary, the functionality does the following:

1. The functionality samples $\alpha \leftarrow \mathbb{F}_q$ to be the global MAC key.
2. The functionality receives some error Δ_α from the adversary and, for each corrupted player $i \in \mathcal{A}$, a share α_i.
3. The functionality samples at random α_i for each $i \notin \mathcal{A}$ subject to $\sum_{i\in\mathcal{P}} \alpha_i = \alpha + \Delta_\alpha$.
4. The functionality sends α_i to party i, for all $i \in \mathcal{P}$.

Macro: $\mathsf{Angle}(x)$ The following will be run by the functionality at several points to create $\langle\cdot\rangle$ representations:

1. The functionality accepts $(\{x_i, \gamma(x)_i\}_{i\in\mathcal{A}}, \Delta_x, \Delta_\gamma)$ from the adversary.
2. The functionality samples at random $\{x_i, \gamma(x)_i\}_{i\notin\mathcal{A}}$ subject to $\sum_{i\in\mathcal{P}} x_i = x+\Delta_x$ and $\sum_{i\in\mathcal{P}} \gamma(x)_i = \alpha\cdot x+\Delta_\gamma$.
3. The functionality is left with $((x_i)_{i\in\mathcal{P}}, (\gamma(x)_i)_{i\in\mathcal{P}})$.

Computation: On input $(\mathsf{DataGen}, \mathit{DataType})$ from all players and the adversary, the functionality executes the data generation procedures specified below.

- On input $\mathit{DataType} = \mathsf{InputPrep}$ and a party $i \in \mathcal{P}$,
  1. If $i \notin \mathcal{A}$, the functionality samples $r^{(i)} \leftarrow \mathbb{F}_q$. Otherwise $i \in \mathcal{A}$ so the functionality accepts $r^{(i)}$ from the adversary.
  2. The functionality runs $\mathsf{Angle}(r^{(i)})$.
  3. For each $j \in \mathcal{P}$, the functionality sends party j the pair $(r_j^{(i)}, \gamma(r^{(i)})_j)$.
  4. Additionally, to party i, the functionality sends $r^{(i)}$.

  Thus the parties obtain a sharing $\langle r^{(i)}\rangle$ of a value $r^{(i)}$ known only to party i.
- On input $\mathit{DataType} = \mathsf{Triple}$,
  1. The functionality samples $a, b \in \mathbb{F}_p$ and computes $c = a \cdot b$.
  2. The functionality calls $\mathsf{Angle}(a)$, $\mathsf{Angle}(b)$ and $\mathsf{Angle}(c)$.
  3. For each $i \in \mathcal{P}$, the functionality sends $((a_i, \gamma(a)_i), (b_i, \gamma(b)_i), (c_i, \gamma(c)_i))$ to party i.

Figure 2. The Offline Functionality $\mathcal{F}_{\text{Prep}}^{\mathcal{P},\mathcal{A}}$ for SPDZ.

The functionality is a little more general than the functionality presented in [KOS16] as we allow the corrupt parties to introduce more errors: the standard SPDZ offline functionality only allows errors to be introduced into the MAC shares and not the data shares, whereas this new functionality allows errors on both. It is fairly intuitive that we will retain a security using this functionality as opposed to the standard one, as an adversary winning having changed shared values and MACs needs to have forged the same MAC equation as an adversary winning after just altering MAC values. The extra ability of altering share values gives him no advantage, a fact which we will prove shortly.

In [KOS16] the following theorem is (implicitly) proved, where $\mathcal{F}_{\text{OT}}$ and $\mathcal{F}_{\text{Rand}}$ are functionalities implementing OT and shared randomness for the parties.

Theorem 1. *There is a protocol $\Pi^{\mathcal{P},\mathcal{A}}_{Prep}$ that securely implements $\mathcal{F}^{\mathcal{P},\mathcal{A}}_{Prep}$ against static, active adversaries in the $\mathcal{F}_{OT}, \mathcal{F}_{Rand}$-hybrid model, where $\mathcal{P}$ is the complete set of parties and $\mathcal{A}$ the set of corrupt parties in $\mathcal{P}$.*

We do not give the definition of the $\Pi^{\mathcal{P},\mathcal{A}}_{\text{Prep}}$ protocol here as it is identical to MASCOT when based on OT, or identical to the original SPDZ pre-processing when based on homomorphic encryption (in spite of the slight difference in functionalities). Note that the paper [KOS16] proves the above theorem by giving a number of different protocols which, when combined, securely implement the required functionality $\mathcal{F}^{\mathcal{P},\mathcal{A}}_{\text{Prep}}$.

3 Feeding One Protocol From Another

In this section we give our main result on feeding pre-processed data from the parties in R to the parties in Q, assuming a set of corruptions in the latter which includes effectively corrupt honest parties. In notation, we are instantiating an instance of $\mathcal{F}^{Q,Q\cap\overline{A}}_{\text{Prep}}$ from an instance of $\mathcal{F}^{R,R\cap\overline{A}}_{\text{Prep}}$ via the protocol $\Pi^{R\to Q,\overline{A}}_{\text{Prep}}$. Note that $R\cap\overline{A} = R\cap A$. We assume we have a *secure cover* $\{Q_i\}_{i\in R}$ of Q. We emphasise that the actual execution of $\Pi^{R\to Q,\overline{A}}_{\text{Prep}}$ is independent of the set of corrupted parties; we only use the superscript $\overline{A}$ notation to indicate the relation between the corrupted parties in the protocol and in the different functionalities.

Method of Redistributing Data: Recall the parties in R will be performing the offline phase on behalf of the parties in Q. The parties in Q will share data in the standard manner (see Section 2), and the same will happen for parties in R. To avoid confusion, a data item $x \in \mathbb{F}$ secret shared amongst partes in Q will be denoted by $\langle x\rangle_Q$, whilst the same data item shared amongst parties in R will be denoted by $\langle x\rangle_R$, where implicitly we are assuming the *same* MAC key α is shared amongst the parties in R and the parties in Q.

When parties in Q want to evaluate a circuit amongst themselves, they follow the online protocol, in Figure 5, and whenever they require a pre-processed data-item, they will ask R to provide one[2]. Thus we simply require a methodology to translate $\langle x\rangle_R$ sharings into $\langle x\rangle_Q$ sharings. Recall a shared value in the network R is denoted by

$$\langle x\rangle_R = ((x_i)_{i\in R}, (\gamma(x)_i)_{i\in R})$$

The principal idea of the protocol is, for each $i \in R$, to take the value x_i held by i and sample a set $\{x^j_i\}_{j\in Q_i}$ subject to $x_i = \sum_{j\in Q_i} x^j_i$ and define $x^j \leftarrow \sum_{i\in R_j} x^j_i$ so that

$$\sum_{i\in R} x_i = \sum_{i\in R}\sum_{j\in Q_i} x^j_i = \sum_{j\in Q}\sum_{i\in R_j} x^j_i = \sum_{j\in Q} x^j$$

which holds because, by definition,

$$\{(i,j) : i \in R,\ j \in Q_i\} = \{(i,j) : j \in Q, i \in R_j\}.$$

If we do the same for the MAC shares, and at initialisation also share the global MAC key α in the same way, we obtain the same secret value x under the same global MAC key but shared instead amongst the parties in Q, which we denote by

$$\langle x\rangle_Q = ((x^j)_{j\in Q}, (\gamma(x)^j)_{j\in Q}).$$

It is hopefully now clear how to define a feeding protocol to send shares from the R parties to the Q parties. We do this by providing a protocol $\Pi^{R\to Q}_{\text{Prep}}$ which assumes the existence of the functionality $\mathcal{F}^{R,R\cap\overline{A}}_{\text{Prep}}$.

It is important to note that honest parties use incoming shares in an entirely deterministic manner; as such, observe that if some party $j \in Q$ is honest but it receives shares from only corrupt parties in R, the adversary has complete control over what this party's share will look like. For this reason, we consider them as "effectively" corrupt, contained in the extended adversary set $\overline{A}$. This is why in the online protocol, run by the parties in Q, we need to consider the set of adversaries as being $Q \cap \overline{A}$.

[2] Of course, Q could ask R for these to be obtained all in one go in a form of outsourced pre-processing.

The Protocol: The idea of the protocol is to convert the pre-processing generated by the parties in R to pre-processing that can be used by the parties in Q. Our goal, then, is to show that if the set of parties $R \cup Q$ is provided with the functionality $\mathcal{F}_{\text{Prep}}^{R,R\cap\overline{A}}$ and the parties engage in the protocol $\Pi_{\text{Prep}}^{R\to Q}$ to send their pre-processing to the parties in Q, then this "looks the same" to the parties in Q as a functionality $\mathcal{F}_{\text{Prep}}^{Q,Q\cap\overline{A}}$. The protocol is given in Figure 3.

Protocol $\Pi_{\text{Prep}}^{R\to Q,\overline{A}}$

Parties in R have ideal access to an instance of $\mathcal{F}_{\text{Prep}}^{R,R\cap\overline{A}}$.

Initialise: On input $(\mathsf{Initialise}, q)$ from all parties in Q,

1. The parties in R execute $\mathcal{F}_{\text{Prep}}^{R,R\cap\overline{A}}$.**Initialise**:
   (a) The functionality samples some $\alpha \leftarrow \mathbb{F}_q$.
   (b) From each corrupt party $i \in R \cap \overline{A}$, the functionality receives $(\alpha_i, \Delta_{\alpha,i})$.
   (c) The functionality samples $\{\alpha_i\}_{i\in R\setminus\overline{A}}$ uniformly subject to the constraint that $\sum_{i\in R}\alpha_i = \alpha + \sum_{i\in R\cap\overline{A}}\Delta_{\alpha,i}$.
   (d) For each $i \in R$, the functionality sends α_i to party i.
   Thus the parties in R obtain a sharing $\langle\alpha\rangle_R$ of the global MAC key.
2. The parties run $\mathsf{FeedValue}(\alpha)$ below to share the global MAC key amongst Q.

(Note that the **Macro** $\mathsf{Angle}(x)$ of $\mathcal{F}_{\text{Prep}}^{R,R\cap\overline{A}}$ is merely an internal procedure of the functionality and is not called by parties in the protocol.)

Macro: $\mathsf{FeedValue}(v)$ On input an element $v \in \mathbb{F}_q$ shared amongst the parties as $v = \sum_{i\in R} v_i$ with party $i \in R$ holding v_i,

1. For each $i \in R$, party i samples $\{v_i^j\}_{j\in Q_i}$ subject to $\sum_{j\in Q_i} v_i^j = v_i$.
2. For each $i \in R$, for each $j \in Q_i$, party i sends v_i^j to party j.
3. For each $j \in Q$, party j sets $v^j = \sum_{i\in R_j} v_i^j$.

Computation: On input $(\mathsf{DataGen}, \mathit{DataType})$ from all players in Q,

- On input $\mathit{DataType} = \mathsf{InputPrep}$ and a value $j \in Q$,
  1. For each $i \in R_j$,
     (a) The parties call $\mathcal{F}_{\text{Prep}}^{R,R\cap\overline{A}}$.**Computation**$(\mathsf{DataGen}, \mathsf{InputPrep})$ with input i:
        i. If $i \notin R \cap \overline{A}$, the functionality will sample some $r^{(i)} \leftarrow \mathbb{F}$, and otherwise will accept $r^{(i)}$ as input from the corrupt party.
        ii. In the execution of Angle, the corrupt parties $k \in R \cap \overline{A}$ each give the functionality some $r_k^{(i)}$ and $\gamma(r^{(i)})_k$ and errors $\Delta_{r^{(i)},k}$ and $\Delta_{\gamma(r^{(i)}),k}$. The functionality samples $\{r_k^{(i)}, \gamma(r_k^{(i)} : k \in R\setminus\overline{A}\}$ such that $\sum_{k\in R} r_k^{(i)} = r^{(i)} + \sum_{k\in R\setminus\overline{A}}\Delta_{r^{(i)},k}$ and $\sum_{k\in R}\gamma(r^{(i)})_k = \alpha\cdot r^{(i)} + \sum_{k\in R\setminus\overline{A}}\Delta_{\gamma(r^{(i)}),k}$.
        iii. For each $k \in R$, the functionality sends party k the pair $(r_k^{(i)}, \gamma(r^{(i)})_k)$.
        iv. Additionally, the functionality sends $r^{(i)}$ to party i.
        This provides the parties with $\langle r^{(i)}\rangle_R$ and party i with $r^{(i)}$.
     (b) The parties run $\mathsf{FeedValue}(r^{(i)})$ and $\mathsf{FeedValue}(\gamma(r^{(i)}))$ to get $\langle r^{(i)}\rangle_Q$.
     (c) Party i sends $r^{(i)}$ to party j.
  2. Party j computes $r_{(j)} \leftarrow \sum_{i\in R_j} r^{(i)}$.
  3. The parties in Q then fix $\langle r_{(j)}\rangle_Q \leftarrow \sum_{i\in R_j}\langle r^{(i)}\rangle_Q$.

  Thus all parties in Q obtain a sharing $\langle r_{(j)}\rangle_Q$ of a value $r_{(j)}$ known only to party $j \in Q$.
- On input $\mathit{DataType} = \mathsf{Triple}$ by parties in Q,
  1. The parties call $\mathcal{F}_{\text{Prep}}^{R,R\cap\overline{A}}$.**Computation**$(\mathsf{DataGen}, \mathsf{Triple})$ to obtain a triple $(\langle a\rangle_R, \langle b\rangle_R, \langle c\rangle_R)$.
  2. The parties in R now run $\mathsf{FeedValue}$ on a, b and c and their MACs.

Figure 3. Protocol $\Pi_{\text{Prep}}^{R\to Q,\overline{A}}$

Main theorem: Before we give the statement of the theorem, we briefly give some intuition as to why our construction gives us the desired security. Recall that we are given a cover $\{Q_i\}_{i\in R}$ of Q, indexed by parties in R, so that each party in R is associated to the set $Q_i \neq \varnothing$ of parties in Q. We defined a cover to be secure if there is at least one pair (i, Q_i) where i is honest and Q_i contains at least one honest party. If a cover is not secure, it means that for every i, we have that i is corrupt, or i is honest but Q_i contains no honest parties. In this case, given a secret value v, for each $i \in R$ the adversary either has share v_i (when $i \in R$ is corrupt), or all "reshares", $\{v_i^j\}_{j\in Q_i}$ (when $i \in R$ is honest but all $j \in Q_i$ are corrupt); using these shares and reshares, the adversary can construct v and hence he breaks secrecy: thus a secure cover is necessary. Conversely, if the cover is secure then at "worst", the adversary obtains all reshares but one; then since all reshares were sampled uniformly at random so that they summed to individual shares, and these shares were sampled so that they summed to the secret, this set of shares is indistinguishable from a uniformly randomly sampled set. Our main theorem (for which the proof is in Appendix B) is as follows.

Theorem 2. *The construction $\Pi_{Prep}^{R\to Q,\overline{A}}$ securely implements the functionality $\mathcal{F}_{Prep}^{Q,Q\cap\overline{A}}$ in the presence of static, active adversaries in the $\mathcal{F}_{Prep}^{R,R\cap\overline{A}}$-hybrid model assuming a secure cover of Q is given.*

4 Creating a Secure Cover

In the introduction, we assumed three potential use-cases. We now consider how to assign a cover securely for each scenario.

1. $R \subseteq Q$. In this case, for each i we define Q_i to be any subset of Q containing i and ensure that their union covers.
2. Each party in Q knows a subset of parties in R in which it believes there is an honest party. The cover is created respecting this knowledge.
3. There is no prior trust relationship.

In this last scenario we have two choices: either to set each covering subset Q_i equal to the whole set Q, or to assign the players randomly to subsets of Q whose union is the whole. In this section we provide an algorithm creating a cover and analyse the security it provides.

Recall that, when creating the secure cover, is necessary to ensure that at least one honest party in Q receives a share from at least one honest party in R with overwhelming probability in the security parameter λ. If this is not true, the adversary is able to reconstruct the share.

Let t_r and t_q be the number of corrupt parties in R and Q respectively. We set $\epsilon_r = t_r/n_r$ and $\epsilon_q = t_q/n_q$ to be the associated ratios. To help with the analysis, and for efficiency and load-balancing reasons, we will assume that each party in R sends to the same number of parties $\ell \geq \lceil n_q/n_r \rceil$ in Q. Note, any assignment of sets to parties in R which covers Q where $\ell = t_q+1$ is automatically secure, since every party in R necessarily sends to at least one honest party in Q. We will see how small ℓ can be to provide statistical security for a given security parameter.

To assign a cover randomly in such a situation we use the algorithm in Figure 4. The high-level idea of the algorithm is the following:

1. For each party in Q, we assign a random party in R, until each party in R has $\lceil n_q/n_r \rceil$ parties in Q assigned to it (or, equivalently, until the sets of parties in Q assigned to parties in R forms a disjoint cover). For ease of exposition, we assume $n_r | n_q$.
2. For each party in R, we assign random parties in Q until each party in R has ℓ total parties which it sends to.

Note that in practice, the parties may want to run this algorithm using a trusted source of randomness (such as a blockchain or lottery), or execute a coin-tossing protocol to generate the necessary randomness.

The algorithm allows different parties in Q to receive from different numbers of parties in R, whilst parties in R always send to the same number of parties in Q. Over $\mathbb{Z}$, each row of the matrix we generate, M, sums to ℓ, whilst the array NoOfOnes records how many parties in Q the i_k^{th} party in R sends to. Step 3 assigns all parties in Q to a party in R: this is the part of the algorithm which ensures we have a cover.

Algorithm for randomly assigning elements of a cover of Q to parties in R.

For ease of notation, we label parties in R as i_k for $k \in [n_r]$ and parties in Q as j_l for $l \in [n_q]$; then the output array M is a binary $n_r \times n_q$ matrix with a 1 in the $(k,l)^{\text{th}}$ position if and only if i_k in R sends to j_l in Q.
Inputs: n_r, n_q, $n = n_r + n_q$, ℓ, and sets $R, Q \subset [n]$ whose disjoint union is $[n]$.
Outputs: Matrix $M \in \mathbb{F}_2^{n_r \times n_q}$.
Method: (Note that ℓ is a constant, whereas l is an index.)

1. Set $M[1..n_r,\ 1..n_q] \leftarrow \{\{0,0,\ldots,0\},\ldots,\{0,0,\ldots,0\}\}$
2. Set NoOfOnes$[1..n_r] \leftarrow \{0,\ldots,0\}$
3. For $l \in [n_q]$,
   - Do
     - $k \leftarrow \mathcal{F}_{\text{Rand}}([n_r])$
   - Until NoOfOnes$[k] < \lceil n_q/n_r \rceil$ and $M[k,l] = 0$
   - $M[k,l] \leftarrow 1$, NoOfOnes$[k] \leftarrow$ NoOfOnes$[k] + 1$
4. For $k \in [n_r]$,
   (a) While NoOfOnes$[k] < \ell$,
   - Do
     - $l \leftarrow \mathcal{F}_{\text{Rand}}([n_q])$
   - Until $M[k,l] = 0$
   - $M[k,l] \leftarrow 1$, NoOfOnes$[k] \leftarrow$ NoOfOnes$[k] + 1$
5. Output matrix M.

Figure 4. Algorithm for randomly assigning elements of a cover of Q to parties in R.

In fact, this is done in such a way that each party in R sends to the *same* number of parties in Q, namely $\lceil n_q/n_r \rceil$. The reason for doing this is that it lends itself better to analysis of relevant probabilities below. Step 4 assign parties in Q to parties in R at random until each party in R is assigned ℓ parties in Q.

In the worst case, there is only one honest party in each of R and Q. Since we ensure that each party in R is assigned the same number of parties, the probability we obtain a secure cover is given by:

$$1 - \Pr[\text{Every good party in } R \text{ is assigned only dishonest parties in Step 3}] \cdot \Pr[\text{Every good party in } R \text{ is assigned only dishonest parties in Step 4}]$$

When performing Step 3, the probability that the first good party in R is assigned only dishonest parties is the number of ways of choosing $\lceil n_q/n_r \rceil$ parties from the t_q corrupt parties divided by the number of ways of choosing $\lceil n_q/n_r \rceil$ parties from all n_q parties:

$$\frac{\binom{t_q}{\lceil n_q/n_r \rceil}}{\binom{n_q}{\lceil n_q/n_r \rceil}}$$

Thus the first $\lceil n_q/n_r \rceil$ corrupt parties in Q have been assigned. Then the probability that the next honest party in R is also assigned only corrupt parties from the remaining $t_q - \lceil n_q/n_r \rceil$ corrupt parties, out of the $n_q - \lceil n_q/n_r \rceil$ remaining parties in Q, is:

$$\frac{\binom{t_q - \lceil n_q/n_r \rceil}{\lceil n_q/n_r \rceil}}{\binom{n_q - \lceil n_q/n_r \rceil}{\lceil n_q/n_r \rceil}}.$$

This continues until all the $n_r - t_r - 1$ honest parties in R have been assigned parties in Q.

Each party in R has been assigned $\lceil n_q/n_r \rceil$ parties in Q so that each party in Q has been assigned to exactly one party in R. In Step 4 of the algorithm, each party in R is randomly assigned parties in Q until all parties in R have ℓ parties assigned to them; they are thus each assigned $\ell - \lceil n_q/n_r \rceil$ more parties in Q. For a given party in R, this is the number of ways of choosing $\ell - \lceil n_q/n_r \rceil$ dishonest parties from the remaining $n_q - \lceil n_q/n_r \rceil$ parties in Q such that they too are all dishonest – i.e. they are from the $t_q - \lceil n_q/n_r \rceil$

remaining dishonest parties:

$$\frac{\binom{t_q-\lceil n_q/n_r\rceil}{\ell-\lceil n_q/n_r\rceil}}{\binom{n_q-\lceil n_q/n_r\rceil}{\ell-\lceil n_q/n_r\rceil}}$$

The choice of parties in Q is *with* replacement since the algorithm is oblivious to the choice of other parties in Step 4 (since Step 3 ensured the cover).

Then the probability that we obtain a secure cover is given by:

$$1-\left(\frac{\binom{t_q}{\lceil n_q/n_r\rceil}\cdot\binom{t_q-\lceil n_q/n_r\rceil}{\lceil n_q/n_r\rceil}\cdot\dots\cdot\binom{t_q-(n_r-t_r-1)\lceil n_q/n_r\rceil}{\lceil n_q/n_r\rceil}}{\binom{n_q}{\lceil n_q/n_r\rceil}\cdot\binom{n_q-\lceil n_q/n_r\rceil}{\lceil n_q/n_r\rceil}\cdot\dots\cdot\binom{n_q-(n_r-t_r-1)\lceil n_q/n_r\rceil}{\lceil n_q/n_r\rceil}}\right)\cdot\left(\frac{\binom{t_q-\lceil n_q/n_r\rceil}{\ell-\lceil n_q/n_r\rceil}}{\binom{n_q-\lceil n_q/n_r\rceil}{\ell-\lceil n_q/n_r\rceil}}\right)^{n_r-t_r}$$

After some simplification we find that this is equal to

$$1-\frac{t_q!\cdot(n_q-(n_r-t_r)\lceil n_q/n_r\rceil)!}{n_q!\cdot(t_q-(n_r-t_r)\lceil n_q/n_r\rceil)!}\cdot\left(\frac{\binom{t_q-\lceil n_q/n_r\rceil}{\ell-\lceil n_q/n_r\rceil}}{\binom{n_q-\lceil n_q/n_r\rceil}{\ell-\lceil n_q/n_r\rceil}}\right)^{n_r-t_r} \tag{1}$$

To see what happens in the extreme case where all but one party is corrupt in each of R and Q, we set $t_q = n_q - 1$ and $t_r = n_r - 1$. Then the probability that we obtain a secure cover is given by

$$\begin{aligned}1 \;-\; &\frac{(n_q-1)!\cdot(n_q-\lceil n_q/n_r\rceil)!}{(n_q)!\cdot(n_q-1-\lceil n_q/n_r\rceil)!}\cdot\frac{\binom{(n_q-1)-\lceil n_q/n_r\rceil}{\ell-\lceil n_q/n_r\rceil}}{\binom{n_q-\lceil n_q/n_r\rceil}{\ell-\lceil n_q/n_r\rceil}}\\ &= 1-\frac{n_q-\lceil n_q/n_r\rceil}{n_q}\cdot\frac{n_q-\ell}{n_q-\lceil n_q/n_r\rceil}=\frac{\ell}{n_q}.\end{aligned}$$

When ℓ is equal to n_q, i.e. each party in R sends to every party in Q, we obtain a secure cover. For any other choice of ℓ with this high proportion of corruptions, we do not obtain a sufficiently high probability of obtaining a secure cover. Thus our protocol will not be secure for any size of R.

When ℓ is at least $t_q + 1$, then every party in R necessarily sends to at least one honest party. For small numbers of parties, the parties in R must send to all parties in Q because chance of the cover being insecure is too great. However, as we increase the total number of parties, the probability that the cover is not secure decreases. For example, if there are 5 parties in R of which at most 2 are corrupt, and 50 parties in Q of which at most 25 are corrupt, each party in R must be assigned $\ell = 23$ parties in Q to ensure at least one honest party in R sends to one honest party in Q with probability at least $1 - 2^{-80}$, instead of the 25 parties we would require to *guarantee* the cover is secure.

For the data in Table 1, we fix the number of parties in R at 5, fix the number of allowable corruptions to be at most 3, and compute the lower bound on the size of Q (i.e. on n_q) to guarantee that the adversary cannot win even where ℓ is fixed as the smallest number of connections necessary to make $\{Q_i\}_{i\in R}$ to cover Q, and vary the number of corruptions we allow in Q. In other words, ℓ need be no larger to provide 80-bit security than it need be for enabling the partition to be an exact cover (i.e. each party in Q sent to by at most one party in R).

We stress that while the idea of completely outsourcing the pre-processing to an independent set of parties often does not result in an efficient protocol, the best use-case of our protocol is when $R \subset Q$; i.e., if there is some subset of parties trusted by all other parties in the network which can do all of the pre-processing and then distribute it to the other parties.

5 SPDZ Online Protocol

The SPDZ online protocol is given in Figure 5 which itself uses the subprocedure MACCheck presented in Figure 6, which itself makes use of a commitment functionality given in Figure 7. It has been shown that UC

n_r	t_r	t_q/n_q	Min. n_q for $\lambda = 80$ and $\ell = \lceil n_q/n_r \rceil$
5	3	1/2	336
5	3	1/3	201
5	3	1/4	148
5	3	1/5	125

Table 1. We fix $n_r = 5$, $t_r = 3$ and vary the fraction of corruptions in Q; the last column in the table is the least n_q such that the cover is secure even if each party in R only sends to $\ell = \lceil n_q/n_r \rceil$ parties.

commitment schemes in the plain model cannot exist, though they do exist in the *common reference string model* (in which one assumes the existence of common string known to all parties) [CF01], or, alternatively, the *random oracle model* (e.g. [HMQ04]).

The SPDZ Online Protocol $\Pi^{\mathcal{P},\mathcal{A}}_{\text{Online}}$.

The set $\mathcal{P}$ is the complete set of parties, and the set $\mathcal{A} \subset \mathcal{P}$ the set of corrupt parties in $\mathcal{P}$.

Initialise: The parties call $\mathcal{F}^{\mathcal{P},\mathcal{A}}_{\text{Prep}}$ for the handles of enough multiplication triples $(\langle a\rangle, \langle b\rangle, \langle c\rangle)$ and enough input mask values $(r_i, \langle r_i\rangle)$ as are needed for the function being evaluated. If $\mathcal{F}^{\mathcal{P},\mathcal{A}}_{\text{Prep}}$ aborts then the parties output $\perp$ and abort.
Input: To share an input x_i, party $i \in \mathcal{P}$ takes an available unused input mask value $(r_i, \langle r_i\rangle)$ and does the following:
1. Broadcast $\varepsilon \leftarrow x_i - r_i$.
2. The parties compute $\langle x_i\rangle \leftarrow \langle r_i\rangle + \varepsilon$.

Add: On input $(\langle x\rangle, \langle y\rangle)$, locally compute $\langle x + y\rangle \leftarrow \langle x\rangle + \langle y\rangle$.
Multiply: On input $(\langle x\rangle, \langle y\rangle)$, the parties do the following:
1. Take one multiplication triple $(\langle a\rangle, \langle b\rangle, \langle c\rangle)$, compute $\langle \varepsilon\rangle \leftarrow \langle x\rangle - \langle a\rangle$ and $\langle \rho\rangle \leftarrow \langle y\rangle - \langle b\rangle$ and partially open these shares to obtain ε and ρ respectively.
   Partially opening a sharing $\langle x\rangle$ consists of each party $i \in \mathcal{P}$ sending its share x_i to every other party $j \in \mathcal{P}$ and computing the sum of all of these shares, including the party's own. The values of $\gamma(x)_i$ are kept secret.
2. Set $\langle z\rangle \leftarrow \langle c\rangle + \varepsilon \cdot \langle b\rangle + \rho \cdot \langle a\rangle + \varepsilon \cdot \rho$.

Output: To output a share $\langle y\rangle$, do the following:
1. Check all partially opened values since the last MACCheck in the following manner.
   (a) The parties have some id's $\mathsf{id}_1, \ldots, \mathsf{id}_k$ for some k, and corresponding partially opened values $x_1, \ldots, x_k$.
   (b) The players agree on a random vector $\mathbf{r} \leftarrow \mathcal{F}_{\text{Rand}}(\mathbb{F}_q^k)$.
   (c) Party i in $\mathcal{P}$ computes $z \leftarrow \sum_{j=1}^k r_j \cdot x_j$ and $\gamma(z)_i \leftarrow \sum_{j=1}^k r_j \cdot \gamma(x_j)_i$ where $\gamma(x_j)_i$ denotes the MAC share held by party $i \in \mathcal{P}$ on x_j.
   (d) The parties now run MACCheck on z, with party i inputting z and $\gamma(z)_i$.
2. If the check fails, output $\perp$ and abort.
3. Open the value by each party $i \in \mathcal{P}$ sending y_i to all other parties $j \in \mathcal{P}$ to compute $y \leftarrow \sum_{j\in\mathcal{P}} y_i$, and then run MACCheck once more, so party $i \in \mathcal{P}$ inputs y and $\gamma(y)_i$, to verify $\langle y\rangle$. If this check fails, output $\perp$ and abort; otherwise, accept y as a valid output.

Figure 5. The SPDZ Online Protocol $\Pi^{\mathcal{P},\mathcal{A}}_{\text{Online}}$.

The MAC check passes if the MAC is correct for the corresponding share. Importantly, the check fails if the MAC is incorrect for the shared value, which occurs if the MAC *or* the value it authenticates (or both) is incorrect. Proofs can be found in [KOS16, App. B] and [DPSZ12, App. D3]. It is precisely because MACCheck detects errors in either the MAC value or share value or both that we can use an offline phase which introduces errors into the share values themselves, and not restrict ourselves to an offline phase in

The MACCheck Protocol from SPDZ/MASCOT.

On input an opened value s, a MAC share $\gamma(s)_i$ and a MAC key share α_i from each party i and a session id sid, each party i does the following:

1. Compute $\sigma_i \leftarrow \gamma(s)_i - s \cdot \alpha_i$ and call $\mathcal{F}_{\text{Commit}}.\mathsf{Commit}(\sigma_i, i, \mathsf{sid})$ to commit to this, and receive the handle τ_i.
2. When commitments are output by all parties call $\mathcal{F}_{\text{Commit}}.\mathsf{Open}(i, \mathsf{sid}, \tau_i)$ to open the commitments.
3. If $\sum_{i=1}^{n} \sigma_i \neq 0$, output $\perp$ and abort; otherwise, continue.

Figure 6. The MACCheck Protocol from SPDZ/MASCOT.

Commitment Functionality $\mathcal{F}_{\text{Commit}}$.

Commit: On input $\mathsf{Commit}(v, i, \mathsf{sid})$ by party i, where v is the value to committed, sample a handle τ_v and send $(i, \mathsf{sid}, \tau_v)$ to all parties.
Open: On input $\mathsf{Open}(i, \mathsf{sid}, \tau_v)$ by party i, output $(v, i, \mathsf{sid}, \tau_v)$ to all parties. If some party P_i is corrupt and the adversary inputs $(\mathsf{Abort}, i, \mathsf{sid}, \tau_v)$, the functionality outputs $(\perp, i, \mathsf{sid}, \tau_v)$ to all parties.

Figure 7. Commitment Functionality $\mathcal{F}_{\text{Commit}}$.

which only errors on MACs are allowed (as in the original SPDZ papers). For clarity, we show this more explicitly in the proof of the next theorem:

Theorem 3. *The protocol $\Pi^{\mathcal{P},\mathcal{A}}_{Online}$ securely implements the functionality $\mathcal{F}^{\mathcal{P},\mathcal{A}}_{MPC}$ in the $\mathcal{F}^{\mathcal{P},\mathcal{A}}_{Prep}$, $\mathcal{F}_{Commit}$, $\mathcal{F}_{Rand}$-hybrid model.*

Proof. The proof is identical to that in [DPSZ12], except that the pre-processing may now introduce errors into the share values as well as the MAC values. To prove the theorem, we must show that no environment can distinguish between an adversary interacting as in the protocol $\Pi^{\mathcal{P},\mathcal{A}}_{\text{Prep}}$ and a simulator interacting with the functionality $\mathcal{F}^{\mathcal{P},\mathcal{A}}_{\text{Prep}}$. Thus the proof runs exactly as in [DPSZ12, App. D3], except that when we run the MACCheck protocol, the error can now be on the value in the share or the MAC. However, the security game presented in [DPSZ12] already allowed the adversary to introduce errors on the shares, so the original protocol already offers the stronger guarantee that no error can occur on either the MAC or the value of the share it authenticated (or both). Note that if the adversary can alter the share and the MAC and have MACCheck pass, then in particular this is equivalent to tweaking some share a_i by t to obtain $a'_i \leftarrow a_i + t$ and then choosing the correct tweak τ on the MAC share, $\gamma(a')_i \leftarrow \gamma(a_i) + \tau$ so that the check passes. However, this means the adversary has successfully guessed $\alpha = \tau/t$, the global MAC key, which it can only do successfully with probability $1/q$. (Since q is expontial in the security parameter λ, this probability is negligible.) □

6 Communication between subnetworks

Here we discuss the topology of the network of secure channels between the subnetworks R and Q. Recall that Q has been partitioned into sets $\{Q_i\}_{i \in R}$ and party $i \in R$ assigned the set Q_i, and each Q_i is assumed to be of size ℓ. The topology depends primarily on the choice for the size ℓ of each set Q_i. We assume Q_i is the same size for all i, and note that obviously ℓ is lower-bounded by $\lceil n_q/n_r \rceil$ (so that $\ell \cdot n_r \geq n_q$), since $\{Q_i\}_{i \in R}$ together need to cover Q.

6.1 Complete

A naïve approach to connecting the two graphs with bilateral secure channels would be to form the complete bipartite graph between them (so $\ell = n_q$). This topology requires $n_r \cdot n_q$ secure connections and is shown in Figure 8. If there is at least one honest party in each of R and Q then an adversary controlling any number of other parties still can never recover the MAC key. Unfortunately, there is a big communication overhead.

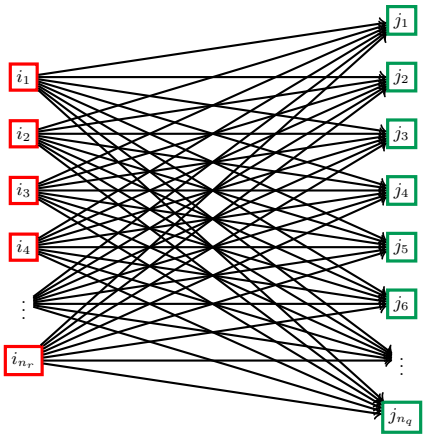

Fig. 8. Complete bipartite graph

Additionally, each party in R must compute n_q reshares for their share. If we assume the adversary is able to corrupt at most some t of the n total parties, we can clearly improve efficiency by instead requiring each party in R to send to $t+1$ parties in Q, since then it is guaranteed each party, and in particular at least one honest party, sends to an honest party in Q.

6.2 Load-balanced

To aim for a load-balanced solution, we could instead ask each party $i \in R$ to reshare its share into $\ell = k \cdot \lceil n_q/n_r \rceil$ shares for some integer $k \geq 1$, and sending these to some set Q_i of ℓ parties in Q. If we have a secure cover, then the intuition is that there exist shares held by only honest players which are independent of all shares held by the adversary and are necessary for reconstructing the secret. This is discussed in more detail in the proof of our main theorem (see Appendix B). Figure 9 shows an example of our load-balanced topology for when $n_q \approx 2n_r$ and $k = 1$. Note that it is not necessarily the case that each party in Q receive the same number of shares, even though we require each party in R to reshare to the same number of parties in Q.

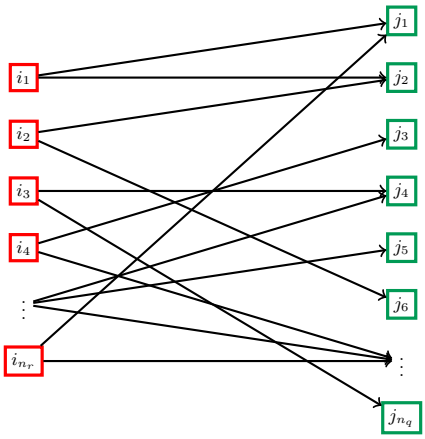

Fig. 9. Load-balanced topology

Acknowledgements

This work has been supported in part by ERC Advanced Grant ERC-2015-AdG-IMPaCT, by EPSRC via grant EP/N021940/1, by Defense Advanced Research Projects Agency (DARPA) and Space and Naval Warfare Systems Center, Pacific (SSC Pacific) under contract No. N66001-15-C-4070, and by the European Union's Horizon 2020 programme under grant No. 731583 (SODA).

References

BCP15. Elette Boyle, Kai-Min Chung, and Rafael Pass. Large-scale secure computation: Multi-party computation for (parallel) RAM programs. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 742–762. Springer, Heidelberg, August 2015.

BDOZ11. Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 169–188. Springer, Heidelberg, May 2011.

Bea96. Donald Beaver. Correlated pseudorandomness and the complexity of private computations. In *28th ACM STOC*, pages 479–488. ACM Press, May 1996.

BLN+15. Sai Sheshank Burra, Enrique Larraia, Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, Emmanuela Orsini, Peter Scholl, and Nigel P. Smart. High performance multi-party computation for binary circuits based on oblivious transfer. Cryptology ePrint Archive, Report 2015/472, 2015. `http://eprint.iacr.org/2015/472`.

BOGW88. Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *20th ACM STOC*, pages 1–10. ACM Press, May 1988.

Can00. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067, 2000. `http://eprint.iacr.org/2000/067`.

Can01. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

CF01. Ran Canetti and Marc Fischlin. Universally composable commitments. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 19–40. Springer, Heidelberg, August 2001.

DGKN09. Ivan Damgård, Martin Geisler, Mikkel Krøigaard, and Jesper Buus Nielsen. Asynchronous multiparty computation: Theory and implementation. In Stanislaw Jarecki and Gene Tsudik, editors, *PKC 2009*, volume 5443 of *LNCS*, pages 160–179. Springer, Heidelberg, March 2009.

DI06. Ivan Damgård and Yuval Ishai. Scalable secure multiparty computation. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 501–520. Springer, Heidelberg, August 2006.

DKL+13. Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. Practical covertly secure MPC for dishonest majority - or: Breaking the SPDZ limits. In Jason Crampton, Sushil Jajodia, and Keith Mayes, editors, *ESORICS 2013*, volume 8134 of *LNCS*, pages 1–18. Springer, Heidelberg, September 2013.

DKMS14. Varsha Dani, Valerie King, Mahnush Movahedi, and Jared Saia. Quorums quicken queries: Efficient asynchronous secure multiparty computation. In *ICDCN*, volume 8314 of *Lecture Notes in Computer Science*, pages 242–256. Springer, 2014.

DPSZ12. Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 643–662. Springer, Heidelberg, August 2012.

HMQ04. Dennis Hofheinz and Jörn Müller-Quade. Universally composable commitments using random oracles. In Moni Naor, editor, *TCC 2004*, volume 2951 of *LNCS*, pages 58–76. Springer, Heidelberg, February 2004.

HN06. Martin Hirt and Jesper Buus Nielsen. Robust multiparty computation with linear communication complexity. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 463–482. Springer, Heidelberg, August 2006.

KOS16. Marcel Keller, Emmanuela Orsini, and Peter Scholl. MASCOT: Faster malicious arithmetic secure computation with oblivious transfer. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 16*, pages 830–842. ACM Press, October 2016.

KSS13. Marcel Keller, Peter Scholl, and Nigel P. Smart. An architecture for practical actively secure MPC with dishonest majority. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 13*, pages 549–560. ACM Press, November 2013.

LOS14. Enrique Larraia, Emmanuela Orsini, and Nigel P. Smart. Dishonest majority multi-party computation for binary circuits. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 495–512. Springer, Heidelberg, August 2014.

NNOB12. 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, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 681–700. Springer, Heidelberg, August 2012.

RBO89. Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In *21st ACM STOC*, pages 73–85. ACM Press, May 1989.

A UC Model Overview

In this section, we give a brief overview of the UC framework, the model in which we proof our main theorem. Readers familiar with the model can skip this section.

Our proof is in the Universal Composability (UC) framework introduced by Canetti [Can00,Can01]. The model was introduced to enable protocols to be "composed", meaning that multiple different protocols (or multiple instances of the same protocol) can be run simultaneously such that the overall system is still "secure". There are some protocols which can offer no security if we permit multiple simulataneous executions. This framework works well in the pre-processing model in MPC, as we split the circuit evaluation into two phases, the offline phase and the online phase; the overall protocol is the composition of these phases.

In this model, we compare executions in an ideal world with executions in the real world. In the real world, there is a set of honest parties who communicate in a protocol with a real-world adversary $\mathcal{A}$. In the ideal world, there is a set of honest parties and an ideal-world adversary called the simulator $\mathcal{S}$, but now these entities interact not with each other, but some trusted third party called a functionality $\mathcal{F}$, which takes inputs from all parties, computes on these data, and provides the output to all parties including the simulator.

The protocol between all parties in the real world somehow needs to resemble the interaction the parties have with the functionality in the ideal world. More formally, the real-world view of an environment, which provides inputs to all parties, sees all internal actions of the real-world adversary, and sees the outputs of all parties, needs to be indistinguishable from the ideal-world view.

To achieve this in "practice", the adversary engages in the protocol with the *simulator*, which extracts the adversary's inputs and forwards them on to the functionality, which then interacts with honest parties. If the protocol is designed correctly, the indistinguishability of the two views of the environment guarantees that the real-world adversary has no more power than the ideal-world adversary. In the design of the functionality, we limit the power of the adversary according to our security model: for example, if we want to allow the adversary to cause the interaction to abort without output, we design the functionality to allow it to accept an abort flag from the adversary and to halt when it receives it. The reason we make the functionality "weaker" than it could be (i.e. why we allow the adversary to have any control in the interaction at all) is because it may be proven that no protocol exists which can stop an adversary doing some particular malicious behaviour. Since the simulator does not have any control over the honest parties and yet is supposed to simulate an execution of the protocol with the real-world adversary, the simulator has to do what it can to provide a view (i.e. messages from the alleged honest parties to the corrupt parties) close to what honest parties would actually send. In practice, this means it internally does what each honest party would do in the protocol and passes this on to the adversary.

The environment models any behaviour of the system in which the protocol is run. Proving the indistinguishability of the views shows there is no efficient attack strategy for breaking the protocol by running other protocols alongside it. If the environment cannot distinguish between the real- and ideal-world executions, the protocol is therefore secure even when run alongside any number of other protocols (as long as those are also proven secure in this model), or even arbitrarily many instances of the same protocol.

B Proof of Theorem 2

Proof. The proof is presented via a simulator (see Figure 10 and Figure 11) whose task is to overlay the functionality $\mathcal{F}_{\text{Prep}}^{Q,Q\cap\overline{A}}$ so that no environment can distinguish whether the adversary is interacting as in $\Pi_{\text{Prep}}^{R\to Q,\overline{A}}$, or with the simulator and $\mathcal{F}_{\text{Prep}}^{Q,Q\cap\overline{A}}$. Figure 12 shows an outline of what each entity does during the simulation. Since we are in the $\mathcal{F}_{\text{Prep}}^{R,R\cap\overline{A}}$-hybrid model, the simulator must reply to all calls the adversary makes to this functionality.

Simulator $\mathcal{S}_{\text{Prep}}^{Q,Q\cap\overline{A}}$

The simulator begins by first setting the set of corrupt parties to be the corrupt parties in R with the corrupt parties in Q, augmented to include all parties in Q which receive from only corrupt parties in R.

Initialise: The simulator receives a message $(\mathsf{Initialise}, q)$ from the adversary and forwards the message to the functionality $\mathcal{F}_{\text{Prep}}^{Q,Q\cap\overline{A}}$. Then the simulator does the following:

1. The simulator simulates $\mathcal{F}_{\text{Prep}}^{R,R\cap\overline{A}}$.**Initialise** with the same input $(\mathsf{Initialise}, q)$:
   - For each corrupt party $i \in R$, the simulator receives an error $\Delta_{\alpha,i}$ and share α_i. The simulator sets $\Delta_\alpha^R \leftarrow \sum_{i\in R\cap\overline{A}} \Delta_{\alpha,i}$.
   - The simulator sends $(\alpha_i)_{i\in R\cap\overline{A}}$ back to the adversary.
2. The simulator receives the call $\mathsf{FeedValue}(\alpha)$ from the adversary, and so the simulator samples a set $\{\alpha_i : i \in R \setminus \overline{A}\} \leftarrow \mathbb{F}$ and they perform the subroutine $\mathsf{SFeedValue}$. This gives the simulator a set of shares $\{\alpha^j\}_{j\in Q}$ and an error Δ_α^Q. The simulator sends $\{\alpha^j : j \in Q\cap\overline{A}\}$ along with the error Δ_α^Q to the functionality $\mathcal{F}_{\text{Prep}}^{Q,Q\cap\overline{A}}$.

Macro: $\mathsf{SFeedValue}$ On input a secret-shared value v (which could be a shared value or its shared MAC, or the global MAC key):

1. The simulator retrieves the shares v_i for all $i \in R$ and also the error given by the adversary, Δ_v^R.
2. For each $i \in R$, the simulator does the following:
   (a) If i is honest, the simulator samples $\{\hat{v}_i^j\}_{j\in Q_i}$ subject to $\sum_{j\in Q_i} \hat{v}_i^j = v_i$ and sends $\{\hat{v}_i^j\}_{j\in Q_i\cap\overline{A}}$ to the adversary. The simulator sets $\Delta_{v,i} \leftarrow 0$.
   (b) If i is corrupt and only sends to honest parties, the simulator receives a set $\{\tilde{v}_i^j\}_{j\in Q_i}$ and determines the error introduced by the adversary by setting $\Delta_{v,i} \leftarrow \left(\sum_{j\in Q_i} \tilde{v}_i^j\right) - v_i$.
   (c) If i is corrupt and sends to at least one corrupt party, the simulator receives a (possibly empty) set $\{\tilde{v}_i^j\}_{j\in Q_i\setminus\overline{A}}$ of reshares of the corrupt party to honest parties. The simulator samples reshares $\{\bar{v}_i^j\}_{j\in Q_i\cap\overline{A}}$ subject to $\sum_{j\in Q_i\cap\overline{A}} \bar{v}_i^j + \sum_{j\in Q_i\setminus\overline{A}} \tilde{v}_i^j = v_i$, and sets $\Delta_{v,i} \leftarrow 0$.
3. Now for each $j \in Q \cap \overline{A}$, the simulator computes $v^j = \sum_{i\in R_j\cap\overline{A}} \bar{v}_i^j + \sum_{i\in R_j\setminus\overline{A}} \hat{v}_i^j$ and for each $j \in Q \setminus \overline{A}$, the simulator computes $v^j = \sum_{i\in R_j\cap\overline{A}} \tilde{v}_i^j + \sum_{i\in R_j\setminus\overline{A}} \hat{v}_i^j$. The simulator also computes the new error $\Delta_v^Q \leftarrow \Delta_v^R + \sum_{i\in R} \Delta_{v,i}$.

Figure 10. Simulator $\mathcal{S}_{\text{Prep}}^{Q,Q\cap\overline{A}}$

To do this we must show that the view of the environment in each case is the same. The view of the environment consists of the joint distribution of: the inputs and outputs of all parties (honest and corrupt), the adversary's internal state, and all messages the adversary sent and received.

SFeedValue: Since it is used at several points throughout the functionality, we start by showing that the view of the environment when the adversary interacts in the protocol $\Pi_{\text{Prep}}^{R\to Q,\overline{A}}$ with the simulator is indistinguishable from its view when the adversary interacts with $\mathcal{F}_{\text{Prep}}^{Q,Q\cap\overline{A}}$. By the time the macro $\mathsf{FeedValue}$ is

Simulator $\mathcal{S}_{\text{Prep}}^{Q,Q\cap\overline{A}}$ cont'd

Computation: On input $(\mathsf{DataGen}, DataType)$ from the adversary, the simulator executes the data generation procedures as follows:

- On input $DataType = \mathsf{InputPrep}$ and a value $j \in Q$ from the adversary,
  1. For each $i \in R_j$,
     (a) The adversary calls $\mathcal{F}_{\text{Prep}}^{R,R\cap\overline{A}}$.**Computation**$(\mathsf{DataGen}, \mathsf{InputPrep})$ with input i so the simulator does the following:
        i. If $i \in R \cap \overline{A}$, the adversary sends the simulator a value $r^{(i)}$.
        ii. In the execution of $\mathcal{F}_{\text{Prep}}^{R,R\cap\overline{A}}$.$\mathsf{Angle}$, the adversary sends
        $$(r_k^{(i)}, \gamma(r^{(i)})_k)_{k\in R\cap\overline{A}} \text{ and errors } (\Delta_{r^{(i)},k}, \Delta_{\gamma(r^{(i)}),k})_{k\in R\cap\overline{A}}$$
        to the simulator. The simulator sets
        $$\Delta_{r^{(i)}}^R \leftarrow \sum_{k\in R\cap\overline{A}} \Delta_{r^{(i)},k} \text{ and } \Delta_{\gamma(r^{(i)})}^R \leftarrow \sum_{k\in R\cap\overline{A}} \Delta_{\gamma(r^{(i)}),k}$$
        iii. The simulator sends $(r_k^{(i)}, \gamma(r^{(i)})_k)_{k\in R\cap\overline{A}}$ back to the adversary.
        iv. If $i \in R \cap \overline{A}$, the simulator sends $r^{(i)}$ to the adversary.
     (b) The simulator receives the calls $\mathsf{FeedValue}(r^{(i)})$ and $\mathsf{FeedValue}(\gamma(r^{(i)}))$ from the adversary, so the simulator samples a set $\{r_k^{(i)}, \gamma(r^{(i)})_k : k \in R \setminus \overline{A}\} \leftarrow \mathbb{F}$ and they perform the subroutine $\mathsf{SFeedValue}$ on each. This gives the simulator a set of shares $\{r^{(i),k}, \gamma(r^{(i)})^k : k \in Q\}$ and errors $\Delta_{r^{(i)}}^Q$ and $\Delta_{\gamma(r^{(i)})}^Q$.
     (c) If $i \in R \setminus \overline{A}$ and $j \in Q \cap \overline{A}$, the simulator sends $r^{(i)}$ to the adversary. If $i \in R \cap \overline{A}$ and $j \in Q \setminus \overline{A}$, the simulator waits for the adversary to send $\tilde{r}^{(i)}$ and modifies the error on $r^{(i)}$ by setting $\Delta_{r^{(i)}}^Q \leftarrow \Delta_{r^{(i)}}^Q + r^{(i)} - \tilde{r}^{(i)}$
  2. The simulator computes $r_{(j)} \leftarrow \sum_{i\in R_j\setminus\overline{A}} r^{(i)} + \sum_{i\in R_j\cap\overline{A}} \tilde{r}^{(i)}$ and then sends the signal $(\mathsf{DataGen}, \mathsf{Input})$ to the functionality $\mathcal{F}_{\text{Prep}}^{Q,Q\cap\overline{A}}$. The functionality awaits input from the simulator:
     - If $j \in Q \cap \overline{A}$ the simulator sends $r_{(j)}$ to the functionality $\mathcal{F}_{\text{Prep}}^{Q,Q\cap\overline{A}}$.
     - The simulator sends the set $\{r^{(i),k}, \gamma(r^{(i)})^k : k \in Q \cap \overline{A}\}$ to the functionality $\mathcal{F}_{\text{Prep}}^{Q,Q\cap\overline{A}}$ along with the errors $\Delta_{r^{(i)}}^Q$ and $\Delta_{\gamma(r^{(i)})}^Q$.
- On input $DataType = \mathsf{Triples}$,
  1. The simulator responds to the adversary's request to execute $\mathcal{F}_{\text{Prep}}^{R,R\cap\overline{A}}$.**Computation**$(\mathsf{DataGen}, \mathsf{Triple})$ by doing the following:
     (a) In the execution of $\mathcal{F}_{\text{Prep}}^{R,R\cap\overline{A}}$.$\mathsf{Angle}$, the adversary sends $\{a_i, \gamma(a)_i, b_i, \gamma(b)_i, c_i, \gamma(c)_i : i \in R \cap \overline{A}\}$ and errors Δ_a, $\Delta_{\gamma(a)}$, Δ_b, $\Delta_{\gamma(b)}$, Δ_c and $\Delta_{\gamma(a)}$ to the simulator.
     (b) The simulator executes the procedure honestly so that it obtains a set $\{a_i, \gamma(a)_i, b_i, \gamma(b)_i, c_i, \gamma(c)_i : i \in R\}$ and sends the set $\{a_i, \gamma(a)_i, b_i, \gamma(b)_i, c_i, \gamma(c)_i : i \in R \cap \overline{A}\}$ back to the adversary.
  2. The simulator now runs $\mathsf{SFeedValue}$ on a, b and c and their MACs with the adversary. This gives the simulator a set of shares $\{a^j, \gamma(a)^j, b^j, \gamma(b)^j, c^j, \gamma(c)^j : j \in Q\}$ and errors Δ_a^Q, $\Delta_{\gamma(a)}^Q$, Δ_b^Q, $\Delta_{\gamma(b)}^Q$, Δ_c^Q and $\Delta_{\gamma(a)}^Q$.
  3. The simulator sends input $(\mathsf{DataGen}, \mathsf{Triple})$ to the functionality $\mathcal{F}_{\text{Prep}}^{Q,Q\cap\overline{A}}$ and then sends it the set $\{a^j, \gamma(a)^j, b^j, \gamma(b)^j, c^j, \gamma(c)^j : j \in Q \cap \overline{A}\}$ and the errors above.

Figure 11. Simulator $\mathcal{S}_{\text{Prep}}^{Q,Q\cap\overline{A}}$ cont'd

called, the simulator has a set $\{v_i\}_{i\in R}$ and the simulator and adversary have agreed on an error Δ_v^R. With each share, the simulator behaves differently depending on the secure cover:

1. If a party i in R is honest, the simulator simply reshares the share v_i held for that party by sampling a set $\{\hat{v}_i^j\}_{j\in Q_i}$ such that $v_i = \sum_{j\in Q_i} \hat{v}_i^j$ and sets $\Delta_{v,i} \leftarrow 0$.

2. If a party i in R is corrupt, then the simulator receives a (possibly empty) set of shares $\{\tilde{v}_i^j\}_{j\in Q_i\setminus\overline{A}}$ from the adversary. Then:
   (a) If the party in R sends only to honest parties, the simulator computes $\Delta_{v,i} \leftarrow \left(\sum_{j\in Q_i} \tilde{v}_i^j\right) - v_i$.
   (b) If the party in R sends to at least one corrupt party, the simulator samples the remaining shares $\{\bar{v}_i^j\}_{j\in Q_i\cap\overline{A}}$ so that $\sum_{j\in Q_i\setminus\overline{A}} \tilde{v}_i^j + \sum_{j\in Q_i\cap\overline{A}} \bar{v}_i^j = v_i$, and sets $\Delta_{v,i} \leftarrow 0$.

The simulator sets $\Delta_v^{\text{Reshare}} \leftarrow \sum_{i\in R} \Delta_{v,i}$. Thus the only time the simulator considers that the adversary has contributed an error is when all shares of a corrupt party are sent to honest parties. The intuition is that if a corrupt party in Q, who receives from at least one corrupt party in R, introduces an error and sends it to the functionality, this is equivalent to the corrupt sender in R sending a different reshare to the corrupt party in Q. This means that no error is "committed" to unless the adversary sends all reshares of a given share to the honest parties. For each $j \in Q\cap\overline{A}$, the simulator fixes $\hat{w}^j \leftarrow \sum_{i\in R_j\setminus\overline{A}} \hat{v}_i^j + \sum_{i\in R_j\cap\overline{A}} \bar{v}_i^j$. Note that since the simulator did not receive the reshares of corrupt parties in R to corrupt parties in Q, this $\hat{w}^j$ is defined using $\bar{v}_i^j$'s not $\tilde{v}_i^j$'s. This is where the simulation $\mathsf{SFeedValue}$ ends.

We will now see formally why the computed error and shares which are sent to $\mathcal{F}_{\text{Prep}}^{Q,Q\cap\overline{A}}$ produce shares for honest parties which are consistent with what the environment expects to see:

- During the call to $\mathcal{F}_{\text{Prep}}^{Q,Q\cap\overline{A}}.\mathsf{Angle}$, the simulator computes the error $\Delta_v^Q \leftarrow \Delta_v^R + \Delta_v^{\text{Reshare}}$ and sends it along with the set $\{\hat{w}^j : j \in Q\cap\overline{A}\}$ to the functionality.
- The functionality will sample $w \leftarrow \mathbb{F}$ and a set $\{w^j\}_{j\in Q\setminus\overline{A}}$ so that $\sum_{j\in Q\cap\overline{A}} \hat{w}^j + \sum_{j\in Q\setminus\overline{A}} w^j = w + \Delta_v^Q$. The functionality will send these shares to honest parties.
- If the adversary were to follow the protocol, it would compute $\tilde{w}^j \leftarrow \sum_{i\in R_j\setminus\overline{A}} \hat{v}_i^j + \sum_{i\in R_j\cap\overline{A}} \tilde{v}_i^j$ for each $j \in Q\cap\overline{A}$.

Putting this together, we obtain

$$
\begin{aligned}
\sum_{j\in Q\cap\overline{A}} \tilde{w}^j + \sum_{j\in Q\setminus\overline{A}} w^j &= \sum_{j\in Q\cap\overline{A}} \tilde{w}^j + w + \Delta_v^Q - \sum_{j\in Q\cap\overline{A}} \hat{w}^j \\
&= \sum_{j\in Q\cap\overline{A}} \left(\sum_{i\in R_j\setminus\overline{A}} \hat{v}_i^j + \sum_{i\in R_j\cap\overline{A}} \tilde{v}_i^j \right) + w + \Delta_v^R + \Delta_v^{\text{Reshare}} - \sum_{j\in Q\cap\overline{A}} \left(\sum_{i\in R_j\setminus\overline{A}} \hat{v}_i^j + \sum_{i\in R_j\cap\overline{A}} \bar{v}_i^j \right) \\
&= \sum_{j\in Q\cap\overline{A}} \sum_{i\in R_j\cap\overline{A}} \tilde{v}_i^j - \sum_{j\in Q\cap\overline{A}} \sum_{i\in R_j\cap\overline{A}} \bar{v}_i^j + w + \Delta_v^R + \Delta_v^{\text{Reshare}} \\
&= \sum_{i\in R\cap\overline{A}} \sum_{j\in Q_i\cap\overline{A}} \tilde{v}_i^j - \sum_{i\in R\cap\overline{A}} \sum_{j\in Q_i\cap\overline{A}} \bar{v}_i^j + w + \Delta_v^R + \Delta_v^{\text{Reshare}} \\
&= \sum_{i\in R\cap\overline{A}} \sum_{j\in Q_i\cap\overline{A}} \tilde{v}_i^j - \sum_{\substack{i\in R\cap\overline{A}: \\ Q_i\cap\overline{A}\neq\varnothing}} \sum_{j\in Q_i\cap\overline{A}} \bar{v}_i^j + w + \Delta_v^R + \Delta_v^{\text{Reshare}} \\
&= \sum_{i\in R\cap\overline{A}} \sum_{j\in Q_i\cap\overline{A}} \tilde{v}_i^j - \sum_{\substack{i\in R\cap\overline{A}: \\ Q_i\cap\overline{A}\neq\varnothing}} \left(v_i - \sum_{j\in Q_i\setminus\overline{A}} \tilde{v}_i^j \right) + w + \Delta_v^R + \sum_{\substack{i\in R\cap\overline{A}: \\ Q_i\cap\overline{A}=\varnothing}} \left(\left(\sum_{j\in Q_i\setminus\overline{A}} \tilde{v}_i^j \right) - v_i \right) \\
&= \sum_{i\in R\cap\overline{A}} \sum_{j\in Q_i\cap\overline{A}} \tilde{v}_i^j - \sum_{\substack{i\in R\cap\overline{A}: \\ Q_i\cap\overline{A}\neq\varnothing}} v_i + \sum_{\substack{i\in R\cap\overline{A}: \\ Q_i\cap\overline{A}\neq\varnothing}} \sum_{j\in Q_i\setminus\overline{A}} \tilde{v}_i^j + w + \Delta_v^R + \sum_{\substack{i\in R\cap\overline{A}: \\ Q_i\cap\overline{A}=\varnothing}} \left(\left(\sum_{j\in Q_i\setminus\overline{A}} \tilde{v}_i^j \right) - v_i \right) \\
&= \sum_{i\in R\cap\overline{A}} \sum_{j\in Q_i\cap\overline{A}} \tilde{v}_i^j - \sum_{i\in R\cap\overline{A}} v_i + \sum_{i\in R\cap\overline{A}} \sum_{j\in Q_i\setminus\overline{A}} \tilde{v}_i^j + w + \Delta_v^R \\
&= \sum_{i\in R\cap\overline{A}} \tilde{v}_i - \sum_{i\in R\cap\overline{A}} v_i + w + \Delta_v^R.
\end{aligned}
$$

This shows that the shares sampled by the functionality for honest parties during this procedure, namely the set $\{w^j\}_{j \in Q \setminus \overline{A}}$, sum with the shares the adversary computes (or would compute) to differ from the secret by precisely the error the adversary introduced on the shares when resharing (the first two summands above) added to the error introduced by the adversary on generating the original secret (the last summand above). For uniformly sampled values in the protocol like $r^{(i)}$, the actual error will not be observed by the environment since the secret is chosen uniformly at random; however, during triple generation, it is necessary that these errors be consistent in Q as in R, since the environment can reconstruct the secrets (from the adversary's shares and the shares honest parties output at the end) and check if the product deviates in accordance with the errors introduced.

The fact that the cover is secure also means that during any execution of $\mathcal{F}_{\text{Prep}}^{R,R\cap\overline{A}}.\mathsf{Angle}$, it suffices for the simulator to sample uniformly at random any honest parties' shares which are to be reshared. This is because the security of the cover ensures the adversary never sees all reshares of all shares (and thus cannot discover the secret). Since the environment is also oblivious to the internal state of honest parties, its lack of the one reshare from honest to honest denies it the ability to reconstruct the "intermediate" secrets which are merely simulated and are therefore different (with high probability) from the final secrets sampled by the functionality $\mathcal{F}_{\text{Prep}}^{Q,Q\cap\overline{A}}$.

Initialise: First, the adversary sends the simulator a set of shares and errors for the global MAC key, and the simulator sends the shares back. Then, to "transfer" the global MAC key from R to Q, the simulator samples $\{\alpha_i : i \in R \setminus \overline{A}\}$ and then the simulator and adversary run $\mathsf{SFeedValue}$. As noted above, the security of the cover means the environment cannot reconstruct the global MAC key shared amongst the parties in R; importantly, this means the environment cannot see that the (implicit) simulated MAC key differs from the MAC key sampled by $\mathcal{F}_{\text{Prep}}^{R,R\cap\overline{A}}$ (with high probability). Note that it is necessary that the simulator defer its call to the functionality $\mathcal{F}_{\text{Prep}}^{Q,Q\cap\overline{A}}$ until after the call to $\mathsf{SFeedValue}$ because the adversary may introduce errors during resharing in addition to those which it specified during the call to $\mathcal{F}_{\text{Prep}}^{R,R\cap\overline{A}}$.

Computation: During this procedure, the simulator acts exactly as $\mathcal{F}_{\text{Prep}}^{R,R\cap\overline{A}}$ would whenever the adversary sends commands for this functionality, so the simulation runs exactly as in the real world. The only other communication is when the simulator and adversary engage in $\mathsf{SFeedValue}$, for which we have already shown that the simulator provides a view for the environment that is indistinguishable between the ideal and real worlds. It was noted above that while for some secrets the error actually need not be the same for Q as for R because they are supposed to be uniformly random elements anyway, when producing triples the errors must be consistent between the two networks since the randomness is correlated. However, this is not a problem since the simulator can explicitly compute and "carry through" errors from parties in R to parties in Q, so the errors on triples are consistent in each network: even though the values for a, b and c are not known to the simulator, the environment will observe the same errors on each in the simulation as in the real world.

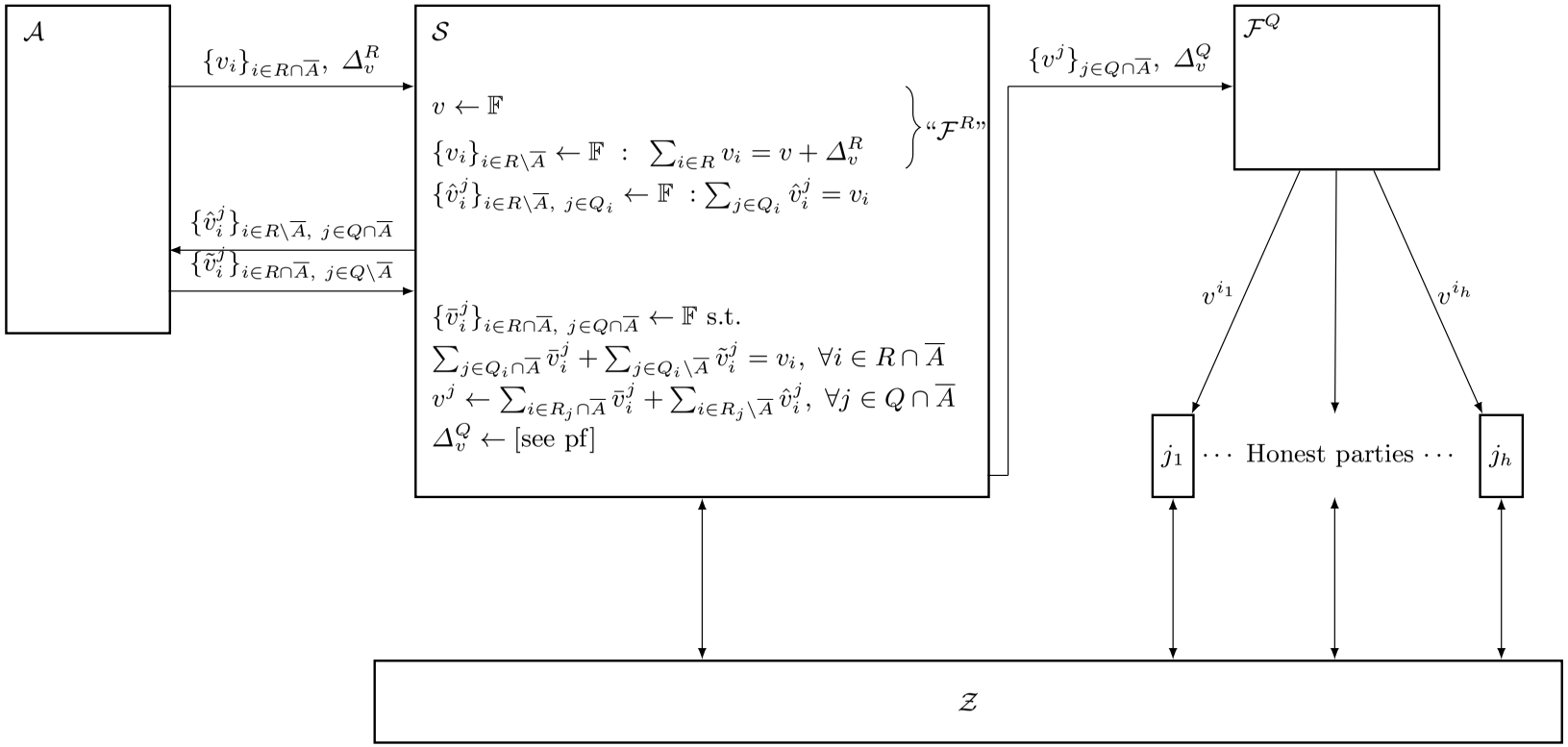

Fig. 12. Overview of the ideal-world simulation SFeedValue.