Asynchronous MPC with $t < n/2$ Using Non-equivocation

Michael Backes* Fabian Bendun† Ashish Choudhury‡ Aniket Kate§

Abstract

Secure Multiparty Computation (MPC) is a fundamental problem in distributed cryptography. Although MPC in the synchronous communication setting has received tremendous attention in security research, recent interest in deploying MPC in real-life systems requires going beyond the synchronous setting and working towards MPC in the weaker asynchronous communication setting. The asynchronous setting, however, does not come without a penalty: asynchronous MPC (AMPC) protocols among n parties can only tolerate up to $t < n/3$ active corruptions in contrast to the synchronous protocols, which can tolerate up to $t < n/2$ corruptions.

In this work, we improve the resiliency bound for AMPC using *non-equivocation*. Non-equivocation is a mechanism to restrict a corrupted party from making conflicting statements to different (honest) parties, and it can be implemented using an increment-only counter, realizable with trusted hardware modules readily available in commodity computers and smartphone devices. In particular, using non-equivocation, we present an AMPC protocol in the asynchronous setting, tolerating $t < n/2$ faults. From a practical point of view, our AMPC protocol requires fewer setup assumptions than the previous AMPC protocol with $t < n/2$ by Beerliová-Trubíniová, Hirt and Nielsen (PODC '10): unlike their AMPC protocol, it does not require any synchronous broadcast round at the beginning of the protocol and avoids the threshold homomorphic encryption setup assumption. Moreover, our AMPC protocol is also efficient and provides a gain of $\Theta(n)$ in the communication complexity per multiplication gate, over the AMPC protocol of Beerliová-Trubíniová et al. In the process, using non-equivocation, we also define the first asynchronous verifiable secret sharing (AVSS) scheme with $t < n/2$, which is of independent interest to threshold cryptographic protocols.

*Department of Computer Science, Saarland University, and MPI-SWS, Email: backes@cs.uni-saarland.de.

†Department of Computer Science, Saarland University, Email: bendun@cs.uni-saarland.de.

‡Department of Computer Science, University of Bristol, Email: ashish.choudhary@bristol.ac.uk.

§MMCI, Saarland University, Email: aniket@mmci.uni-saarland.de.

1 Introduction

Threshold secure multi-party computation (MPC) is one of the most important primitives in distributed cryptographic systems. Informally, in a system of n mutually distrusting parties, an MPC protocol allows the n parties to "securely" evaluate any agreed-on function f of their private inputs, in the presence of a centralized active adversary $\mathcal{A}$, controlling at most t out of the n parties. In the synchronous communication model, where the message transfer delays are bounded by a known constant, the MPC problem has been studied extensively (e.g., [4,6,10,12,18,24,29,40,47]). In the real-world settings, however, message delays cannot always be bounded accurately without being extremely pessimistic. Thus there is growing interest in generalizing MPC to an asynchronous communication model [9,14,17] that does not place any bound on the communication delays. The weaker restrictions on the adversary in the asynchronous model not only worsen the required resiliency conditions and communication complexities, but also make designing distributed protocols a more challenging task; intuitively this is because in a completely asynchronous setting, it is not possible to distinguish between a slow (but honest) sender (whose messages are arbitrarily delayed) and a corrupted sender (who does not send any message at all). Due to this, at any "stage" of an asynchronous protocol, no party can afford to wait to hear from all the parties (as this may turn out to be endless) and so the communication from t (potentially honest) parties may be ignored [17]. Due to their complexity, only a few asynchronous MPC (AMPC) protocols are available in the literature [5, 9, 11, 20, 31, 32, 38, 42].

In this work, we focus on an asynchronous model with a *computationally bounded* adversary, where the parties are connected by pairwise authenticated links. In this setting, it is well known that AMPC protocols are possible if and only if $t < n/3$ [31, 32]. This is in contrast to the synchronous world, where we can tolerate upto $t < n/2$ corruptions [30]. Motivated by the problem of bridging the gap between the resilience of synchronous and asynchronous MPC protocols, Beerliová-Trubíniová, Hirt and Nielsen [7] showed that it is possible to design an AMPC protocol tolerating $t < n/2$ corruptions in a "partial" synchronous network. More specifically, assuming one *synchronous broadcast* round at the beginning of the protocol, where each party can synchronously broadcast to every other party, they designed an AMPC protocol tolerating $t < n/2$ corruptions. Due to the availability of the synchronous broadcast round, their protocol could also ensure "input provision", i.e. the inputs of all the (honest) parties are considered for the computation, which otherwise is impossible to achieve in an asynchronous protocol [17]. Nevertheless, their requirement of one synchronous broadcast round per MPC instance may not always be realizable in practice. Specifically, if we try to implement the synchronous broadcast round over the pairwise channels using a computationally secure broadcast protocol, then it would require $\Theta(t)$ synchronous rounds of communication for a deterministic broadcast protocol [26] or $\mathcal{O}(1)$ (with a large constant) expected synchronous rounds of communication for a randomized broadcast protocol [27,44]. It was left as an open problem in [7] to see whether one can design an AMPC protocol with $t < n/2$ under other simplified assumptions.

In distributed computing research, a similar problem with asynchronous protocols has recently been addressed by introducing a minimal trusted hardware assumption [21–23, 33, 34, 36]. In particular, it was shown that, the resilience of asynchronous distributed computing tasks such as reliable broadcast, Byzantine agreement, and state machine replication (SMR) can be improved using a *minimal trusted hardware module* at each party. The hardware module utilized is just a trusted, increment-only local counter, which can be realized with trusted platform module (TPM) chips [45] available in almost all computers, or ARM TrustZone modules [46] available in smartphones and a majority of smartcard devices. Using such trusted hardware with each party, one can design asynchronous reliable broadcast tolerating up to $t < n$ active faults [23], and asynchronous Byzantine agreement (ABA) and SMR protocols tolerating up to $t < n/2$ [21, 34, 36] active faults, all of which otherwise require $t < n/3$ [44].

At a conceptual level, such a trusted module makes it impossible for a corrupted party to perform *equivocation*, which essentially means making conflicting statements to different (honest) parties. Clement et al. [22] generalized the results from [21,23,36] and proved that *non-equivocation* (i.e., making equivocation

impossible) along with digital signatures allow treating active (or Byzantine) faults as crash failure for many distributed computing primitives. In particular, they present a generic transformation that enables any crash-fault tolerant distributed protocol to tolerate the same number of Byzantine faults using non-equivocation and signatures. Nevertheless, their generic transformation considers only the basic distributed computing requirements of safety and liveness. It does not apply to cryptographic tasks such as AMPC where *confidentiality (or privacy)* of inputs is also required. This presents an interesting challenge to assess the utility of non-equivocation for the secure distributed computing task of AMPC.

1.1 Our Contributions and Related Work

In this work, we study the power of non-equivocation in the context of AMPC. We demonstrate that using a non-equivocation mechanism, one can improve the resiliency bound of AMPC from $t < n/3$ to $t < n/2$ without requiring any synchrony assumption. In particular, we present a general MPC protocol in a completely asynchronous communication model with $n \geq 2t + 1$. Our AMPC protocol, called NeqAMPC , improves upon the previous AMPC protocol [7] with $n \geq 2t + 1$ in the following two ways:

(a) Simplified assumptions. The NeqAMPC protocol needs a non-equivocation mechanism, but unlike [7] neither makes any synchronous broadcast round assumption nor requires a threshold homomorphic encryption setup. Given the feasibility of realizing non-equivocation over a majority of computing devices, we argue that on the Internet non-equivocation is a more practical assumption than the synchronous broadcast round assumption.

(b) Efficiency. For a security parameter κ, our AMPC protocol requires an amortized communication complexity of $\mathcal{O}(n^3\kappa)$ bits per multiplication gate, which improves upon the AMPC protocol of [7] by a factor of $\Theta(n)$.

To reduce the setup assumptions for the NeqAMPC protocol, we avoid the traditional threshold additive homomorphic encryption based circuit evaluation approach as used in [7,31,32]. Instead, we employ a secret sharing-based circuit evaluation approach [10,18,40], where confidentiality of the computation is maintained via secret sharing. Nevertheless, as detailed in our protocol overview in the next section, secret-sharing based AMPC with $n = 2t + 1$ and $\mathcal{O}(n^3\kappa)$ communication complexity (per multiplication) presents several challenges and as a result the NeqAMPC protocol is significantly different than those in the literature [5, 7, 31, 32].

In the process, we also present the first computationally secure asynchronous verifiable secret sharing (AVSS) [1,2,14,17] scheme with $n \geq 2t+1$ (using non-equivocation), which otherwise requires $t < n/3$ [2]. Moreover, our AVSS has communication complexity $\mathcal{O}(n^2\kappa)$, which improves upon the previous best AVSS scheme of [2] by a factor of $\Theta(n)$. Our AVSS scheme has an additional useful feature—it is the first publicly verifiable [43] AVSS scheme, as it allows any third party to publicly verify the "consistency" of the shares. With its efficiency and public verifiability, our AVSS scheme may be of independent interest to other cryptographic protocols.

Comparison with Existing Work. The best known computationally secure AMPC protocols are reported in [7, 32]. The protocol in [32] considers a *fully* asynchronous setting with $t < n/3$, whereas [7] assumes one synchronous broadcast round and can tolerate up to $t < n/2$ corruptions. Both the protocols require a threshold (additive) homomorphic encryption instantiation, and incur an (amortized) communication complexity of $\mathcal{O}(n^2\kappa)$ and $\mathcal{O}(n^4\kappa)$ bits per multiplication gate respectively.[1]

We do not employ a threshold encryption scheme, but rather prefer a more standard public key encryption setup with the addition of a non-equivocation mechanism. Our AMPC protocol with $t < n/2$ performs

[1] Beerliová-Trubíniová et al. [7] focused on designing a protocol with $t < n/2$, and the communication complexity of $\mathcal{O}(n^4\kappa)$ of their protocol (measured by us in Appendix A) can possibly be improved.

circuit evaluation by secret-sharing the inputs and incurs a communication complexity of $\mathcal{O}(n^3\kappa)$ bits per multiplication gate. Nevertheless, we observe that by modifying our protocol and employing a threshold encryption scheme (coupled with a non-equivocation mechanism), we can tolerate $t < n/2$ faults with communication complexity $\mathcal{O}(n^2\kappa)$ bits per multiplication gate. However, we prefer the secret-sharing based AMPC, as we aim to reduce the assumptions relied upon.

We note that unlike [7], our AMPC protocol could not enforce input provision: the input from t potentially honest parties may be ignored for computation. As discussed earlier, this is inherent to asynchronous protocols and presents a trade-off between our protocol and that of [7] based on which is more important: input provision or getting rid of the synchrony assumption. Notice that, using a non-equivocation mechanism, one can realize asynchronous reliable broadcast (see Section 3.3) with $t < n/2$ and consequently get rid of the synchronous broadcast round required in [7]. Nevertheless, the resultant protocol will still require the threshold homomorphic encryption setup and $\mathcal{O}(n^4\kappa)$ communication complexity, and it will no longer support input provision.

2 Overview of Our AMPC Protocol

Without loss of generality, we assume that $n = 2t + 1$ and so $t = \Theta(n)$. We assume that the function f to be computed is expressed as an arithmetic circuit over the field $\mathbb{Z}_p$, where $p > n$ is a κ bit prime and κ is the security parameter. The circuit consists of two input addition (linear) and multiplication (non-linear) gates, apart from random gates. Our AMPC protocol consists of two phases: an input phase and a computation phase. During the input phase, the parties share their inputs, while during the computation phase, the parties jointly evaluate f on the shared inputs and publicly reconstruct the output. Linear gates can be evaluated locally if the underlying secret-sharing scheme is linear; thus, we use the polynomial-based (Shamir) secret-sharing scheme with threshold t [41]. We denote a sharing of a value s by $[s]$. It follows that locally adding the shares of $[x]$ and $[y]$ provides the shares for $[x + y]$.

Multiplication gates cannot be evaluated locally since multiplying the individual shares results in the underlying sharing polynomial having degree $2t$. Therefore we evaluate multiplication gates using the standard Beaver's circuit randomization technique [3]. This technique requires three "pre-processed" secret-shared values, say $([u], [v], [w])$, unknown to $\mathcal{A}$, such that $w = u \cdot v$. Given such a shared *multiplication triple*, and shared inputs of a multiplication gate, say $[x]$ and $[y]$, the multiplication gate is securely evaluated as follows: to have secret $x \cdot y$ shared, the parties use the equation $[x \cdot y] = (x - u) \cdot (y - v) + [v] \cdot (x - u) + [u] \cdot (y - v) + [u \cdot v]$. They compute the sharing of $(x - u)$ and $(y - v)$, and publicly reconstruct the same. Once $(x - u)$ and $(y - v)$ are public, the parties can compute their shares of $x \cdot y$, using the above equation and employing the linearity property of the secret sharing. As u and v are random and unknown, the knowledge of $(x - u)$ and $(y - v)$ does not violate the privacy of x and y.

2.1 Pre-processing Phase

Although the idea of our AMPC protocol is the same as the existing information theoretically secure MPC and AMPC protocols [5, 6, 20, 25], our major challenge lies in generating the required shared multiplication triples with $n = 2t + 1$ parties; all the above protocols employ at least $n > 3t$ parties for this purpose[2]. These triplets are independent of the circuit and the inputs of the parties, and generated in an additional *pre-processing* phase. Generating these triplets efficiently is the important problem we solve in our protocol, and the protocol gains a factor of $\Theta(n)$ over the previous AMPC protocol with $n = 2t + 1$ [7]. In the rest of the section, we give an overview of how $(c_M + c_R)$ shared random triples are generated, where c_M and c_R

[2]Shared multiplication triples with $n = 2t+1$ have been generated in the synchronous setting [4,12]; however, their adaptability to our asynchronous setting is unclear.

are the number of multiplication gates and random gates in the circuit. In order to do so, we first describe how a single triple is generated (see Figure 1 for a pictorial representation of the protocols involved) and then extend this to $c_M + c_R$ triples.

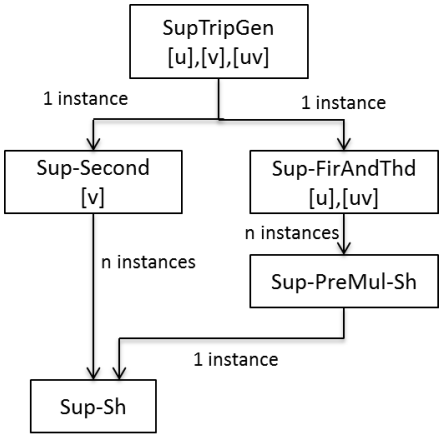

Figure 1: **Multiplication Triple Generation under Supervision of a P_{king}**

Supervised Triple Generation (Section 5). The idea for generating a random shared multiplication triple $[u], [v], [w]$ is to compute a random sharing $[v]$ and then combining "several" $[u^i]$s and $[u^i \cdot v]$s to get $[u]$ and $[w]$. The triple generation protocol uses two sub-protocols: Sup-Sh and Sup-PreMul-Sh. Protocol Sup-Sh allows a *dealer* D to compute the sharing $[u]$ of his value u, while the Sup-PreMul-Sh protocol allows a dealer D to compute a sharing $[u]$ and $[u \cdot v]$, given u and $[v]$. We can use Sup-Sh and Sup-PreMul-Sh in the following way to generate $([u], [v], [w])$: first, we ask each party P_i to act as a dealer D and invoke an instance of Sup-Sh to share a uniformly random value, say $v^{(i)}$. The parties then agree on a common subset (say $\mathcal{T}_v$) of $n - t = t + 1$ dealers whose Sup-Sh instances will eventually be terminated by all the parties. We set $v = \sum_{P_i \in \mathcal{T}_v} v^{(i)}$. The shared value v will be random and unknown to $\mathcal{A}$, as $\mathcal{T}_v$ has at least one honest P_i. Next, each party P_i is asked to act as a D and invoke an instance of Sup-PreMul-Sh to share a uniformly random value $u^{(i)}$ as well as $u^{(i)} \cdot v$. The parties then agree on a common subset of $n - t = t + 1$ dealers, say $\mathcal{T}_u$, whose instance of Sup-PreMul-Sh will eventually be terminated by all the parties. For $u = \sum_{P_i \in \mathcal{T}_u} u^{(i)}$ and $w = \sum_{P_i \in \mathcal{T}_u} u^{(i)} \cdot v$, the triple (u, v, w) is a random multiplication triple.

There are, however, some important subtleties. The Sup-PreMul-Sh protocol actually needs more than the dealer D knowing that there exists a value v shared among the parties, as a *precondition*; it expects D to have *encryptions* of all n shares of v, encrypted under the individual keys of the respective share-holders, where the encryption scheme is additively homomorphic (and not threshold additively homomorphic). We call a party having these encrypted shares to be *privileged*. However, due to asynchronicity, the Sup-Sh protocol cannot guarantee that *all* the n parties are privileged with respect to each individual $v^{(i)}$ (corresponding to each $P_i \in \mathcal{T}_v$). We solve this problem by ensuring that there exists a designated (possibly corrupted) *supervisor* P_{king} called *king*. The Sup-Sh protocol ensures that the king is a privileged party with respect to *each* $v^{(i)}$ (corresponding to each $P_i \in \mathcal{T}_v$). An honest P_{king} can then compute all the n encrypted shares of v using the homomorphic properties of the encryption scheme, and can reliably broadcast the encrypted shares of v. The required reliable broadcast protocol is possible for $n > t$ using the non-equivocation mechanism. Once P_{king} (correctly) broadcasts the n encrypted shares of v, then each P_i can invoke its instance of Sup-PreMul-Sh as a D.

The resultant wrapper protocols are called Sup-Second and Sup-FirAndThd, where Sup-Second generates $[v]$ under the supervision of P_{king} and Sup-FirAndThd generates $[u]$ and $[w = u \cdot v]$ under the supervision of P_{king}. A combination of Sup-Second and Sup-FirAndThd leads to the protocol SupTripGen

under the supervision of a designated P_{king}, which outputs a uniformly random and private multiplication triple $([u], [v], [w])$.

Preprocessing Phase Protocol (Section 6). Protocol SupTripGen may not terminate for a *corrupted* P_{king}. So we ask each party P_i to act as a king and generate shared random multiplication triples under its supervision by invoking an instance of SupTripGen. As the instances of honest kings will eventually terminate, we distribute the load of generating $c_M + c_R$ shared random multiplication triples among n parties. So each party P_i is asked to act as a king and generate $\frac{c_M+c_R}{t+1}$ shared multiplication triples in its instance of SupTripGen. The parties then agree on a common subset $\mathcal{TCORE}$ of $t+1$ kings whose instances of SupTripGen will eventually be completed by everyone and the $|\mathcal{TCORE}| \cdot \frac{c_M+c_R}{t+1} = c_M + c_R$ shared triples obtained in these instances are considered as the final output.

2.2 Important Sub-protocols for the Preprocessing Phase

We now discuss the realization of the main sub-protocols Sup-Sh and Sup-PreMul-Sh for the preprocessing phase.

Protocol Sup-Sh (Section 4). Our Sup-Sh protocol is almost equivalent to the AVSS primitive [2, 14, 17]: it allows a *dealer* D to "verifiably" share a secret s, thus generating $[s]$, and ensures that at least *one* honest party is privileged to obtain all the n shares encrypted for the respective share holders. The existing computationally secure AVSS protocols [2, 14] are designed with $n = 3t + 1$ and are based on sharing a secret using a bivariate polynomial of degree t in each variable and (homomorphic) commitments. In this paradigm, it is ensured that D has distributed "consistent" shares to $n - t = 2t + 1$ parties such that (at least) $n - 2t = t + 1$ *honest* parties among them can "enable" the remaining parties to get their shares. Unfortunately, this approach cannot be used with $n = 2t + 1$, as with $n = 2t + 1$ we can only ensure that D has distributed consistent shares to $n - t = t + 1$ parties. In the worst case, there will be *only one* honest party in this set, who does not have sufficient information to help the remaining t parties to complete a sharing of a bivariate polynomial of degree t.

We solve this problem by introducing encryptions of the shares.[3] Moreover, instead of using bivariate polynomials, we employ univariate polynomials. Now each party is given a vector of n encrypted shares as well as homomorphic commitments of those shares by D. The non-equivocation mechanism is used to ensure that a *corrupted* D does not distribute different sets of encrypted and committed shares to the parties. Now once $n - t = t + 1$ parties confirm that they have received "consistent" n encrypted and committed shares, there must exist at least one honest privileged party with all n encrypted shares, who can transfer the individual encrypted shares to the individual parties. Non-equivocation with signatures ensures that corrupted privileged parties do not transfer incorrect encryptions.

Protocol Sup-PreMul-Sh (Section 4). The protocol takes as input an existing sharing $[v]$ of value v unknown to everybody including $\mathcal{A}$, such that all the parties are privileged, i.e., all the parties hold encryptions of all shares. The protocol then allows a dealer D to verifiably share its value u as well as $u \cdot v$ (i.e. $[u]$ and $[u \cdot v]$). The protocol ensures that $u \cdot v$ remains secure in general and u is secure for an *honest* D. The idea behind the protocol is that knowing the encrypted and committed shares of v and employing the homomorphic properties of encryptions and commitments, D can compute the encrypted and committed shares corresponding to $u \cdot v$ for his choice of u, even without knowing v. The dealer can then (non-equivocally) distribute the encrypted and committed shares to the parties. Once it is confirmed that $t + 1$ parties have

[3] We argue that the problem is inherently not solvable for $n = 2t + 1$ with only commitments usually employed in computationally secure VSS protocols [2, 14], and that we have to employ encryptions which allow a *single honest* party to procure encrypted shares of all the parties. Interestingly, the problem persists even when we assume the adversary $\mathcal{A}$ is only passive (but crashable), not active.

received all the n encrypted and committed shares of $u \cdot v$, it is ensured that there exists a honest privileged party, who can relay the individual encrypted shares of $u \cdot v$ to the respective parties.

We take a more bottom-up approach in the rest of the paper. We start our construction with subprotocols Sup-Sh and Sup-PreMul-Sh in Section 4. We then present our supervised multiplication triple generation in Section 5 and finally describe the complete AMPC protocol in Section 6.

3 Preliminaries

3.1 Model

We follow the standard active adversary and the asynchronous communication model [9, 17, 31, 32]. We consider a set $\mathcal{P} = \{P_1, \ldots, P_n\}$ of n parties connected by pairwise authenticated channels, where $n = 2t + 1$. A centralized static adversary $\mathcal{A}$ can actively corrupt any t out of the n parties and force them to deviate in any arbitrary manner. A party not under the control of $\mathcal{A}$ is called *honest*. The adversary $\mathcal{A}$ is modeled as a probabilistic polynomial time (PPT) algorithm, with respect to a security parameter κ. The communication channels between the parties are asynchronous with arbitrary but finite delay (i.e. the messages reach their destinations eventually). During a protocol execution, the message delivery order is decided by a *scheduler*. To capture the worst case scenario, we assume that the scheduler is controlled by $\mathcal{A}$. Nevertheless, the scheduler cannot change the "contents" of the messages exchanged between honest parties. By $[q, r]$ we represent the set $\{q, q + 1, \ldots, r\} \subset \mathbb{N}$.

Similar to [5,17,31,32], we assume that a protocol execution is preceded by an initialization function init involving set-up tasks such as initializing the parties and setting up their cryptographic keys. An actual protocol execution is then considered as a sequence of *atomic* steps, where a single party is active in each such step. A party is activated upon receiving a message, after which it performs some local computation and possibly outputs messages on its outgoing links. The scheduler controls the order of these atomic steps. At the beginning of the execution, each party will be in a special *start* state. A party is said to *terminate/complete* the computation if it reaches a *halt* state. A protocol execution is said to *be complete/terminate* when all honest parties complete/terminate the protocol execution. We assume that each protocol step has a *unique* publicly known identifier (label), and every message sent by a party during an execution has a publicly known unique *identity* associated with it.

3.2 Definitions

Computationally Secure AMPC. We briefly review computationally secure AMPC here and refer the readers to [31,32] for a formal definition. Informally, in an AMPC protocol Π_{MPC}, every party first provides its input to the computation (in a secure fashion). Due to the asynchronous nature of communication, the parties cannot wait to consider the inputs of all n parties, and instead they agree on inputs from a set $\mathcal{CORE}$ of $n - t$ parties. The parties then compute an "approximation" of f on the inputs from the $\mathcal{CORE}$ set and assuming a default value (say 0) as the remaining t inputs. For every possible $\mathcal{A}$ and for all possible inputs and random coins of the (honest) parties, we expect the following properties for a Π_{MPC} instance, except with a negligible probability: *(1) Termination:* all the honest parties eventually terminate Π_{MPC}; *(2) Correctness:* the honest parties obtain the correct output of the function f; *(3) Privacy:* the adversary $\mathcal{A}$ obtains no additional information other than what may be inferred from the inputs and outputs of the corrupted parties.

The above properties are formalized to the standard simulation-based definition following the real-world/ideal-world paradigm [9, 17, 31, 32].

Non-equivocation. Non-equivocation has been employed in several distributed systems [21–23, 34, 36]; however, it has not yet been formalized. We formalize non-equivocation and present an idealized definition

as follows:

Definition 3.1 (Non-Equivocation Oracle)**.** *A non-equivocation oracle* Neq *supports two types of queries:*

- $\mathsf{Neq\text{-}Sign}(\ell, m)$ *takes a label* $\ell \in \{0,1\}^*$ *and a message* $m \in \{0,1\}^*$ *as input. If* $\ell \in L$ *(where* L *is initially empty), then return* $\perp$*. Otherwise update* $L := L \cup \{\ell\}$ *and return* $\sigma_{\ell,m} \in \{0,1\}^*$*.*
- $\mathsf{Neq\text{-}Verify}(\ell, m, \sigma_{\ell,m})$ *takes a label* ℓ*, a message* m*, and a tag* $\sigma_{\ell,m}$ *as input and outputs either* 1 *or* 0 *indicating whether there was a query* $\mathsf{Neq\text{-}Sign}(\ell, m)$ *returning* $\sigma_{\ell,m}$*.*

For a tag $\sigma_{\ell,m} = \mathsf{Neq\text{-}Sign}(\ell, m)$ *for a label* ℓ *and a message* m*, the* Neq *oracle satisfies the following properties except with negligible probability:*
(1) Correctness*:* $\mathsf{Neq\text{-}Verify}(\ell, m, \sigma_{\ell,m}) = 1$. (2) Statefulness*: Any* re-invocation *of* Neq-Sign *for label* $\ell \in L$ *outputs* $\perp$*; i.e.,* $\mathsf{Neq\text{-}Sign}(\ell, \cdot) = \perp$. (3) Unforgeability*: For* $m' \neq m$*,* $\mathsf{Neq\text{-}Verify}(\ell, m', \sigma_{\ell,m}) = 0$.

Practical implementations of the Neq oracle with respect to a probabilistic polynomial time (PPT) adversary, which can invoke the oracle, have to satisfy the above properties with overwhelming probability.

Definition 3.2 (Secure implementation of Neq)**.** *Let* N *be an oracle answering the same queries as the* Neq *oracle does. Let* A *be a PPT adversary and* $Exp_A^N(\kappa)$ *the following experiment.*

1. *The oracle is initiated with the security parameter* κ*.*
2. A *is executed on* 1^κ *with the oracle by sending* Neq-Verify *and* Neq-Sign *queries to it and receives the answers.*
3. A *terminates outputting a tuple* $\langle l, m, \sigma \rangle$*, where* $\sigma \in \{0,1\}^{\Omega_\kappa}$ *.*
4. *If* $\mathsf{Neq\text{-}Verify}(l, m, \sigma) = 1$ *and* l, m *was not queried to* N*, output* 1 *otherwise output* 0*.*

N *provides a* secure Neq implementation *if* $\Pr[Exp_A^N(\kappa) = 1]$ *is negligible in* κ *for all PPT adversaries* A*.*

The tag of a secure implementation has usually a size of $\mathcal{O}(\kappa)$ bits for the security parameter κ, cf. Appendix B for instantiations.

In our protocols, we use an oracle Neq_i for every party $P_i \in \mathcal{P}$ such that only P_i can send Neq-Sign queries to the oracle Neq_i, while every party can send Neq-Verify queries to Neq_i,. This requires a mapping (e.g., using identifiers) from parties to their respective oracles. In Appendix B, we discuss implementations of non-equivocation.

For the sake of simplicity, we write P *non-equivocally transfers a message* m *on behalf of* Q *to party* R, for P sending m together with the tag $\sigma_{\ell,m}$ made by Q's non-equivocation oracle. Here, the label ℓ is assumed to be known by R due to the step in the protocol at which m is expected to be transferred by Q to any other party. As shorthand for P non-equivocally transferring a message m on behalf of P to some party R, we write that P *non-equivocally sends* m *to* R. Similarly, we write that R *non-equivocally receives* m *from* P *on behalf of* Q, for the event that R receives m together with a tag $\sigma_{\ell,m}$ which is successfully verified by R using Neq-Verify on $m, \sigma_{\ell,m}$ and the expected ℓ.

3.3 Employed Primitives

Homomorphic Encryptions and Commitments. We assume a *linear homomorphic encryption* scheme $(\mathsf{Enc}, \mathsf{Dec})$ which is IND-CPA secure. Every party P_i has its own key-pair $(\mathsf{pk}_i, \mathsf{sk}_i)$, for which the public key pk_i is known to all the parties. Given two ciphertexts $\mathbf{c}_{m_1} = \mathsf{Enc}_{\mathsf{pk}_i}(m_1, \cdot)$ and $\mathbf{c}_{m_2} = \mathsf{Enc}_{\mathsf{pk}_i}(m_2, \cdot)$, we require that there exist operations $\boxplus$ and $\boxdot$ on ciphertexts such that $\mathbf{c}_{m_1} \boxplus \mathbf{c}_{m_2} = \mathsf{Enc}_{\mathsf{pk}_i}(m_1 + m_2, \cdot)$ and $a \boxdot \mathbf{c}_{m_i} = \mathsf{Enc}_{\mathsf{pk}_i}(a \cdot m_i, \cdot)$ holds. We also assume a *linear homomorphic commitment* scheme $(\mathsf{Commit}, \mathsf{Open})$ with the analogous homomorphic operations; these are denoted by $\oplus$ and $\odot$. Furthermore, the commitment scheme has to provide at least the properties of *computational hiding* and *computational binding*. For the sake of readability, we leave the randomness of encryptions and commitments implicit. For instantiating the encryption and commitment, we propose to use the encoding-free additive El-Gamal encryption [19] and Pedersen commitments [39].

Zero-knowledge (ZK) Proofs. We assume the presence of the following two-party ZK protocols.
1) Zero knowledge proof of equality of encrypted and committed values. Protocol ZK-PoE does the following: there exists a Prover $\in \mathcal{P}$ who has computed and published a commitment $\mathsf{Com}_m = \mathsf{Commit}(m, \cdot)$ and a ciphertext $\mathbf{c}_m = \mathsf{Enc}_{\mathsf{pk}_i}(m, \cdot)$. Then using ZK-PoE, Prover can prove to *any* Verifier $\in \mathcal{P}$ (knowing $\mathsf{Com}_m, \mathbf{c}_m$ and pk_i) that the message encrypted in $\mathbf{c}_m$ is also committed in Com_m; i.e.,

$$\exists m, r_1, r_2 : \mathbf{c}_m = \mathsf{Enc}_{\mathsf{pk}_i}(m, r_1) \quad \wedge \quad \mathsf{Com}_m = \mathsf{Commit}(m, r_2).$$

2) Zero knowledge proof of correct pre-multiplication. Protocol PoCM does the following: there exist publicly known ciphertexts $\mathbf{c}_{v_j} = \mathsf{Enc}_{\mathsf{pk}_j}(v_j)$ and commitments $\mathsf{Com}_{v_j} = \mathsf{Commit}(v_j)$ for $j \in [1, n]$. There also exists a Prover $\in \mathcal{P}$ who has selected u and a polynomial $r(\cdot)$ of degree at most t with $r(0) = 0$. Let $r_j = r(j)$ for $j \in [0, n]$. In addition, the Prover computed and published $\mathsf{Com}_u = \mathsf{Commit}(u), \mathsf{Com}_r = \mathsf{Commit}(r(0))$ and $\{\mathsf{Com}_{r_j} = \mathsf{Commit}(r(j))\}_{j\in[1,n]}$. Moreover, by using the linearity properties of the commitment and encryption scheme, Prover has computed and published the encryption $\mathbf{c}_{u\cdot v_j+r_j}$ and commitment $\mathsf{Com}_{u\cdot v_j+r_j}$ of $u \cdot v_j + r_j$. Then using the protocol PoCM, Prover can prove to *any* Verifier $\in \mathcal{P}$ that the values $\mathbf{c}_{u\cdot v_j+r_j}$ and $\mathsf{Com}_{u\cdot v_j+r_j}$ were generated by multiplying $\mathbf{c}_{v_j}$ and Com_{v_j} with u followed by a rerandomization using the $r(\cdot)$ polynomial; i.e.,

$$\begin{aligned}\exists u, r(\cdot) : \mathsf{Com}_u = \mathsf{Commit}(u) \ \wedge \ \mathsf{degree}(r(\cdot)) \leq t \ \wedge \ r(0) = 0 \ \wedge \\ \mathsf{Com}_{u\cdot v_j+r_j} = u \odot \mathsf{Com}_{v_j} \oplus \mathsf{Com}_{r_j} \ \wedge \ \mathbf{c}_{u\cdot v_j+r_j} = u \boxdot \mathbf{c}_{v_j} \boxplus \mathsf{Enc}_{\mathsf{pk}_j}(r(j)).\end{aligned}$$

Both the ZK protocols are based on standard Σ-protocols [8] and have communication complexities $\mathcal{O}(\kappa)$ bits and $\mathcal{O}(n\kappa)$ bits respectively. See Appendix C for the instantiation based on the ZK protocols in [16].

Certificates of Claims. The concept of certificates was introduced in [32] to allow a Prover $\in \mathcal{P}$ to *publicly* prove the correctness of certain claims (like real-life certificates), without revealing any additional information. Suppose Prover wants to certify the validity of some statement m. Then Prover proves m to every *verifier* $P_i \in \mathcal{P}$ by executing an instance of the appropriate zero-knowledge (ZK) protocol. A verifier P_i, upon successful verification, sends a signature to Prover. Finally, upon obtaining $(n - t) = t + 1$ signatures, Prover concatenates them to construct a certificate α for the claim m. Since there exists at least one *honest* party in the set of $(n - t)$ parties who provided its signature on m only after verifying m, with overwhelming probability m is true. Moreover, even if t corrupted parties do not provide signatures for a *true* claim m, there always exist at least $(n - t)$ honest parties who will eventually provide signatures. Finally if Prover is *honest* then nothing beyond the validity of m is revealed.

We reduce the size of a certificate α to $\mathcal{O}(\kappa)$ by using a *threshold signature scheme* with threshold t [7]. In this case, the parties send signature *shares* instead of signatures and the Prover *combines* $(n - t)$ such shares into a single signature, instead of concatenating them.[4] In the rest of the paper, we represent the above abstraction as follows: let m be a claim and zkp be the corresponding two-party ZK protocol. Then by $\alpha = \mathsf{certify}_{\mathsf{zkp}}(m)$, we denote the above abstraction of constructing the certificate α for m. Similarly we say that "P_i verifies the certificate α for the claim m" to mean that that P_i verifies whether α is a valid (threshold) signature on m. Note that the communication cost of constructing α is the same as that of executing n instances of the corresponding ZK protocol zkp.

Asynchronous Reliable Broadcast (r-broadcast). This primitive [13, 44] allows a sender Sen $\in \mathcal{P}$ to send some message m identically to *all* the parties: When Sen is *honest*, all honest parties eventually terminate the protocol with output m; if Sen is *corrupted* and some honest party terminates with m', then every honest party eventually terminates with output m'. It is well-known that $n \geq 3t + 1$ is necessary and sufficient

[4] Note that the AMPC protocols of [7, 32] also assume a threshold signature setup, and in all the three cases, it can be replaced by a standard signature, albeit with a $\Theta(n)$ increment in the communication complexity.

to implement r-broadcast [13, 44]. However, assuming non-equivocation, we can design an r-broadcast protocol with $n \geq t+1$ and $\mathcal{O}(n^2(\ell+\kappa))$ bits of communication for broadcasting an ℓ-bit message [22,23]; see Appendix C. In the rest of the paper, "P_i *broadcasts* m" means that P_i as a Sen invokes an instance of r-broadcast for m. Similarly, "P_j *receives* m *from the broadcast of* P_i" means that P_j terminates the instance of r-broadcast invoked by P_i (as a Sen) with the output m.

Agreement on a Common Subset (ACS). This asynchronous primitive allows the parties to agree on a common subset of $(n-t)$ parties, who correctly invoked some protocol, say Π, satisfying the following requirements: (a) If Π is invoked by an honest party then all the (honest) parties eventually terminate the instance of Π; (b) If Π is invoked by a corrupted party and some honest party terminates the instance of Π, then every other honest party eventually does the same. The idea behind ACS is to execute n instances of an asynchronous Byzantine agreement (ABA) protocol, one on behalf of each party, to decide if it should be included in the common subset. Assuming a non-equivocation mechanism, ABA, and hence ACS, can be implemented with $n \geq 2t+1$ [22,23]. An efficient ACS protocol with expected communication complexity of $\mathcal{O}(n^3\kappa)$ bits can be obtained by using the non-equivocation oracle in the multi-valued ABA from [15].

3.4 Secret Sharing Notations

Given a secret $s \in \mathbb{Z}_p$, let $f(\cdot) \in \mathbb{Z}_p[x]$ be a *sharing polynomial* of degree at most t with $f(0) = s$. Here, for $j \in [1,n]$, $s_j = f(j)$ is the share of s for party P_j. Let $\mathbf{c}_{s_j} = \mathsf{Enc}_{\mathsf{pk}_j}(s_j)$ and $\mathsf{Com}_j = \mathsf{Commit}(s_j)$ respectively represent encryption and commitment of share s_j, and let $\mathsf{Com}_s = \mathsf{Commit}(s)$. We call $\{\mathbf{c}_{s_j}\}_{j\in[1,n]}$ the encrypted shares and $\{\mathsf{Com}_{s_j}\}_{j\in[1,n]}$ the committed shares of s.

Privileged party: party $P_i \in \mathcal{P}$ is called a privileged party if it holds the encrypted shares $\{\mathbf{c}_{s_j}\}_{j\in[1,n]}$.

$[\cdot]$-sharing: s is said to be $[\cdot]$-shared, if every (honest) $P_i \in \mathcal{P}$ holds s_i, $\{\mathsf{Com}_{s_j}\}_{j\in[1,n]}$ and Com_s. The information held by the (honest) parties corresponding to $[\cdot]$-sharing of s is denoted as $[s]$.

Due to the linearity of polynomial-based sharing and commitments, $[\cdot]$-sharing is *linear* in nature: given $[a]$, $[b]$ and a publicly known constant c, each party can locally compute its respective information corresponding to $[a+b]$ and $[c \cdot a]$, which we denote as $[a+b] = [a] + [b]$ and $[c \cdot a] = c \cdot [a]$.

4 Supervised Sharing Protocols

We present two protocols for generating $[\cdot]$-sharings with different properties under the supervision of a king P_{king}; if the protocols terminate, then an *honest* P_{king} will be a *privileged* party with respect to the generated sharings.

4.1 Protocol Sup-Sh: Supervised $[\cdot]$-sharing

Protocol Sup-Sh (Figure 2) allows a *dealer* $\mathsf{D} \in \mathcal{P}$ to select a value s and verifiably generate $[s]$ under the supervision of a designated *king* $P_{\mathsf{king}} \in \mathcal{P}$. The verifiability ensures that if the protocol terminates, then there exists a value, say $\overline{s}$, which will be $[\cdot]$-shared among the parties. If D is *honest* then $\overline{s} = s$ and $\overline{s}$ will be unknown to $\mathcal{A}$. Moreover, if P_{king} is *honest* then it will be a privileged party (thus holding the vector of n encrypted shares). The protocol always terminates for an *honest* D and P_{king} and has communication complexity $\mathcal{O}(n^2\kappa)$. In the protocol code, a certain part is in boldface; in the next section, we will argue that removing this part of the code leads to an AVSS scheme.

The idea behind the protocol is as follows: D first generates n (Shamir) shares $\{s_j\}_{j\in[1,n]}$ of s with threshold t and computes a commitment $\mathsf{Com}_{s_j} = \mathsf{Commit}(s_j)$[5] and an encryption $\mathbf{c}_{s_j} = \mathsf{Enc}_{\mathsf{pk}_j}(s_j)$ of

[5]The randomness used for computing the commitment of the shares should lie on a random degree-t polynomial (as in the Pedersen VSS [39]); however, we abstract it out here for easy reading.

Protocol $\mathsf{Sup\text{-}Sh}(\mathsf{D}, P_{\mathsf{king}}, s)$

i. D-DEPENDENT PHASE:

SHARE COMPUTATION AND CERTIFICATE GENERATION—The following code is executed only by D:

1. On having the secret s, select a *sharing polynomial* $f(\cdot)$ of degree at most t such that $f(0) = s$. For $j \in [1, n]$, compute the *share* $s_j = f(j)$, *encrypted share* $\mathbf{c}_{s_j} = \mathsf{Enc}_{\mathsf{pk}_j}(s_j)$ and *committed share* $\mathsf{Com}_{s_j} = \mathsf{Commit}(s_j)$. In addition compute the commitment $\mathsf{Com}_s = \mathsf{Commit}(s)$.
2. Non-equivocally send $\{\mathbf{c}_{s_j}\}_{j\in[1,n]}, \{\mathsf{Com}_{s_j}\}_{j\in[1,n]}, \mathsf{Com}_s$ to each P_i and start constructing the certificate $\alpha^{\mathsf{D}} = \mathsf{certify}_{\mathsf{ZK\text{-}PoE}}(\mathsf{claim}_{\alpha,\mathsf{D}})$ where $\mathsf{claim}_{\alpha,\mathsf{D}}$ is the claim that "$\forall j \in [1, n], \exists s_j : \mathbf{c}_{s_j} = \mathsf{Enc}_{\mathsf{pk}_j}(s_j) \wedge \mathsf{Com}_{s_j} = \mathsf{Commit}(s_j)$ and there exists a polynomial $f(\cdot)$ of degree at most t with $s_j = f(j)$ and $\mathsf{Com}_s = \mathsf{Commit}(f(0))$"; for this, run an instance of ZK-PoE for each s_j with every party. Broadcast α^{D} once it is constructed.

SHARE VERIFICATION AND CERTIFICATION—Every party $P_i \in \mathcal{P}$ including D and P_{king} executes the following code:

1. Wait to non-equivocally receive $\{\mathbf{c}_{s_j}\}_{j\in[1,n]}, \{\mathsf{Com}_{s_j}\}_{j\in[1,n]}, \mathsf{Com}_s$ from D. On receiving, perform the following verifications and upon successful verification, **additionally broadcast the message** $(\mathtt{OK}, \mathsf{D})$ **if** $P_i = P_{\mathsf{king}}$:
   (a) Verify whether the committed shares $\{\mathsf{Com}_{s_j}\}_{j\in[1,n]}$ define a unique polynomial of degree at most t; for this use Com_s and the properties of Vandermonde matrices [14, 28].
   (b) If the verification in the previous step is successful then participate (as a Verifier) in the instances of ZK-PoE with D to verify the claim $\mathsf{claim}_{\alpha,\mathsf{D}}$ and enable D to construct the certificate α^{D} for $\mathsf{claim}_{\alpha,\mathsf{D}}$.

ii. D-INDEPENDENT PHASE AND TERMINATION—Every party $P_i \in \mathcal{P}$ including D and P_{king} executes the following code:

1. Wait to receive the certificate α^{D} from the broadcast of D **and the message** $(\mathtt{OK}, \mathsf{D})$ **from the broadcast of** P_{king}. On receiving these messages, verify α^{D} for the claim $\mathsf{claim}_{\alpha,\mathsf{D}}$. Upon successful verification, do the following:
   (a) If $\{\mathbf{c}_{s_j}\}_{j\in[1,n]}, \{\mathsf{Com}_{s_j}\}_{j\in[1,n]}, \mathsf{Com}_s$ has been non-equivocally received from D then compute the share $s_i = \mathsf{Dec}_{\mathsf{sk}_i}(\mathbf{c}_{s_i})$. In addition, non-equivocally transfer *only* $\mathbf{c}_{s_j}$ and Com_{s_j} on behalf of D to party P_j for $j \in [1, n]$.
   (b) Else wait for $\mathbf{c}_{s_i}, \mathsf{Com}_{s_i}$ to be non-equivocally transferred to P_i on behalf of D from some $P_j \in \mathcal{P}$. Once transferred, compute $s_i = \mathsf{Dec}_{\mathsf{sk}_i}(\mathbf{c}_{s_i})$.
2. Wait to have s_i and Com_{s_i} and then non-equivocally transfer Com_{s_i} to every party $P_j \in \mathcal{P}$ on behalf of D.
3. Wait for $t + 1$ Com_{s_j}s to be non-equivocally transferred on behalf of D from $t + 1$ parties. Then using the properties of Vandermonde matrices, compute $\{\mathsf{Com}_{s_j}\}_{j\in[1,n]}$ and Com_s and terminate.

Figure 2: **Protocol for generating** $[s]$ **under** P_{king}

each share s_j. Notice that s_j is encrypted with the public key pk_j of party P_j, as the share s_j is intended *only* for P_j. In addition, D also computes the commitment $\mathsf{Com}_s = \mathsf{Commit}(s)$. The next task for D would be to send the intended shares to the individual parties and prove to them that they are indeed "valid" shares of s. To do this, instead of sending only s_i to P_i, the dealer *non-equivocally sends* $\{\mathbf{c}_{s_j}\}_{j\in[1,n]}, \{\mathsf{Com}_{s_j}\}_{j\in[1,n]}$ and Com_s to *every* party P_i and claims that the plaintext encrypted in $\mathbf{c}_{s_j}$ is committed in Com_{s_j} and that the values committed in $\{\mathsf{Com}_{s_j}\}_{j\in[1,n]}$ constitute valid shares of the secret committed in Com_s with threshold t. Notice that sending the full vector $\{\mathbf{c}_{s_j}\}_{j\in[1,n]}$ and $\{\mathsf{Com}_{s_j}\}_{j\in[1,n]}$ to each party is quite non-intuitive; however, as discussed in Section 2, it is the *crux* of our protocol to ensure that every party eventually receives its information corresponding to $[s]$.

To verify the claim of D, a party P_i on (non-equivocally) receiving the information from D uses the properties of Vandermonde matrices [14, 28] to verify whether the committed shares $\{\mathsf{Com}_{s_j}\}_{j\in[1,n]}$ constitute valid Shamir sharing of the secret committed in Com_s, with threshold t. In addition, P_i engages in n instances of ZK-PoE (one instance for each individual encrypted and committed share) with D. As D non-equivocally sends the information to the parties, it is ensured that all participants use the *same* $\{\mathbf{c}_{s_j}\}_{j\in[1,n]}, \{\mathsf{Com}_{s_j}\}_{j\in[1,n]}$ and Com_s. D then constructs a certificate α^{D} to support his claim and broadcasts this certificate. Only upon receiving a valid certificate α^{D} from the broadcast of D does a party proceed. Even when the dealer D is *corrupted*, a valid α^{D} implies that at least one honest party, say P_h, has verified D's claim. Moreover, P_h will be an *honest privileged party*. To satisfy our requirement that even (an honest) P_{king} should be a privileged party, we additionally enforce that every party should also receive an acknowl-

edgement from P_{king} (apart from the certificate from D) of having verified the claim of D before proceeding further in the protocol. All this computation and communication in the protocol constitutes what we call the D DEPENDENT PHASE.

Now notice that even if a valid certificate α^{D} from D and an acknowledgement from P_{king} arrives, it does not ensure that every (honest) P_i holds its information corresponding to $[s]$. Specifically, due to the asynchronicity or due to possible corrupted behavior of D, t honest P_is may be barred from their information corresponding to $[s]$. We solve this problem by using the following *two* "rounds" of communication, which we call the D INDEPENDENT PHASE, as it does not require any help from D. We *first* ask every *privileged party* to *non-equivocally transfer* $\mathbf{c}_{s_j}$ and Com_{s_j} on behalf of D to every corresponding P_j. If α^{D} has been generated, then there exists at least one *honest privileged* party P_h and so P_h can transfer $\mathbf{c}_{s_j}$ and Com_{s_j} to the respective P_js, who can decrypt $\mathbf{c}_{s_j}$ and obtain its share s_j. Thus it is ensured that each honest P_j eventually receives its share s_j and also Com_{s_j}.

We *next* ask each P_j to *non-equivocally transfer* Com_{s_j} to every other party on behalf of D. Now every party waits for $t+1$ committed shares to be non-equivocally transferred. As there exist $t+1$ honest parties who would have received their respective Com_{s_j} at the end of first "round" of the D INDEPENDENT PHASE, it is ensured that eventually every honest party will have $t+1$ correct committed shares transferred to them non-equivocally. Now using the linearity property of the commitment scheme and using the properties of Vandermonde matrices, every party can compute the remaining committed shares and also Com_s, thus possessing all the necessary information corresponding to $[s]$. It is important for the parties to follow the above communication "pattern" during the D INDEPENDENT PHASE; this ensures that the communication complexity of this phase is $\mathcal{O}(n^2\kappa)$.

Note that instead of non-equivocally committing the shares, D could have non-equivocally sent the commitments of the coefficients of the sharing polynomial (as done in some VSS schemes) and using the homomorphic property of the commitments, each (honest) privileged party could implicitly obtain commitment of each share. Although this could have simplified the protocol, this would have increased communication complexity of the D INDEPENDENT PHASE by $\Theta(n)$: as the non-equivocation mechanism is not homomorphic, the privileged parties cannot non-equivocally transfer only Com_{s_j} to P_j on the behalf of D; rather, the privileged parties have to non-equivocally transfer the full committed coefficients vector and only then can P_j derive the commitments of the shares.

The properties of Sup-Sh are proved in Appendix D.1. We follow the proof strategy from [7, 31, 32], and we do not consider the real-world/ideal-world paradigm. However, we stress that using standard techniques [17,24], our protocols can be easily adapted and proved secure in the more rigorous real-world/ideal-world paradigm.

4.1.1 Designing AVSS through a Variant of Sup-Sh

A closer look at Sup-Sh reveals that if P_{king} is *corrupted*, then the protocol may not terminate even if D is *honest*; this is because a *corrupted* P_{king} may not broadcast the $(\mathtt{OK}, \mathsf{D})$ message required for the termination. As a result, Sup-Sh fails to qualify as an AVSS sharing protocol, since an AVSS sharing protocol needs to terminate if D is *honest* [2, 14]. However, we can easily get rid of this problem as follows: we remove the (additional) requirement for P_{king} to broadcast the $(\mathtt{OK}, \mathsf{D})$ message; as a result, every party now waits *only* to receive a valid certificate from the broadcast of D during the D-INDEPENDENT PHASE. The resultant protocol (Sh) will allow (an honest) D to generate $[s]$, and its pseudocode can be obtained by removing the instructions in boldface in Figure 2.

Given $[s]$ generated using Sup-Sh and Sh, the standard reconstruction (Rec) protocol used in the existing computationally secure VSS [2, 14, 39] will allow the parties to *robustly* reconstruct s. In the protocol, each party sends its share to all the parties, which are verified with the corresponding commitment, available with the parties (as part of $[s]$). Once $t+1$ "correct" shares are received, the sharing polynomial, and hence s,

Protocol $\mathsf{Sup\text{-}PreMul\text{-}Sh}(\mathsf{D}, P_{\mathsf{king}}, \mathcal{P}, [v])$

Let $\{\mathbf{c}_{v_j}\}_{j\in[1,n]}, \{\mathsf{Com}_{v_j}\}_{j\in[1,n]}$ and Com_v be information corresponding to $[v]$ that is available to *all* the (honest) parties; each (honest) P_i will also have the share $v_i = f_v(i)$ of v, where $f_v(\cdot)$ denotes the sharing polynomial corresponding to $[v]$.

i. GENERATING $[u]$:

1. On having a value u, D invokes an instance of $\mathsf{Sup\text{-}Sh}$ to generate $[u]$ under the supervision of P_{king}. Let $f_u(\cdot)$ be the sharing polynomial selected by D and let $\{u_j\}_{j\in[1,n]}, \{\mathbf{c}_{u_j}\}_{j\in[1,n]}, \{\mathsf{Com}_{u_j}\}_{j\in[1,n]}$ and Com_u denote the shares, encrypted shares, committed shares and the commitment respectively, which are computed and communicated during this instance of $\mathsf{Sup\text{-}Sh}$.
2. Every party $P_i \in \mathcal{P}$ (including D and P_{king}) participates in the instance of $\mathsf{Sup\text{-}Sh}$ invoked by D and wait for its termination.

ii. GENERATING $[u \cdot v]$—The following code is executed by the respective parties only upon terminating the instance of $\mathsf{Sup\text{-}Sh}$:

a. D DEPENDENT PHASE

1. SHARE COMPUTATION AND CERTIFICATE GENERATION—The following code is executed only by D:
   (a) Select a random *masking polynomial* $r(\cdot)$ of degree at most t with $r(0) = 0$. For $j \in [1, n]$, compute $r_j = r(j)$ and commitment $\mathsf{Com}_{r_j} = \mathsf{Commit}(r_j)$, along with the commitment $\mathsf{Com}_r = \mathsf{Commit}(r(0))$.
   (b) For $j \in [1, n]$, using the homomorphic property of the encryption and commitment scheme, compute the *new encrypted share* $\mathbf{c}_{u\cdot v_j+r_j} = u \boxdot \mathbf{c}_{v_j} \boxplus \mathsf{Enc}_{\mathsf{pk}_j}(r_j)$ and the *new committed share* $\mathsf{Com}_{u\cdot v_j+r_j} = u \odot \mathsf{Com}_{v_j} \oplus \mathsf{Com}_{r_j}$. In addition, compute the *new commitment* $\mathsf{Com}_{u\cdot v} = u \odot \mathsf{Com}_v \oplus \mathsf{Com}_r$.
   (c) Non-equivocally send $\mathsf{Com}_u, \{\mathbf{c}_{u\cdot v_j+r_j}\}_{j\in[1,n]}, \{\mathsf{Com}_{u\cdot v_j+r_j}\}_{j\in[1,n]}, \{\mathsf{Com}_{r_j}\}_{j\in[1,n]}, \mathsf{Com}_r$ and $\mathsf{Com}_{u\cdot v}$ to every $P_i \in \mathcal{P}$ and start constructing the certificate $\beta^{\mathsf{D}} = \mathsf{certify}_{\mathsf{PoCM}}(\mathsf{claim}_{\beta,\mathsf{D}})$ where $\mathsf{claim}_{\beta,\mathsf{D}}$ is the claim that "$\exists u, r(\cdot) : \mathsf{Com}_u = \mathsf{Commit}(u) \wedge \mathsf{degree}(r(\cdot)) \leq t \wedge r(0) = 0 \wedge \mathsf{Com}_{u\cdot v_j+r_j} = u \odot \mathsf{Com}_{v_j} \oplus \mathsf{Com}_{r_j} \wedge \mathbf{c}_{u\cdot v_j+r_j} = u \boxdot \mathbf{c}_{v_j} \boxplus \mathsf{Enc}_{\mathsf{pk}_j}(r(j))$" by executing an instance of the ZK protocol PoCM for every party.
   (d) Broadcast the certificate β^{D} once it is constructed.
2. SHARE VERIFICATION AND CERTIFICATION—Every $P_i \in \mathcal{P}$ including D and P_{king} executes the following code:
   (a) Wait to non-equivocally receive $\mathsf{Com}_u, \{\mathbf{c}_{u\cdot v_j+r_j}\}_{j\in[1,n]}, \{\mathsf{Com}_{u\cdot v_j+r_j}\}_{j\in[1,n]}, \{\mathsf{Com}_{r_j}\}_{j\in[1,n]}, \mathsf{Com}_r$ and $\mathsf{Com}_{u\cdot v}$ from D. On receiving, perform the following verifications and upon successful verification, additionally broadcast the message $(\texttt{approve}, \mathsf{D})$ if you are P_{king} (i.e. if $P_i = P_{\mathsf{king}}$):
      i. Participate (as a $\mathsf{Verifier}$) in the instances of PoCM with D to verify the claim $\mathsf{claim}_{\beta,\mathsf{D}}$ and enable D to construct the certificate β^{D} for $\mathsf{claim}_{\beta,\mathsf{D}}$.

b. D INDEPENDENT PHASE AND TERMINATION—Every party $P_i \in \mathcal{P}$ including D and P_{king} executes the following code:

1. Wait to receive a valid certificate β^{D} from the broadcast of D for the claim $\mathsf{claim}_{\beta,\mathsf{D}}$ and the message $(\texttt{approve}, \mathsf{D})$ from the broadcast of P_{king}. On receiving, do the following:
   (a) If $\{\mathbf{c}_{u\cdot v_j+r_j}\}_{j\in[1,n]}, \{\mathsf{Com}_{u\cdot v_j+r_j}\}_{j\in[1,n]}$ and $\mathsf{Com}_{u\cdot v}$ has been non-equivocally received from D in the previous phase then compute the share $w_i = \mathsf{Dec}_{\mathsf{sk}_i}(\mathbf{c}_{u\cdot v_i+r_i})$. In addition, non-equivocally transfer *only* $\mathbf{c}_{u\cdot v_j+r_j}$ and $\mathsf{Com}_{u\cdot v_j+r_j}$ on behalf of D to party P_j for $j \in [1, n]$.
   (b) Else wait for $\mathbf{c}_{u\cdot v_i+r_i}$ and $\mathsf{Com}_{u\cdot v_i+r_i}$ to be non-equivocally transferred to P_i on behalf of D from some $P_j \in \mathcal{P}$. Once transferred, compute $w_i = \mathsf{Dec}_{\mathsf{sk}_i}(\mathbf{c}_{u\cdot v_i+r_i})$.
2. Wait to have $w_i = u \cdot v_i + r_i$ and $\mathsf{Com}_{u\cdot v_i+r_i}$ and then non-equivocally transfer $\mathsf{Com}_{u\cdot v_i+r_i}$ to every party $P_j \in \mathcal{P}$ on behalf of D.
3. Wait for $t + 1$ $\mathsf{Com}_{u\cdot v_j+r_j}$s to be non-equivocally transferred on behalf of D from $t + 1$ parties. Then using the properties of Vandermonde matrices, compute $\{\mathsf{Com}_{u\cdot v_j+r_j}\}_{j\in[1,n]}$ and $\mathsf{Com}_{u\cdot v+r}$ and terminate.

Figure 3: **Protocol for generating $[u]$ and $[u \cdot v]$ under P_{king}.**

is reconstructed. As there exist at least $t + 1$ honest parties whose shares will eventually be communicated among themselves, the Rec protocol eventually terminates. For the formal details, see [2,14,39]. Therefore, the pair of protocols $(\mathsf{Sh}, \mathsf{Rec})$ constitutes an AVSS scheme with $n = 2t+1$ and communication complexity $\mathcal{O}(n^2\kappa)$ bits. The scheme is also publicly verifiable [43] as any third party can verify the consistency of the shares using the valid certificate produced by D.

4.2 Supervised Pre-multiplication Protocol

Protocol Sup-PreMul-Sh (Figure 3) takes as input a $[\cdot]$-shared value v, where v is completely random and unknown; in addition, every (honest) party is a privileged party with respect to $[v]$ (having all the n encrypted shares of v). The protocol allows a dealer $\mathsf{D} \in \mathcal{P}$ to select a value u and verifiably generate $[u]$ as well as $[u \cdot v]$, under the supervision of a designated *king* $P_{\mathsf{king}} \in \mathcal{P}$. The verifiability ensures that if the protocol terminates, then there exists a value, say $\overline{u}$, such that $\overline{u}$ and $\overline{u} \cdot v$ will be $[\cdot]$-shared among the parties. During the protocol, v and $\overline{u} \cdot v$ remains private, and when D is *honest* then $\overline{u} = u$ and u is also unknown to $\mathcal{A}$. Finally if P_{king} is *honest* then it will be a privileged party with respect to $[\overline{u}]$ as well as $[\overline{u} \cdot v]$. The protocol always terminates for an *honest* D and P_{king} and has communication complexity $\mathcal{O}(n^2\kappa)$ bits. The protocol works along the same lines as Sup-Sh and uses Sup-Sh as a black-box.

Let $\{\mathbf{c}_{v_j}\}_{j\in[1,n]}, \{\mathsf{Com}_{v_j}\}_{j\in[1,n]}$ and Com_v be the encrypted shares, committed shares and the commitment corresponding to $[v]$ that is available to all the parties. Each P_i will also have the share $v_i = f_v(i)$ of v, where $f_v(\cdot)$ with $f_v(0) = v$ denotes the sharing polynomial corresponding to $[v]$. Thus $\mathbf{c}_{v_j} = \mathsf{Enc}_{\mathsf{pk}_j}(v_j), \mathsf{Com}_{v_j} = \mathsf{Commit}(v_j)$ and $\mathsf{Com}_v = \mathsf{Commit}(v)$. To generate $[u]$, D first invokes an instance of Sup-Sh. The next task for D would be to generate $[u \cdot v]$ and that without knowing v. To do this, we observe that $u \cdot f_v(\cdot)$ constitutes a "potential" sharing polynomial for $[u \cdot v]$; this is because $u \cdot f_v(\cdot)$ has degree at most t with $u \cdot v$ as the constant term. This also implies that $\{u \cdot v_j\}_{j\in[1,n]}$ constitute valid shares for $u \cdot v$ and so using the homomorphic properties of the encryption and commitment scheme, D can compute the encrypted shares $\{\mathbf{c}_{u\cdot v_j} = u \boxdot \mathbf{c}_{v_j}\}_{j\in[1,n]}$, the committed shares $\{\mathsf{Com}_{u\cdot v_j} = u \odot \mathsf{Com}_{v_j}\}_{j\in[1,n]}$ and the commitment $\mathsf{Com}_{u\cdot v} = u \odot \mathsf{Com}_v$, corresponding to $[u \cdot v]$ and distribute the same to the parties. But this violates the privacy of u, as a corrupted P_i can easily obtain u by decrypting $\mathbf{c}_{u\cdot v_i}$; this is because v_i will be available to P_i (as a part of $[v]$). Thus we use a slightly different idea; we observe that $u \cdot f_v(\cdot) + r(\cdot)$ also constitutes a sharing polynomial for $u \cdot v$, provided $r(\cdot)$ is a polynomial (we call it a *masking polynomial*) of degree at most t with 0 as the constant term. Thus $\{w_j = u \cdot v_j + r(j)\}$ constitutes valid shares for $u \cdot v$, with $\{\mathbf{c}_{w_j} = \mathbf{c}_{u\cdot v_j + r(j)} = u \boxdot \mathbf{c}_{v_j} \boxplus \mathsf{Enc}_{\mathsf{pk}_j}(r(j))\}_{j\in[1,n]}, \{\mathsf{Com}_{w_j} = \mathsf{Com}_{u\cdot v_j + r_j} = u \odot \mathsf{Com}_{v_j} \oplus \mathsf{Com}(r(j))\}_{j\in[1,n]}$ and $\mathsf{Com}_{u\cdot v} = u \odot \mathsf{Com}_v \oplus \mathsf{Com}(r(0))$ constituting the required information corresponding to $[u \cdot v]$.

The rest of the protocol now follows the same principle as Sup-Sh, except that PoCM is used instead of ZK-PoE by D to construct the certificate that he has distributed "correct" information corresponding to $[u \cdot v]$. Intuitively as nothing about the $r(j)$s is revealed to the parties, u remains private; see Appendix D.2 for the details.

5 Supervised Triple Generation

We now present an asynchronous protocol SupTripGen which makes it possible to generate $[\cdot]$-sharing $([u], [v], [w])$ of a uniformly random multiplication triple (u, v, w), unknown to $\mathcal{A}$, under the supervision of a king $P_{\mathsf{king}} \in \mathcal{P}$ with $\mathcal{O}(n^3\kappa)$ communication complexity. The protocol is further designed using two sub-protocols, namely Sup-Second and Sup-FirAndThd, based on the same principle, but with different outputs.

5.1 Generating the Second Component of the Triple

Protocol Sup-Second generates $[\cdot]$-sharing $[v]$ of a uniformly random value v unknown to $\mathcal{A}$, under the supervision of a king P_{king}; additionally each (honest) party will be a privileged party (i.e. having all the n encrypted shares) with respect to $[v]$ (the need for this additional requirement will be clear in the next subsection). The protocol is based on the following idea: each party $P_i \in \mathcal{P}$ is asked to act as a D and invoke an instance Sup-Sh to generate $[\cdot]$-sharing $[v^{(i)}]$ of a uniformly random value, under the supervision of P_{king}; call this instance $\mathsf{Sup\text{-}Sh}_i$. Now let $\mathcal{T}_{\mathsf{king}}$ be the set of $t + 1$ P_ks, such that P_{king} has locally

terminated Sup-Sh$_k$. We set v to be $v = \sum_{P_k \in \mathcal{T}_{\mathsf{king}}} v^{(k)}$. Then v will be random and unknown to $\mathcal{A}$, as there exists at least one *honest* $P_k \in \mathcal{T}_{\mathsf{king}}$ and the corresponding $v^{(k)}$ will be random. In addition, P_{king} is supposed to be a privileged party for *every* $[v^{(k)}]$ and thus P_{king} can compute all the n encrypted shares, corresponding to $[v]$. We ask P_{king} to broadcast $\mathcal{T}_{\mathsf{king}}$ and every party verifies whether they indeed have locally terminated the instances Sup-Sh$_k$s corresponding to P_ks in $\mathcal{T}_{\mathsf{king}}$. This way every party can compute its share of v and all the committed shares of v. What remains is to ensure that every party holds all the n encrypted shares corresponding to $[v]$. However, the honest parties may not have the required information to compute them (like the king), since corresponding to each $v^{(k)}$, they may be non-privileged. The way out of this problem is that P_{king} has to "support" all the parties by broadcasting the n encrypted shares of v, which costs $\mathcal{O}(n^3\kappa)$ bits. However, we have to ensure that a potentially *corrupted* P_{king} indeed broadcasted the correct encrypted shares of v. For this we also ask P_{king} to *non-equivocally transfer* the encrypted shares corresponding to each $[v^{(k)}]$ to every other party; we stress that this information is communicated over the *point-to-point* channels and *not* broadcasted. This costs $\mathcal{O}(n^3\kappa)$ bits. Ideally P_{king} should be able to perform the transfer as he is a privileged party during each Sup-Sh$_k$

Once this information is *non-equivocally transferred* to a party, it can *re-compute* the encrypted shares of v and match it with what P_{king} has broadcasted and announce the same publicly. If at least $t+1$ parties announced "positively", then it implies that P_{king} has indeed broadcasted the correct encrypted shares of v, as there exists at least one honest party, to whom the the encrypted shares corresponding to the $[v^{(k)}]$s were non-equivocally transferred by P_{king} and who would have locally recomputed and verified what P_{king} broadcasted as encrypted shares of v. Due to space constraints, the protocol and its properties are given in Appendix E.1.

5.2 Generating First and Third Components of the Triple

Protocol Sup-FirAndThd takes as input a $[\cdot]$-shared value v, where v is uniformly random and unknown, such that every party is a privileged party and possesses all the n encrypted shares of v. It then generates $[\cdot]$-sharing $[u]$ of a uniformly random value u unknown to $\mathcal{A}$, along with the $[\cdot]$-sharing $[u \cdot v]$, under the supervision of a king P_{king}. The protocol follows the same principle as Sup-Second, except that each party P_i now invokes an instance Sup-PreMul-Sh$_i$ of Sup-PreMul-Sh with $[v]$ and a uniformly random value $u^{(i)}$ to generate $[u^{(i)}]$ and $[u^{(i)} \cdot v]$; this is possible because each P_i is now a privileged party with respect to $[v]$, which is a pre-condition for Sup-PreMul-Sh. Now we set $u = \sum_{P_k \in \mathcal{T}_{\mathsf{king}}} u^k$ and $w = \sum_{P_k \in \mathcal{T}_{\mathsf{king}}} u^k \cdot v$ and accordingly the parties output $[u]$ and $[w]$; here $\mathcal{T}_{\mathsf{king}}$ will be the set of $n - t = (t+1)$ P_ks such that the instance Sup-PreMul-Sh$_k$ has been locally terminated by P_{king}. Note that unlike Sup-Second, we do not demand that the parties also output all the n encrypted shares of u and w. The complete formal details can be found in Appendix E.2.

5.3 Sup-Second + Sup-FirAndThd $\Longrightarrow$ SupTripGen

Protocol SupTripGen for the supervised generation of a shared random multiplication triple consists of the following two steps: **(1).** The parties execute the protocol Sup-Second and output $[v]$. **(2).** On terminating Sup-Second, the parties execute the protocol Sup-FirAndThd and output $[u]$ and $[w] = [u \cdot v]$. The parties then output $([u], [v], [w])$ and terminate. For details, see Appendix E.3.

6 The NeqAMPC Protocol

The idea of the NeqAMPC protocol has already been discussed in Section 2. It follows a sequence of three phases, each implemented by a sub-protocol, and in which a party proceeds to the next phase only

after completing the current phase: the pre-processing phase generates $[\cdot]$-sharing of $c_M + c_R$ random multiplication triples. This is followed by an input phase, where each party $[\cdot]$-shares its private input, using an instance of Sh; due to the asynchronicity, the parties agree on a set of $(n-t)$ input providers and substitute 0 as the default input of the remaining t parties. Finally during the computation phase, the parties evaluate the circuit in a shared fashion, using the idea discussed earlier. As the protocol is standard [31, 32], we present it in Appendix F and state the following theorem.

Theorem 6.1 (The NeqAMPC Theorem). *Let $f : \mathbb{Z}_p^n \rightarrow \mathbb{Z}_p$ be a function expressed as an arithmetic circuit over $\mathbb{Z}_p$, consisting of c_M multiplication gates and c_R random gates. Assume a non-equivocation oracle associated with every party. Then for every possible $\mathcal{A}$ and for every possible scheduler, there exists a computationally secure AMPC protocol to securely compute f with communication complexity $\mathcal{O}(((c_M + c_R) \cdot n^3 + n^3)\kappa)$ bits.*

References

[1] M. Backes, A. Datta, and A. Kate. Asynchronous Computational VSS with Reduced Communication Complexity. In *CT-RSA*, pages 259–276, 2013.

[2] M. Backes, A. Kate, and A. Patra. Computational Verifiable Secret Sharing Revisited. In *Proc. of ASIACRYPT'11*, pages 590–609, 2011.

[3] D. Beaver. Efficient Multiparty Protocols Using Circuit Randomization. In *Proc. of CRYPTO'91*, pages 420–432, 1991.

[4] Z. Beerliová-Trubíniová and M. Hirt. Efficient Multi-party Computation with Dispute Control. In *Proc. of TCC'06*, pages 305–328, 2006.

[5] Z. Beerliová-Trubíniová and M. Hirt. Simple and Efficient Perfectly-Secure Asynchronous MPC. In *Proc. of ASIACRYPT'07*, pages 376–392, 2007.

[6] Z. Beerliová-Trubíniová and M. Hirt. Perfectly-Secure MPC with Linear Communication Complexity. In *Proc. of TCC'08*, pages 213–230, 2008.

[7] Z. Beerliová-Trubíniová, M. Hirt, and J. B. Nielsen. On the Theoretical Gap between Synchronous and Asynchronous MPC Protocols. In *Proc. of PODC'10*, pages 211–218, 2010.

[8] M. Bellare and O. Goldreich. On defining proofs of knowledge. In *Proc. of CRYPTO'92*, pages 390–420, 1992.

[9] M. Ben-Or, R. Canetti, and O. Goldreich. Asynchronous Secure Computation. In *Proc. of STOC'93*, pages 52–61, 1993.

[10] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation. In *Proc. of STOC'88*, pages 1–10, 1988.

[11] M. Ben-Or, B. Kelmer, and T. Rabin. Asynchronous Secure Computations with Optimal Resilience (Extended Abstract). In *Proc. of PODC'94*, pages 183–192, 1994.

[12] E. Ben-Sasson, S. Fehr, and R. Ostrovsky. Near-Linear Unconditionally-Secure Multiparty Computation with a Dishonest Minority. In *Proc. of CRYPTO'12*, pages 663–680, 2012.

[13] G. Bracha. An Asynchronous [(n-1)/3]-Resilient Consensus Protocol. In *Proc. of PODC'84*, pages 154–162, 1984.

[14] C. Cachin, K. Kursawe, A.Lysyanskaya, and R. Strobl. Asynchronous Verifiable Secret Sharing and Proactive Cryptosystems. In *Proc. of CCS'02*, pages 88–97, 2002.

[15] C. Cachin, K. Kursawe, F. Petzold, and V. Shoup. Secure and efficient asynchronous broadcast protocols. In *CRYPTO*, pages 524–541, 2001.

[16] J. Camenisch and M. Stadler. Proof systems for general statements about discrete logarithms. Technical report, 1997. 260, Dept. of Computer Science, ETH Zurich.

[17] R. Canetti. *Studies in Secure Multiparty Computation and Applications*. PhD thesis, The Weizmann Institute of Science, 1996.

[18] D. Chaum, C. Crépeau, and I. Damgård. Multiparty unconditionally secure protocols. In *Proc. of STOC'88*, pages 11–19, 1988.

[19] B. Chevallier-Mames, P. Paillier, and D. Pointcheval. Encoding-free elgamal encryption without random oracles. In *Proc. of PKC'06*, pages 91–104, 2006.

[20] A. Choudhury, M. Hirt, and A. Patra. Unconditionally secure asynchronous multiparty computation with linear communication complexity. To appear in DISC 2013. Available as Cryptology ePrint Archive `http://eprint.iacr.org/`, Report 2012/517, 2012.

[21] B.-G. Chun, P. Maniatis, S. Shenker, and J. Kubiatowicz. Attested append-only memory: making adversaries stick to their word. In *Proc. of SOSP'07*, pages 189–204, 2007.

[22] A. Clement, F. Junqueira, A. Kate, and R. Rodrigues. On the (limited) Power of Non-Equivocation. In *Proc. of PODC'12*, pages 301–308, 2012.

[23] M. Correia, G. S. Veronese, and L. C. Lung. Asynchronous Byzantine Consensus with 2f+1 Processes. In *Proc. of SAC'10*, pages 475–480, 2010.

[24] R. Cramer, I. Damgård, and J. B. Nielsen. Multiparty Computation from Threshold Homomorphic Encryption. In *Proc. of EUROCRYPT'01*, pages 280–299, 2001.

[25] I. Damgård and J. B. Nielsen. Scalable and Unconditionally Secure Multiparty Computation. In *Advances in Cryptology—CRYPT0*, pages 572–590, 2007.

[26] D. Dolev and H. R. Strong. Authenticated Algorithms for Byzantine Agreement. *SIAM J. Comput.*, 12(4):656–666, 1983.

[27] P. Feldman and S. Micali. Byzantine Agreement in Constant Expected Time (and Trusting No One). In *FOCS*, pages 267–276, 1985.

[28] R. Gennaro, M. Rabin, and T. Rabin. Simplified VSS and Fast-Track Multiparty Computations with Applications to Threshold Cryptography. In *Proc. of PODC'98*, pages 101–111, 1998.

[29] O. Goldreich, S. Micali, and A. Wigderson. How to play ANY mental game. In *Proc. of STOC '87*, pages 218–229, 1987.

[30] M. Hirt and J. B. Nielsen. Robust Multiparty Computation with Linear Communication Complexity. In *Proc. of CRYPTO'06*, pages 463–482, 2006.

[31] M. Hirt, J. B. Nielsen, and B. Przydatek. Cryptographic Asynchronous Multi-party Computation with Optimal Resilience (Extended Abstract). In *Proc. of EUROCRYPT'05*, pages 322–340, 2005.

[32] M. Hirt, J. B. Nielsen, and B. Przydatek. Asynchronous Multi-Party Computation with Quadratic Communication. In *Proc. of ICALP'08*, pages 473–485, 2008.

[33] A. Jaffe, T. Moscibroda, and S. Sen. On the price of equivocation in byzantine agreement. In *PODC*, pages 309–318, 2012.

[34] R. Kapitza, J. Behl, C. Cachin, T. Distler, S. Kuhnle, S. V. Mohammadi, W. Schröder-Preikschat, and K. Stengel. Cheapbft: resource-efficient byzantine fault tolerance. In *EuroSys*, pages 295–308, 2012.

[35] L. Lamport, R. E. Shostak, and M. C. Pease. The Byzantine Generals Problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.

[36] D. Levin, J. R. Douceur, J. R. Lorch, and T. Moscibroda. Trinc: Small trusted hardware for large distributed systems. In *Proc. of NSDI'09*, pages 1–14, 2009.

[37] J. M. McCune, Y. Li, N. Qu, Z. Zhou, A. Datta, V. Gligor, and A. Perrig. Trustvisor: Efficient tcb reduction and attestation. In *Proceedings of the 2010 IEEE Symposium on Security and Privacy*, SP '10, pages 143–158, 2010.

[38] A. Patra, A. Choudhary, and C. P. Rangan. Efficient Asynchronous Byzantine Agreement with Optimal Resilience. In *Proc. of PODC'09*, pages 92–101, 2009.

[39] T. P. Pedersen. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In *Proc. of CRYPTO'91*, pages 129–140, 1991.

[40] T. Rabin and M. Ben-Or. Verifiable Secret Sharing and Multiparty Protocols with Honest Majority (Extended Abstract). In *Proc. of STOC'89*, pages 73–85, 1989.

[41] A. Shamir. How to Share a Secret. *Commun. ACM*, 22(11):612–613, 1979.

[42] K. Srinathan and C. P. Rangan. Efficient Asynchronous Secure Multiparty Distributed Computation. In *Proc. of INDOCRYPT'00*, pages 117–129, 2000.

[43] M. Stadler. Publicly Verifiable Secret Sharing. In *Proc. of EUROCRYPT'96*, pages 190–199, 1996.

[44] S. Toueg. Randomized Byzantine Agreements. In *Proc. of PODC'84*, pages 163–178, 1984.

[45] `http://www.trustedcomputinggroup.org/developers/trusted_platform_module`.

[46] `http://www.arm.com/products/processors/technologies/trustzone.php`.

[47] A. Yao. Protocols for secure computations. In *Proc. of FOCS'82*, pages 160–164. IEEE, 1982.

A Analysis of the AMPC Protocol of [7]

The AMPC protocol of [7] operates over $\mathbb{Z}_N$. The input stage consists of a synchronous broadcast round, where every party encrypts its input and broadcasts it, along with a NIZK proof that it knows the underlying plaintext, corresponding to the ciphertext. Thus the input stage consists of a broadcast of $\mathcal{O}(n\kappa)$ bits. The secure evaluation of the circuit is then done using the *king-slave* paradigm, where every party in $\mathcal{P}$ acts as a king and all the n parties (including the king) act as slaves and perform the computation on behalf of the king, so as to enable the king to obtain the output of the function (to be computed). So in principle, the actual circuit is evaluated n times, once on behalf of each party. We focus on the actual communication done among the slaves to evaluate the circuit on the behalf of a single king. Due to the homomorphic property of the encryption scheme, evaluating the addition gates required no interaction among the slaves. For a multiplication gate, a random encrypted multiplication triple unknown to $\mathcal{A}$ is generated for the slaves, under the supervision of the king. For this, the parties begin with a publicly known default encrypted multiplication triple, which is then randomized to new encrypted triples, for $t+1$ iterations, by different slaves; the triple obtained after $t+1$th iteration is taken as the final triple. In every iteration, to perform the randomization of an encrypted triple, the king sends a randomization request to all the n slaves. A slave, on receiving a randomization request, performs the randomization, and to prove to the king that he has the performed the randomization correctly, the slave provides a NIZK proof of $\mathcal{O}(\kappa)$ bits to every other slave, so as to obtain a threshold signature. In short, in every iteration, each slave performs a randomization and communicates $\mathcal{O}(n\kappa)$ bits to the other slaves to prove that he has the performed the randomization correctly. Therefore in total, each iteration involves a communication of $\mathcal{O}(n^2\kappa)$ bits, and so $t+1$ iterations require a total communication of $\mathcal{O}(n^3\kappa)$ bits. Thus evaluating a single multiplication gate under the king requires a communication of $\mathcal{O}(n^3\kappa)$ bits and so for c_M multiplication gates, it will incur a total communication of $\mathcal{O}(c_M n^3\kappa)$ bits for a single king. Therefore, for n kings, the protocol will require an overall communication of $\mathcal{O}(c_M n^4\kappa)$ bits.

B Non-equivocation Implementations

Chun et al. [21] observed that the fundamental distributed computing problem of "Byzantine generals" has been proved unsolvable for three parties when one of those is corrupted [35] precisely because the corrupted party can spread contradictory messages to the remaining two honest parties. They demonstrated that if we can stop the corrupted party from *equivocating* (i.e., making conflicting statements to different honest parties) using a small trusted module on every party, it is possible to improve the resilience of distributed computing tasks in the asynchronous setting. They implemented non-equivocation using a (signed) trusted log abstraction called Attested Append-Only Memory (A2M), and designed a Byzantine-tolerant state machine replication (SMR) system for $n \geq 2t + 1$.

Levin et al. [36] further simplified the trust assumption from [21] and showed that a minimal trusted module called TrInc consisting of only a non-decreasing counter $c \in \mathbb{N}$ and a signing key-pair $(\mathsf{pk}, \mathsf{sk})$ is sufficient to generate A2M logs and to implement SMR with $n \geq 2t + 1$. Conceptually, TrInc provides unique, once-in-a-lifetime attestations, and implements non-equivocation using the fact that the counter cannot be decreased, and consequently for every counter value c there is at most one message signed by the module.

Levin et al. implemented TrInc on Gemalto .NET SmartCards. It is also possible to implement TrInc over the computers enabled with TPM chips, where its features of trusted identity, sealed storage, and remote code attestation will be used. Although the TPM specification does not readily implement a trusted counter, it can be achieved using a TPM-based hypervisor framework such as TrustVisor [37].

Recently, Clement el al. [22] observed that the definitional non-equivocation itself actually does not provide any improvement to the resiliency bound; however, combining it with digital signatures provides the improvements observed in [21, 23, 36], where the transferability of verifications provided by signatures is a key along with non-equivocation. They further noted that this combination also provides a generic transformation that allows a crash fault tolerant protocol to tolerate the same number of Byzantine faults. Nevertheless, their generic transformation does not consider privacy (or confidentiality), required in the AVSS and AMPC tasks, and we observe in this paper that encryptions and zero-knowledge proofs are required along with signatures when privacy is required.

C Instantiation of Various Primitives

In this section we instantiate the primitives we used in our protocol construction. These are the following: commitment scheme, encryption scheme and zero-knowledge proofs. As commitment scheme we simply use Pedersen commitments [39], i.e., we commit to m using randomness r by computing $g^m h^r$ for two generators g, h of a suitable group. In particular, this commitment scheme has the properties we required in section 3.3.

C.1 Encryption scheme Enc

We use the encoding-free ElGamal encryption scheme proposed in [19]. Let p, q be primes such that $q \mid p-1$ and let g be an integer of order pq modulo p^2 that generates a group $\mathbb{G} = \langle g \rangle$. Let $\langle x, y \rangle$ be the unique integer in $\mathbb{Z}_{pq}$ such that $\langle x, y \rangle = x \mod p$ and $\langle x, y \rangle = y \mod q$. The class of an element of $w = g^{\langle x,y \rangle} \in \mathbb{G}$ is x. We denote the class of w as $[\![w]\!]$. It is easy to see that $[\![w \cdot w']\!] = [\![w]\!] + [\![w']\!]$ and that $\mathbb{G}_p := \langle g \mod p \rangle$ has order q.

Definition C.1 (Encoding-Free ElGamal [19])**.**

1. Setup*: Let p, q be primes such that $q \mid p - 1$ and let g be a generator of $\mathbb{G}_p$ of order q.*

2. Key generation*: The private key is a random $x \in \mathbb{Z}_q$; the public key is $h = g^x \mod p$.*

3. Encryption*: The encryption algorithm chooses randomly an $r \in \mathbb{Z}_q$ and computes*

$$\mathsf{Enc}(m, r) = (g^r \mod p, m + [\![h^r \mod p]\!] \mod p)$$

4. Decryption*: The decryption works as follows:*

$$\mathsf{Dec}(x, (R, c)) = c - [\![R^x \mod p]\!] \mod p$$

The scheme is CPA-secure under the Decisional Class Diffie-Hellman problem [19] which is defined as the Diffie-Hellman problem under the class operation $[\![\cdot]\!]$. It has been shown that the Computational Class Diffie-Hellman problem is equivalent to the Computational Diffie-Hellman problem. However, the same result for the decisional case has not been shown.

We define two operations on encryptions in order to describe their homomorphic properties.

Definition C.2 (Homorphic operations)**.**
Let $a, b, c, d, v \in \mathbb{Z}_q$.

- $\mathsf{Enc}(a, b) \boxplus \mathsf{Enc}(c, d) := (g^{b+d} \mod p, (a + c) + [\![h^{b+d} \mod p]\!] \mod p)$
- $v \boxdot \mathsf{Enc}(a, b) := (g^{vb} \mod p, va + [\![h^{vb} \mod p]\!] \mod p)$

Note that these operations can be computed without knowing the content of the ciphertext. For the first one this is the case because $[\![w]\!] + [\![w']\!] = [\![w \cdot w']\!]$. The latter operation can be done by iteratively applying the first operation.

The encryption scheme has the properties required in section 3.3. For more details, we refer to [19].

C.2 Zero-knowledge Proof Scheme ZK-PoE

In this subsection we will present a proof scheme using the primitivies instantiated previously. The structure of the ZK-PoE-protocol is based on Σ-protocols [8]. These protocols have been well-studied and are usually easy to understand. Intuitively, a Σ-protocol is a proof that a party knows a witness w for a statement x such that $(x, w) \in \mathcal{R}$. The relation $\mathcal{R}$, which can be proven, is specific for the Σ protocol.

Definition C.3 (Σ-Protocol [8])**.** *Let $\mathcal{R}$ be a relation. A Sigma Protocol for a relation $\mathcal{R}$ is a 3-round protocol, i.e., it consists of four algorithms (P_1, P_2, V_1, V_2) where P_1, P_2 is for the prover and V_1, V_2 is for the verifier such that the following holds. Let x, w be bitstrings and $(a, s_P) := P_1(x, w)$, $(c, s_V) := V_1(x, a)$, $p := P_2(c, s_P)$ and $d := V_2(s_V, p)$. Then the following holds:*

1. Completeness*: If $(x, w) \in \mathcal{R}$ and the prover and verifier are both honest, then the verifier always outputs $d = 1$.*

2. Special soundness*: There is an extraction algorithm E such that for any fixed statement x and for any two transcripts (a, c, p) and (a, c', p') such that the verifier outputs 1 for both and where $c \neq c'$ holds, it follows that $(x, \mathsf{E}(a, c, c', p, p')) \in \mathcal{R}$.*

3. Special honest verifier zero-knowledge*: There is a simulator S such that for any x for which there is a w for which $(x, w) \in \mathcal{R}$ holds, the simulator produces on input x and random input c a transcript (a, c, p) which is computationally indistinguishable from a protocol transcript generated during the real execution of the protocol.*

The relation for ZK-PoE we need to prove is the following:

$$\exists m : \mathsf{Dec}_{\mathsf{sk}_i}(\mathbf{c}_m) = m \quad \wedge \quad \mathsf{Open}(m, \mathsf{Com}_m) = 1.$$

Since the corresponding prover does not know sk_i, we prove the first part of the statement by showing that we know a randomness that leads to $\mathbf{c}_m$. Using the instantiations we get the statement:

$$\exists m, R_e, R_c : \mathbf{c}_m = \mathsf{Enc}(m, R_e) \wedge \mathsf{Com}_m = g^m h^{R_c}$$

In more detail, for $\mathbf{c}_m = (\mathbf{c}_{m,1}, \mathbf{c}_{m,2})$ we want to prove $\mathbf{c}_{m,1} = g^{R_e}$ and $\mathbf{c}_{m,2} = m + [\![h^{R_e}]\!]$. In order to prove the equations for Com_m and $\mathbf{c}_{m,1}$, we use the technique described in [16].

The proof scheme for these equations works as follows. For a set of generators $\{g_i\}$ we prove that we know $\{x_i\}$ such that $y = \prod_i g_i^{x_i}$ by randomly choosing r_i and sending $t = \prod_i g_i^{r_i}$ to the verifier. The verifier then sends a challenge c and the prover computes $s_i := r_i - cx_i$ and sends the s_i to the verifier. The verifier accepts iff $t = y^c \prod_i g_i^{s_i}$.

For the remaining part, i.e., $\mathbf{c}_{m,2} = m + [\![h^{r_e}]\!]$, we give a Σ-protocol following the idea of the one above. This is done by computing r_i and s_i as in [16], however, the t is computed and verified differently; this is done by computing t as $r_m + [\![h^{r_e}]\!]$ and verifying it to be $c \cdot \mathbf{c}_{m,2} + s_m + [\![h^{s_e}]\!]$. Given this construction it is straightforward to prove that the corresponding protocol is indeed a Σ-protocol.

Combining these two Σ-protocols we get a Σ-protocol for ZK-PoE.

C.3 Zero-knowledge Proof Scheme PoCM

As for the previous proof scheme, we will use a Σ-protocol in order to construct an interactive ZK proof scheme. The overall statement that we show is the following.

$$\begin{aligned}\exists u, \{r_j, k_j\}_{j\in[1,n]} : \mathsf{Open}(u, \mathsf{Com}_u) = 1 \wedge \\ \mathbf{c}_{u\cdot v+r} = u \boxdot \mathbf{c}_v \boxplus \mathsf{Enc}_{\mathsf{pk}_i}(r) \wedge \\ \mathsf{Com}_{u\cdot v_j+r_j} = u \odot \mathsf{Com}_{v_j} \oplus \mathsf{Com}(r_j, k_j) \wedge \\ \mathsf{degree}(\{r_j\}_{j\in[1,n]}, t) \wedge \\ \mathsf{degree}(\{k_j\}_{j\in[1,n]}, t) \wedge r_0 = 0 = k_0\end{aligned}$$

The check for the degree can be done using the representation problem, e.g., in order to check that variables x and y satisfy the equation $2x + 5y = 1$ we can check that we know the representation of g^1 with respect to the base $\{g^2, g^5\}$. Consequently, we can check the degree by evaluating the (inverse) Vandermonde matrix on the quantified values and comparing the result with constants.

However, in order to give an instantiation with respect to the previously defined instantiations of Com and Enc we need to existentially quantify over the corresponding randomnesses as well. In addition, we split the proof into two statements which can be combined into a proof for the conjunction using standard techniques, i.e., using the same r_x, s_x for shared variables x.

The first relation can be proven using results described in [16]. We want to prove the knowledge of a representation of the commitments — with respect to the base of the Pedersen commitments — Com_u and Com_r together with an additional verification step to verify the property we require for $\mathsf{Com}_{u\cdot v+r}$. This additional check can be rephrased as follows:

$$X := \mathsf{Com}_{u\cdot v+r} \ominus \mathsf{Com}_r = (g^v h^{r_v})^u$$

Hence, we need to verify that we know the representation of X with respect to the base Com_v (and that this is the same as for the representation of Com_u). Therefore we can simply use the proof scheme [16], as for ZK-PoE, for the first part of the scheme.

For the opening part of the second proof scheme, we can use the same technique. However, it is not obvious that we can use this technique in order to verify $\mathbf{c}_{u\cdot v+r} = u \boxdot \mathbf{c}_v \boxplus \mathsf{Enc}_{\mathsf{pk}_i}(r)$. Looking at the first component of our instantiation of Enc leads to an equation of the form $X = Y^u \cdot g^{R_e}$ which corresponds to the representation knowledge, since the X is represented via the base $\{Y, g\}$ using u and R_e. Hence we can use the same technique. We need to show the correspondence for the second part as well. This part looks like

$$X = uY + R + R_e \cdot Z$$

This is a linear version of the base representation problem and the technique can be applied here as well. The corresponding t of this equation is $r_u \odot \mathbf{c}_v \boxplus \mathsf{Enc}_{\mathsf{pk}_i}(r_R, r_e)$.

C.4 Asynchronous Reliable Broadcast Using Non-equivocation

Let $\mathsf{Sen} \in \mathcal{P}$ be a party with message m, which it wants to send identically to all the n parties; then the protocol r-broadcast allows it to do the same. The high-level idea of the protocol is very simple: Sen first non-equivocally sends m to all the parties; this prevents a corrupted Sen from sending different messages to different honest parties. However, a corrupted Sen may not send the message to *all* the honest parties. So to ensure that all the honest parties eventually obtain the message, we add an additional "round" of communication. Namely if an honest party non-equivocally receives some message from Sen, then it non-equivocally transfers the same on behalf of Sen to every other party. This ensures that if any honest party has received the message from Sen then it will be eventually transferred to every other honest party. Moreover, we also ensure that if some message is non-equivocally transferred (on the behalf of Sen) to some honest party P_i from another party P_j, then P_i futher non-equivocally transfers m to every other party on the behalf of Sen; this is because it may be possible that both P_j and Sen are corrupted and so we need to ensure that the message received by P_i is finally available to everyone, as P_j and Sen may not send them to everyone. Protocol r-broadcast is presented in Fig. 4.

Protocol r-broadcast

CODE FOR THE SENDER Sen (WITH INPUT m) — The following code is executed only by Sen:

1. Non-equivocally send m to every party in $\mathcal{P}$.

CODE FOR THE PARTY P_i — every party in $\mathcal{P}$, including Sen, executes this code:

1. If m is non-equivocally received from Sen, then non-equivocally transfer m to every party in $\mathcal{P}$ on behalf of Sen, output m and terminate.
2. If m is non-equivocally transferred from some party $P_j \in \mathcal{P}$ on behalf of Sen, then non-equivocally transfer m to every party in $\mathcal{P}$ on behalf of Sen, output m and terminate.

Figure 4: Asynchronous reliable broadcast protocol using non-equivocation tolerating $t < n$ corruptions.

The properties of the protocol are stated in Theorem C.4, which follows easily from the protocol description and the properties of non-equivocation.

Theorem C.4. *Protocol* r-broadcast *achieves the following for every possible $\mathcal{A}$ and scheduler:*

(1) TERMINATION: *If* Sen *is honest, then all the honest parties eventually terminate the protocol. Moreover, even if* Sen *is corrupted and some honest party terminates the protocol, then except with negligible probability, every other honest party eventually terminates the protocol* . (2) CORRECTNESS: **(a)** *If* Sen *is*

honest then except with negligible probability, all honest parties output m. **(b)** *If* Sen *is corrupted and some honest party outputs m', then except with negligible probability, all the honest parties output m'.* (3) COMMUNICATION COMPLEXITY: *The protocol incurs communication of $\mathcal{O}(n^2(\ell+\kappa))$ bits, where the message m is of size ℓ bits.*

D Properties of the Various Supervised Sharing Protocols and Proofs

D.1 Properties of the Protocol Sup-Sh

Lemma D.1. *Let s be the* D*'s secret. Then for every possible $\mathcal{A}$ and scheduler, protocol* Sup-Sh *achieves the following properties:* (1) TERMINATION*: if* D *and P_{king} are honest then all the honest parties eventually terminate the protocol, except with negligible probability. Moreover, if some honest party terminates the protocol, then every other honest party eventually does the same, except with negligible probability.* (2) CORRECTNESS*: if some honest party terminates the protocol, then there exists a value $\overline{s}$ which will eventually be $[\cdot]$-shared among the parties, except with negligible probability. Moreover, if* D *is honest then $\overline{s} = s$. Furthermore if P_{king} is honest then P_{king} will be a privileged party.* (3) PRIVACY*: if* D *is honest then s remains private.* (4) COMMUNICATION COMPLEXITY*: the protocol has communication complexity $\mathcal{O}(n^2\kappa)$ bits.*

PROOF: For TERMINATION, we first consider an honest D and P_{king}. In this case, D will non-equivocally send $\{\mathbf{c}_{s_j}, \mathsf{Com}_{s_j}\}, \mathsf{Com}_s$ to all parties. In particular all honest parties will eventually receive them and start participating in the instances of ZK-PoE, where all the verifications will pass. So P_{king} will eventually broadcast the $(\mathtt{OK}, \mathsf{D})$ message, which from the properties of r-broadcast will eventually reach every honest party with high probability. Moreover, since there exist at least $n - t = t + 1$ honest parties, D will be able to construct the certificate α^{D} and eventually broadcast the same. Therefore every honest party eventually receives α^{D} as well as the $(\mathtt{OK}, \mathsf{D})$ message. Moreover, P_{king} will be a privileged party and non-equivocally transfers $\mathbf{c}_{c_j}$ and Com_{s_j} to every P_j. It now follows easily that every honest P_j will eventually receive $\mathbf{c}_{s_j}$ and Com_{s_j} from P_{king} and obtain its share s_j by decrypting $\mathbf{c}_{s_j}$. Moreover, since every such P_j non-equivocally transfers its Com_{s_j} to every other party and there are at least $t+1$ such P_js, it follows that every honest party will eventually have at least $t + 1$ committed shares with high probability, using which it will homomorphically obtain the remaining committed shares and terminate.

Now consider a corrupted D (and possibly a corrupted P_{king}) and let P_i be an honest party that terminates the protocol. We show that all other honest parties will eventually do the same. Since P_i terminated the protocol, it implies that P_i received α^{D} from the broadcast of D as well as $(\mathtt{OK}, \mathsf{D})$ from P_{king}'s broadcast. From the properties of broadcast, it follows that with high probability, every other honest party will eventually receive them. In addition, since α^{D} was constructed, at least $t + 1$ and hence at least one honest party, say P_h, must have received $\{\mathbf{c}_{s_j}, \mathsf{Com}_{s_j}\}, \mathsf{Com}_s$ from D and successfully performed all the verifications. Since P_h is a privileged party, the rest of the proof follows using the same arguments as above, except that P_h plays the role of P_{king}.

CORRECTNESS: If some honest party, say P_i, has terminated the protocol, then it follows that it has received a valid certificate α^{D} from the broadcast of D, which implies that with high probability, there exists at least one honest party, say P_h, who would have participated in the construction of α^{D}. This further implies that P_h must have non-equivocally received $\{\mathbf{c}_{s_j}, \mathsf{Com}_{s_j}\}_{j\in[1,n]}$ and Com_s from D and successfully performed the required verifications. Particularly, P_h would have verified that there exists some polynomial of degree at most t, say $\overline{f}(\cdot)$, such that $\mathsf{Com}_s = \mathsf{Commit}(\overline{f}(0))$ and $\mathsf{Com}_{s_j} = \mathsf{Commit}(\overline{f}(j))$. We define $\overline{s}$ to be $\overline{f}(0)$ and show that eventually $\overline{s}$ will be $[\cdot]$-shared. Since P_i has terminated the protocol, from the termination property of the protocol, it follows that each honest party will eventually terminate with its share s_j

and a vector of committed shares, so what remains is to show that they correspond to $[\overline{f}(0)]$. However, this follows from the properties of non-equivocation. Specifically, neither a corrupted D nor any corrupted party can send or transfer any other encrypted and committed share, different from $\mathbf{c}_{s_j}$ and Com_{s_j} respectively, to any honest P_j. Similarly, no corrupted party P_k can transfer its committed share, different from Com_{s_k}, to any honest party. Thus with high probability, $\overline{s}$ will be $[\cdot]$-shared.

It follows easily that if D is honest then $\overline{s} = s$, as in this case the polynomial $\overline{f}(\cdot)$ is the same as $f(\cdot)$, as selected by D. Moreover it follows easily that if P_{king} is honest then it will be a privilged party, since an honest P_{king} will broadcast the $(\mathtt{OK}, \mathsf{D})$ message only after non-equivocally receiving $\{\mathbf{c}_{s_j}, \mathsf{Com}_{s_j}\}, \mathsf{Com}_s$ from D and successfully verifying it.

PRIVACY: We show that for an honest dealer D, and any $s, \overline{s}$ the adversary can not distinguish whether D shared s or $\overline{s}$. So let $\mathcal{T}_{cor}$ be the set of corrupted parties. Hence define $\mathcal{K}_{corr} := \{s_i, k_i \mid P_i \in \mathcal{T}_{corr},$ where s_i is the share of party P_i and k_i its encryption and decryption keys$\} \cup \{sk_i \mid sk_i$ is the signing key of $P_i\}$. Note that we only assumed authentic channels; therefore, it is easy to see that the view of the adversary $\mathsf{view}_A(x)$ consists of $\mathsf{Com}_x, \{\mathsf{Com}_{x_j}, \mathbf{c}_{x_j}\}_{j\in[1,n]}$ as well as $\alpha^{\mathsf{D}}, (\mathtt{OK}, \mathsf{D})$ and the messages during the protocol executions of ZK-PoE.

Assume there is an adversary A that can distinguish whether $x = s$ or $x = \overline{s}$ is shared with non-negligible probability. We then show that there is an adversary that can distinguish Com_s from $\mathsf{Com}_{\overline{s}}$ with non-negligible probability. We do this in several steps; in particular, we define the following views and show that these are indistinguishable for s and $\overline{s}$.

- $\mathsf{view}^1_A(x) := \mathsf{view}_A(x) \cup \mathcal{K}_{corr}$
- $\mathsf{view}^2_A(x) := \mathsf{Com}_x, \{\mathsf{Com}_{x_j}, \mathbf{c}_{x_j}\}_{j\in[1,n]}$ and $\mathcal{K}_{corr}$
- $\mathsf{view}^3_A(x) := \mathsf{Com}_x, \{\mathsf{Com}_{x_j}\}_{j\in[1,n]}$ and $\mathcal{K}_{corr}$
- $\mathsf{view}^4_A(x) := \mathsf{Com}_x$ and $\mathcal{K}_{corr}$
- $\mathsf{view}^5_A(x) := \mathcal{K}_{corr}$

Next, we show for $i \in [1, 4]$ that $\mathsf{view}^i_A(x) \sim \mathsf{view}^{i+1}_A(x)$. For each step we need to show that if for each adversary A^i having input $\mathsf{view}^i_A(x)$, there is an adversary A^{i+1} which has an indistinguishable output on input $\mathsf{view}^{i+1}_A(x)$.[6]

1. $\mathsf{view}^1_A(x) \sim \mathsf{view}^2_A(x)$

   Let A^1 be given, A^2 internally uses A^1 by computing $(\mathtt{OK}, \mathsf{D})$, computing α^{D} and simulating the zero-knowledge proofs ZK-PoE. The first part is trivial, the second part can be done since the message signed in α can be deduced from $\mathsf{view}^2_A(x)$ and A^2 has access to all signing key shares. In order to prove the last part we need to distinguish two cases: ZK-PoE executions with honest parties and ZK-PoE executions with corrupted parties. The executions with honest parties can be computed by A^2 using the simulator of the honest-verifier zero-knowledge property, since the honest parties choose their challenge randomly. For the corrupted parties, the adversary A^2 knows their s_i. Consequently, he can act as the dealer in ZK-PoE and use the adversary of the protocol execution in order to generate these proofs.

   Finally, we need to show that the output of A^1 is indistinguishable from the output of A^2. By construction of A^2 it is sufficient to show that the input given to A^1 inside A^2 is indistinguishable from the input that A^1 gets. For $(\mathtt{OK}, \mathsf{D})$ and α this is obvious. For the ZK proofs of honest parties, this

[6] Note that the adversary even knows the signing keys of the parties.

is implied by the honest-verifier zero-knowledge. The zero-knowledge proofs of the corrupted parties consist of messages (a, c, e). Here a has the same distribution as in $\mathsf{view}_A^1(x)$, i.e., uniformly at random. The part c has the same distribution since A^2 internally invokes the adversary of the protocol execution. Finally, e is completely determined by a and c. Therefore e has the same distribution as well.

2. $\mathsf{view}_A^2(x) \sim \mathsf{view}_A^3(x)$

   In this step we basically remove the ciphertexts from the input of A^2. We construct A^3 by internally running A^2 on $\mathsf{view}^3(x)$ and the ciphertexts computed by A^3. In order to compute the ciphertexts A^3 needs to distinguish two cases, ciphertexts of corrupted and ciphtertexts of honest parties. For corrupted parties, A^3 can simply access the plaintexts using $\mathcal{K}_{corr}$ and encrypt them as D does, hence having indistinguishablility. The ciphertexts of honest parties are replaced by encryptions of 0s. By the IND-CPA property, it follows that this ciphertext is indistinguishable from the original message's ciphertext. Therefore the input to A^2 is indistinguishable to $\mathsf{view}^2(x)$ and consequently its output as well.

3. $\mathsf{view}_A^3(x) \sim \mathsf{view}_A^4(x)$

   In this step we remove all commitments except the commitment to x. The adversary $A^4(x)$ can compute the set $\{\mathsf{Com}_{x_j}\}$ for the corrupted parties P_j by recomputing them. For the other commitments, the adversary A^4 can interpolate the polynomial inside the commitments; since he has t values Com_{x_j} and the value Com_x this leads to a unique polynomial inside the commitments, i.e., $\{\mathsf{Com}_{x_j}\}_{j\in[1,n]}$. Then A^4 invokes A^3 on the computed input.

4. $\mathsf{view}_A^4(x) \sim \mathsf{view}_A^5(x)$

   Finally we want to remove the commitment Com_x. Since there are exactly t shares s_i in the adversaries' knowledge, any x can be used in order to determine a unique polynomial. By the computational hiding property, committing to any random value using uniform randomness cannot be distinguished from Com_x. Therefore $A^5(x)$ computes such a commitment and runs A^4 on this input.

We can conclude that $\mathsf{view}_A(s) \sim \mathsf{view}_A^5(s)$ and $\mathsf{view}_A(\overline{s}) \sim \mathsf{view}_A^5(\overline{s})$. Since we assume that $\mathsf{view}_A(s)$ is distinguishable from $\mathsf{view}_A(\overline{s})$, we can conclude that $\mathcal{K}_{corr}(s) = \mathsf{view}_A^4(s)$ is distinguishable from $\mathcal{K}_{corr}(\overline{t}) = \mathsf{view}_A^4(\overline{t})$. However, both $\mathcal{K}_{corr}(s)$ and $\mathcal{K}_{corr}(\overline{s})$ consists of the adversaries keys and — since D is honest — t values which are uniformly random. Therefore they cannot be distinguished (a contradiction). Hence the assumption has to be wrong and *privacy* follows by the contraposition.

COMMUNICATION COMPLEXITY: During the D-DEPENDENT PHASE, D has to non-equivocally distribute $\mathcal{O}(n)$ encryted shares and committed shares to every party, which costs $\mathcal{O}(n^2\kappa)$ bits. Construction of the certificate α^{D} requires $\mathcal{O}(n^2\kappa)$ bits of communication, as there are n ecnrypted and committed shares and so D needs to execute in total n^2 instances of ZK-PoE. Broadcasting α^{D} costs $\mathcal{O}(n^2\kappa)$ bits of communication, as the certificate is of size $\mathcal{O}(\kappa)$ bits. During the D-INDEPENDENT PHASE, each party just needs to send one encrypted share and one committed share to every other party, incurring a communication of $\mathcal{O}(n^2\kappa)$ bits. □

D.2 Properties of the Protocol Sup-PreMul-Sh

Lemma D.2. *Let v be a completely random and unknown value which is $[\cdot]$-shared among $\mathcal{P}$ and let u be a value selected by D. Then for every possible $\mathcal{A}$ and scheduler, protocol* Sup-PreMul-Sh *achieves the following (properties (1) and (2) up to a negligible error probability):* (1) TERMINATION*: if D and P_{king}*

are honest then all the honest parties eventually terminate the protocol. Moreover, if some honest party terminates the protocol, then every other honest party eventually does the same. (2) CORRECTNESS*: if some honest party terminates the protocol, then there exists a value $\overline{u}$, such that $\overline{u}$ and $\overline{u} \cdot v$ will eventually be $[\cdot]$-shared among the parties. If* D *is honest then $\overline{u} = u$. Moreover if P_{king} is honest then P_{king} will be a privileged party with respect to $[\overline{u}]$ as well as $[\overline{u} \cdot v]$.* (3) PRIVACY*: v and $u \cdot v$ remains private at the end of the protocol. Additionally, if* D *is honest then u also remains private.* (4) COMMUNICATION COMPLEXITY*: the protocol has communication complexity $\mathcal{O}(n^2\kappa)$ bits.*

Proof.

1. COMMUNICATION COMPLEXITY: The *generating* phase of the protocol consists of one instantce of the Sup-Sh protocol which has communication complexity $\mathcal{O}(n^2\kappa)$. The D independent phase of the protocol is similar as in the Sup-Sh protocol and hence has communication complexity $\mathcal{O}(n^2\kappa)$. The same holds for the *share verification and certification* part of the protocol, a broadcast from the king with complexity $\mathcal{O}(n^2\kappa)$ complexity and n^2 instances of PoCM (n parties running n instances each). Hence this part has communication complexity $\mathcal{O}(n^2\kappa)$ as well. Finally, in the *share communication and certificate generation* part, the certificate β is broadcasted by the dealer and the dealer takes part in all executions of PoCM. In addition the dealer sends $3 + 2n$ commitments and n ciphertexts non-equivocally to every party. The broadcast and PoCM executions have communication complexity $\mathcal{O}(n^2\kappa)$ and the same holds for sending $\mathcal{O}(n)$ values of size $\mathcal{O}(\kappa)$ non-equivocally to every party. Consequently the overall protocol has complexity $\mathcal{O}(n^2\kappa)$.

2. TERMINATION: Termination of the *generating* $\langle u \rangle$ phase follows by the corresponding properties of the Sup-Sh protocol. For the remaining phases of the protocol the termination properties follow completely analogously to the corresponding properties of Sup-Sh since the protocol structure is essentially the same.

3. CORRECTNESS: For correctness we have to show that in the end there is an t-sharing $\bar{u}$ and $\bar{u}v$. We have to show that for an honest D it holds that $\bar{u} = u$ and that if the king is honest, he is priviledged with respect to $[u]$ and $[u \cdot v]$.

   The correctness of the protocol Sup-Sh already implies that there is a t-sharing of $\bar{u}$ and that an honest D will share $u = \bar{u}$. In addition, the correctness of Sup-Sh implies that P_{king} is privileged with respect to $[u]$.

   Therefore we only need to show that upon termination, there is a t-sharing of $\bar{u} \cdot v$ and that an honest P_{king} is privileged with respect to this sharing. Since an honest P_{king} only broadcasts $(approve, \mathsf{D})$ when he has received $\mathsf{Com}_u, \{\mathbf{c}_{u \cdot v_j + r_j}\}_{j \in [1,n]}, \{\mathsf{Com}_{u \cdot v_j + r_j}\}_{j \in [1,n]}$ and $\mathsf{Com}_{u \cdot v}$, as a subset of this message received all necessary information to be priviledged with respect to $\bar{u} \cdot v$. Finally, upon termination, every party P_i received $\mathbf{c}_{\bar{u} \cdot v_i + r_i}$ as well as the corresponding commitment. For the same reason as in the proof of protocol Sup-Sh, these values correspond to the verified shares, i.e., belong to a degree t polynomial, and by the nonequivocation property, $\bar{u} = u$ is ensured.

4. PRIVACY: The privacy proof is threefold. First, we have to show that v remains private, i.e., communication during the protocol does not help in distinguishing the value of two different v. Second, we have to show that $u \cdot v$ remains private in the same sense and third, we have to show that u is private if the dealer D is honest.

   (a) *v remains private:* Assume an adversary A can distinguish v from v' after seeing the additional information of the Sup-PreMul-Sh execution for some u. Since the execution requires that $[v]$ (or $[v']$) is already shared, it follows that an adversary $\mathcal{B}_A$ can internally simulate an execution of Sup-PreMul-Sh using u and then invoke A in order to distinguish v from v'. Hence v an adversary does not gain any additional information about v by the execution of Sup-PreMul-Sh.

(b) $u \cdot v$ *remains private:* Assume an adversary A can distinguish $w = u \cdot v$ from $w' = u' \cdot v'$. Since D may be corrupted, D can choose $u = u' = 1$. As a consequence A can now distinguish the cases $v = w$ from $v' = w'$ contradicting the privacy of v. Hence $u \cdot v$ remains private as well.

(c) u *remains private if* D *is honest:* This case is more difficult than the other two cases since the protocol leaks more information about u than the protocol Sup-Sh executed on u. However, we can follow the privacy proof of Sup-Sh.

Assume there is an adversary A that distinguishes the protocol execution for some u and u'. In addition to the execution of Sup-Sh on u, the adversary gets $\{\mathbf{c}_{u \cdot v_j + r_j}, \mathsf{Com}_{u \cdot v_j + r_j}\}_{j \in [1,n]}, \mathsf{Com}_{u \cdot v}$, the executions of PoCM and $(approve, \mathsf{D})$ as well as β^{D}. As in the proof of privacy with respect to the protocol Sup-Sh, the information $(approve, \mathsf{D})$ and β^{D} does not help the adversary in distinguishing u from u'.

Since $\mathsf{Com}_{u \cdot v}$ can be computed from $\mathsf{Com}_{u \cdot v_j + r_j}$, we know there is an adversary that distinguishes u from u' without the input $\mathsf{Com}_{u \cdot v}$. Using the zero-knowledge property we can also simulate the proofs leading to an indistinguishable outcome of A (by definiton of zero-knowledge). As a consequence there is an adversary that distinguishes u from u' by seeing the output of Sup-Sh, $\mathbf{c}_{u \cdot v_j + r_j}, \mathsf{Com}_{u \cdot v_j + r_j}$. Since the dealer D is honest, we can use the IND-CPA property to remove the ciphertexts $\mathbf{c}_{u \cdot v_j + r_j}$ as we did in the privacy proof for Sup-Sh. Also following the privacy proof for Sup-Sh, we can reduce the input of the adversary to $\mathsf{Com}_{u \cdot v}$, which finally contradicts the computational hiding property of our commitment scheme. Consequently, there is no such adversary A.

□

E Protocol for Supervised Triple Generation and its Properties

Here we present our supervised triple-generation protocol; we first present the subprotocols used.

E.1 Protocol Sup-Second and Its Properties

Protocol Sup-Second (for generating the second component of the shared multiplication triple) is presented in Fig. 5.

The properties of the protocol Sup-Second are stated in Lemma E.1.

Lemma E.1. *For every possible $\mathcal{A}$ and every possible scheduler, the protocol* Sup-Second *achieves the following properties (properties (1) and (2) up to a negligible error probability):* (1) TERMINATION: *if P_{king} is honest, then all honest parties eventually terminate the protocol. Moreover, even if P_{king} is corrupted and some honest party terminates the protocol, then every other honest party eventually does the same.* (2) CORRECTNESS: *if the honest parties terminate the protocol, then the parties output $[\cdot]$-sharing $[v]$ of a value v. Moreover, each party will be a privileged party having all the encrypted shares of v.* (3) PRIVACY: *the output shared value will be random from the viewpoint of $\mathcal{A}$.* (4) COMMUNICATION COMPLEXITY: *the protocol has communication complexity $\mathcal{O}(n^3\kappa)$ bits.*

Proof. 1. TERMINATION: in order to show termination, we need to show two properties; first, if P_{king} is honest, then every honest party eventually terminates and second, if any honest party terminates, all other honest parties will do the same.

(a) *Honest king leads to termination:* if the king is honest, then the set $\mathcal{T}_{\mathsf{king}}$ will eventually reach the size $t + 1$, because of Theorem D.1 and since there are $t + 1$ honest dealers in the executions of

Protocol $\mathsf{Sup\text{-}Second}(P_{\mathsf{king}})$

i. SHARING RANDOM VALUES—Every party $P_i \in \mathcal{P}$ including P_{king} executes the following code:

1. Select a random value $v^{(i)}$ and invoke an instance of $\mathsf{Sup\text{-}Sh}$ as a D to generate $[v^{(i)}]$ under the supervision of P_{king}; let this instance of $\mathsf{Sup\text{-}Sh}$ be denoted as $\mathsf{Sup\text{-}Sh}_i$. Moreover, let $f_i(\cdot), \{\mathbf{c}_{v_{i,j}}\}_{j\in[1,n]}, \{\mathsf{Com}_{v_{i,j}}\}_{j\in[1,n]}$ and $\mathsf{Com}_{v^{(i)}}$ denote the sharing polynomial, encrypted shares, committed shares and commitment, generated during $\mathsf{Sup\text{-}Sh}_i$, where $v_{i,j} = f_i(j)$ is the jth share of $v^{(i)}$ and $\mathbf{c}_{v_{i,j}} = \mathsf{Enc}_{\mathsf{pk}_j}(v_{i,j}), \mathsf{Com}_{v_{i,j}} = \mathsf{Commit}(v_{i,j})$ and $\mathsf{Com}_{v^{(i)}} = \mathsf{Commit}(v^{(i)})$.
2. For $j \in [1, n]$, participate in the instance $\mathsf{Sup\text{-}Sh}_j$, invoked by P_j as a D.

ii. COLLECTING AND DISTRIBUTING THE INFORMATION FOR THE FINAL OUTPUT—Only P_{king} executes the following code:

1. Include party P_k in an accumulative set $\mathcal{T}_{\mathsf{king}}$, which is initially $\emptyset$, if the instance $\mathsf{Sup\text{-}Sh}_k$ is (locally) terminated.
2. Wait till $|\mathcal{T}_{\mathsf{king}}| = t+1$. Then using the linearity property of the encryption scheme, compute $\mathbf{c}_{v_j} = \boxplus_{P_k\in\mathcal{T}_{\mathsf{king}}}\mathbf{c}_{v_{k,j}}$ for $j \in [1, n]$ and broadcast $\mathcal{T}_{\mathsf{king}}, \{\mathbf{c}_{v_j}\}_{j\in[1,n]}$.
3. For every $P_k \in \mathcal{T}_{\mathsf{king}}$, non-equivocally transfer the encrypted shares $\{\mathbf{c}_{v_{k,j}}\}_{j\in[1,n]}$ (received as a privileged party from the dealer P_k during the instance $\mathsf{Sup\text{-}Sh}_k$) to every party in $\mathcal{P}$ on behalf of P_k.

iii. RESPONDING TO P_{king} AND TERMINATION—Every party $P_i \in \mathcal{P}$ including P_{king} executes the following code:

1. Include party P_k in an accumulative set $\mathcal{T}_i$, which is initially $\emptyset$, if the instance $\mathsf{Sup\text{-}Sh}_k$ is (locally) terminated.
2. Wait to receive $\mathcal{T}_{\mathsf{king}}$ and $\{\mathbf{c}_{v_j}\}_{j\in[1,n]}$ from the broadcast of P_{king}.
3. If $\{\mathbf{c}_{v_{k,j}}\}_{j\in[1,n]}$ is non-equivocally transferred by P_{king} on behalf of each $P_k \in \mathcal{T}_{\mathsf{king}}$, then wait till $\mathcal{T}_{\mathsf{king}} \subseteq \mathcal{T}_i$. Once $\mathcal{T}_{\mathsf{king}} \subseteq \mathcal{T}_i$, broadcast the message $(\mathtt{OK}, i)$ only if $\mathbf{c}_{v_j} = \boxplus_{P_k\in\mathcal{T}_{\mathsf{king}}}\mathbf{c}_{v_{k,j}}$ holds for every $j \in [1, n]$.
4. Wait to receive the $(\mathtt{OK}, \star)$ message from the broadcast of at least $t+1$ parties. On receiving, wait till $\mathcal{T}_{\mathsf{king}} \subseteq \mathcal{T}_i$ and then compute $v_i = \sum_{P_k\in\mathcal{T}_{\mathsf{king}}} v_{k,i}, \{\mathsf{Com}_{v_j} = \oplus_{P_k\in\mathcal{T}_{\mathsf{king}}}\mathsf{Com}_{v_{k,j}}\}_{j\in[1,n]}$ and $\mathsf{Com}_v = \oplus_{P_k\in\mathcal{T}_{\mathsf{king}}}\mathsf{Com}_{v^{(k)}}$, where $v_{k,i}$ denotes the share obtained at the end of the instance $\mathsf{Sup\text{-}Sh}_k$ and $\{\mathsf{Com}_{v_{k,j}}\}_{j\in[1,n]}$ and $\mathsf{Com}_{v^{(k)}}$ denotes the vector of committed shares and the commitment obtained at the end of the instance $\mathsf{Sup\text{-}Sh}_k$. Finally, output $v_i, \{\mathbf{c}_{v_j}\}_{j\in[1,n]}, \{\mathsf{Com}_{v_j}\}_{j\in[1,n]}$ and Com_v and terminate.

Figure 5: **Supervised generation of $[v]$ for a random v under the supervision of P_{king}; if the protocol terminates then each party will be a privileged party and will have all n encrypted shares of v.**

$\mathsf{Sup\text{-}Sh}$. In particular, since the honest party P_{king} terminates for all executions corresponding to $\mathcal{T}_{\mathsf{king}}$, all honest P_i will eventually terminate for the same instances by the termination property of $\mathsf{Sup\text{-}Sh}$. Since P_{king} is honest, the checks done by the honest parties in the *response and termination* phase will succeed and they will broadcast $(\mathtt{OK}, \star)$. Thus, eventually the number of received $(\mathtt{OK}, \star)$ broadcasts will reach $t+1$ and the honest parties terminate.

(b) *If an honest party terminates, then all honest parties do so:* let P_i be the honest party that terminates. We show that if a party P_j is honest, then P_j eventually terminates. The set $\mathcal{T}_i$ contains all parties for which P_i terminated the corresponding $\mathsf{Sup\text{-}Sh}$ when P_i terminated. By termination of the $\mathsf{Sup\text{-}Sh}$ protocol, it follows that eventually $\mathcal{T}_i \subseteq \mathcal{T}_j$. Since P_{king} broadcasts $\mathcal{T}_{\mathsf{king}}$ all parties will eventually receive the same $\mathcal{T}_{\mathsf{king}}$ and the condition $\mathcal{T}_{\mathsf{king}} \subseteq \mathcal{T}_j$ will eventually be satisfied. In addition, since P_i terminated, it received at least $t+1$ broadcasted messages $(\mathtt{OK}, \star)$ which will — since these messages were broadcast — eventually arrive at P_j. Consequently, this condition is satisfied as well. Thus P_j finally terminates.

2. CORRECTNESS: At the end of the protocol execution every party outputs $v_i, \{\mathbf{c}_{v_j}, \mathsf{Com}_{v_j}\}_{j\in[1,n]}$ and Com_v. There is an honest party that verifies that this message is a linear combination of the sharings run before. Since these precomputed sharings follow the correct protocol $\mathsf{Sup\text{-}Sh}$ and since sharings are linear, it follows that the parties hold a sharing of some value v when terminating.

3. PRIVACY: For all honest parties it holds that their sharing is indistinguishable for any value they shared, by the privacy of $\mathsf{Sup\text{-}Sh}$. Since the overall output is a linear combination of $t+1$ sharings, at least one honest sharing is contained in this combination. Assuming the adversary A could distinguish

this for any two different values v, v', then the adversary could use this to break the privacy of Sup-Sh, since he can control the t other parties from the output sharing. Thus, the privacy of Sup-Sh implies the privacy of Sup-Second.

Moreover, the honest party of which a share is contained in the linear combination chooses this share uniformly at random. Consequently, the linear combination contains a value that is uniformly random as well.

4. COMMUNICATION COMPLEXITY: in the *Sharing random values* part of the protocol, there are n instances of the Sup-Sh protocol, i.e., a communication complexity of $\mathcal{O}(n^3\kappa)$.

   The *collection and distribution of the information* is done only by the party P_{king}. During the execution of this part of the protocol, the king broadcasts $\mathcal{T}_{\mathsf{king}}$ and $\{\mathbf{c}_{v_j}\}_{j\in[1,n]}$ and non-equivocally transfers $\{\mathbf{c}_{v_{k,j}}\}_{j\in[1,n]}$ for all $P_k \in \mathcal{T}_{\mathsf{king}}$. The broadcast message has size $\mathcal{O}(n\kappa)$ leading to a communication complexity of $\mathcal{O}(n^3\kappa)$ and the non-equivocally transfered data has size $\mathcal{O}(n^2\kappa)$ leading to communication complexity $\mathcal{O}(n^3\kappa)$ as well.

   The final *response and termination* part which is executed by all parties consists only of broadcasting $(\texttt{OK}, i)$. This message has size $\mathcal{O}(1)$ because the identity is encoded in the broadcast protocol. However, there are n broadcasts in the worst case, leading to a communication complexity of $\mathcal{O}(n^3\kappa)$.

□

E.2 Protocol Sup-FirAndThd and Its Properties

Protocol Sup-FirAndThd (for generating the first and third component of the shared multiplication) is presented in Figure 6.

The properties of the protocol Sup-FirAndThd are presented in Lemma E.2.

Lemma E.2. *For every possible A and every possible scheduler, the protocol* Sup-FirAndThd *achieves the following properties:* (1) TERMINATION: *if P_{king} is honest, then all honest parties eventually terminate. Moreover, if one honest party terminates, then all honest parties eventually terminate.* (2) CORRECTNESS: *after termination, all parties hold a sharing for $[u], [w]$ such that $[w]$ is a sharing of $[u], [v]$.* (3) PRIVACY: *the execution is indistinguishable for different values of u and v.* (4) COMMUNICATION COMPLEXITY: *the protocol has communication complexity $\mathcal{O}(n^3\kappa)$ bits.*

Proof. The proof completely follows the proof of Sup-Second, except that properties are now implied from Sup-Sh, instead of Sup-PreMul-Sh. □

E.3 The Supervised Tripled Generation Protocol SupTripGen

Protocol SupTripGen for the supervised triple generation, which is a combination of Sup-Second and Sup-FirAndThd, is presented in Fig. 7.

The following lemma follows easily from the properties of Sup-Second and Sup-FirAndThd and the protocol steps:

Lemma E.3. *For every possible $\mathcal{A}$ and every possible scheduler, the protocol* SupTripGen *achieves the following properties (properties (1) and (2) up to a negligible error probability):* (1) TERMINATION: *if P_{king} is honest then all honest parties eventually terminate the protocol. Moreover, even if P_{king} is corrupted and some honest party terminates the protocol, then every other honest party eventually does the same.* (2) CORRECTNESS: *if the honest parties terminate the protocol, then the parties output $[\cdot]$-sharing $([u], [v], [w])$ of a multiplication triple (u, v, w).* (3) PRIVACY: *the shared multiplication triple (u, v, w) will be random from the viewpoint of $\mathcal{A}$.* (4) COMMUNICATION COMPLEXITY: *the protocol has communication complexity $\mathcal{O}(n^3\kappa)$ bits.*

Protocol $\mathsf{Sup\text{-}FirAndThd}(P_{\mathsf{king}}, [v])$

i. SHARING RANDOM VALUES—Every party $P_i \in \mathcal{P}$ including P_{king} executes the following code:

1. Select a random value $u^{(i)}$ and invoke an instance of $\mathsf{Sup\text{-}PreMul\text{-}Sh}$ on $[v]$ as a D to generate $[u^{(i)}]$ and $[u^{(i)} \cdot v] = [w^{(i)}]$ under the supervision of P_{king}; let this instance of $\mathsf{Sup\text{-}PreMul\text{-}Sh}$ be denoted as $\mathsf{Sup\text{-}PreMul\text{-}Sh}_i$. Moreover, let $\{u_{i,j}\}_{j\in[1,n]}, \{\mathbf{c}_{u_{i,j}}\}_{j\in[1,n]}, \{\mathsf{Com}_{u_{i,j}}\}_{j\in[1,n]}$ and $\mathsf{Com}_{u^{(i)}}$ denote the vector of shares, vector of encrypted shares, vector of committed shares and the commitment corresponding to $[u^{(i)}]$ generated during $\mathsf{Sup\text{-}PreMul\text{-}Sh}_i$. Similarly, let $\{w_{i,j}\}_{j\in[1,n]}, \{\mathbf{c}_{w_{i,j}}\}_{j\in[1,n]}, \{\mathsf{Com}_{w_{i,j}}\}_{j\in[1,n]}$ and $\mathsf{Com}_{w^{(i)}}$ denote the vector of shares, vector of encrypted shares, vector of committed shares and the commitment corresponding to $[w^{(i)}]$ generated during $\mathsf{Sup\text{-}PreMul\text{-}Sh}_i$.
2. For $j \in [1, n]$, participate in the instance $\mathsf{Sup\text{-}PreMul\text{-}Sh}j$, invoked by P_j as a D.

ii. COLLECTING AND DISTRIBUTING THE INFORMATION FOR THE FINAL OUTPUT—Only P_{king} executes the following code:

1. Include party P_k in an accumulative set $\mathcal{T}_{\mathsf{king}}$, which is initially $\emptyset$, if the instance $\mathsf{Sup\text{-}Sh}_k$ is (locally) terminated.
2. Wait till $|\mathcal{T}_{\mathsf{king}}| = t + 1$. Then broadcast $\mathcal{T}_{\mathsf{king}}$.

iii. RESPONDING TO P_{king} AND TERMINATION—Every party $P_i \in \mathcal{P}$ including P_{king} executes the following code:

1. Include party P_k in an accumulative set $\mathcal{T}_i$, which is initially $\emptyset$, if the instance $\mathsf{Sup\text{-}Sh}_k$ is (locally) terminated.
2. Wait to receive $\mathcal{T}_{\mathsf{king}}$ from the broadcast of P_{king}.
3. On receiving $\mathcal{T}_{\mathsf{king}}$, check if it is of size $t + 1$ and if so then wait till $\mathcal{T}_{\mathsf{king}} \subseteq \mathcal{T}_i$.
4. Compute $u_i = \sum_{P_k \in \mathcal{T}_{\mathsf{king}}} u_{k,i}, \{\mathsf{Com}_{u_j} = \oplus_{P_k\in\mathcal{T}_{\mathsf{king}}}\mathsf{Com}_{u_{k,j}}\}_{j\in[1,n]}, \mathsf{Com}_u = \oplus_{P_k\in\mathcal{T}_{\mathsf{king}}}\mathsf{Com}_{u^{(k)}}$, where $u_{k,i}, \{\mathsf{Com}_{u_{k,j}}\}_{j\in[1,n]}$ and $\mathsf{Com}_{u^{(k)}}$ is obtained at the end of $\mathsf{Sup\text{-}PreMul\text{-}Sh}_k$, corresponding to $[u^{(k)}]$. Similarly compute $w_i = \sum_{P_k \in \mathcal{T}_{\mathsf{king}}} w_{k,i}, \{\mathsf{Com}_{w_j} = \oplus_{P_k\in\mathcal{T}_{\mathsf{king}}}\mathsf{Com}_{w_{k,j}}\}_{j\in[1,n]}, \mathsf{Com}_w = \oplus_{P_k\in\mathcal{T}_{\mathsf{king}}}\mathsf{Com}_{w^{(k)}}$, where $w_{k,i}, \{\mathsf{Com}_{w_{k,j}}\}_{j\in[1,n]}$ and $\mathsf{Com}_{w^{(k)}}$ is obtained at the end of $\mathsf{Sup\text{-}PreMul\text{-}Sh}_k$, corresponding to $[w^{(k)}]$. Finally, output $u_i, \{\mathsf{Com}_{u_j}\}_{j\in[1,n]}, \mathsf{Com}_u$ as well as $w_i, \{\mathsf{Com}_{w_j}\}_{j\in[1,n]}, \mathsf{Com}_w$ and terminate.

Figure 6: **Supervised generation of $[u]$ and $[w = u \cdot v]$ for a random u under the supervision of P_{king}, where v is an existing $[\cdot]$-shared value, with every party being a privileged party with respect to $[v]$.**

Protocol $\mathsf{SupTripGen}(P_{\mathsf{king}})$

i. GENERATING THE SECOND COMPONENT OF THE TRIPLE—The parties in $\mathcal{P}$ execute an instance of $\mathsf{Sup\text{-}Second}(P_{\mathsf{king}})$ to generate a uniformly random $[\cdot]$-shared value, say $[v]$.

ii. GENERATING THE FIRST AND THIRD COMPONENT OF THE TRIPLE—On terminating the instance of $\mathsf{Sup\text{-}Second}(P_{\mathsf{king}})$, the parties execute $\mathsf{Sup\text{-}FirAndThd}(P_{\mathsf{king}}, [v])$ to obtain $[u]$ and $[w = u \cdot v]$, output $([u], [v], [w])$ and terminate.

Figure 7: **Supervised generation of a uniformaly random $[\cdot]$-shared multiplication triple, unknown to $\mathcal{A}$, under the supervision of P_{king}.**

F Outline of Our AMPC Protocol

Our AMPC protocol is a sequence of the following three phases:

Preprocessing Phase. To generate $c_M + c_R$ $[\cdot]$-shared random multiplication triples, the preprocessing phase protocol $\mathsf{PreProcess}$ performs the following steps: each party $P_i \in \mathcal{P}$ is asked to act as a king and invoke $\frac{c_M+c_R}{t+1}$ *parallel* instances of $\mathsf{SupTripGen}$ to generate $\frac{c_M+c_R}{t+1}$ random $[\cdot]$-shared multiplication triples under its supervision. The parties then execute an instance of ACS and agree on a common subset $\mathcal{TCORE}$ of $(n - t) = t + 1$ kings whose instances of $\mathsf{SupTripGen}$ (as a king) will eventually be terminated by all the parties. The parties finally output the shared multiplication triples, generated during

the instances of SupTripGen, corresponding to the kings in $\mathcal{TCORE}$ and terminate; thus they will obtain $|\mathcal{TCORE}| \cdot \frac{c_M+c_R}{t+1} = c_M + c_R$ shared multiplication triples. As there exist at least $t+1$ honest parties, whose instances of SupTripGen as a king will be eventually terminated by all the (honest) parties (see Lemma E.3), protocol PreProcess will eventually terminate. Similarly, as the shared triples generated in the instances of SupTripGen corresponding to each king in $\mathcal{TCORE}$ remain private, the output shared triples remain private. It is easy to see that PreProcess has communication complexity $\mathcal{O}(n \cdot \frac{c_M+c_R}{t+1} \cdot n^3\kappa) = \mathcal{O}((c_M + c_R)n^3\kappa)$ bits as $t = \Theta(n)$. As the protocol is quite straightforward, we skip the formal details.

Input Phase. The goal of the input phase protocol Input is to allow each individual party $P_i \in \mathcal{P}$ to generate $[\cdot]$-sharing of its private input x_i for the computation. For this each party $P_i \in \mathcal{P}$ invokes an instance of the sharing protocol Sh (see section. 4.1.1) as D to generate $[x_i]$. To avoid indefinite waiting, the parties execute an instance of ACS and agree on a common subset of $(n - t)$ parties, say $\mathcal{CORE}$, whose instances of Sh (as a dealer) will eventually be terminated by all the parties. The parties finally output the sharings, generated during the instances of Sh, corresponding to the parties in $\mathcal{CORE}$; on behalf of the remaining parties in $\mathcal{P} \setminus \mathcal{CORE}$, a default $[\cdot]$-sharing of 0 is considered. As there exist at least $t + 1$ honest parties, whose instances of Sh as a dealer will eventually be terminated by all the (honest) parties, protocol Input will eventually terminate. The shared inputs generated in the instances of Sh corresponding to the *honest* parties in $\mathcal{CORE}$ remain private due to the privacy property of Sh. The Input protocol runs n instances of Sh, and has communication complexity of $\mathcal{O}(n \cdot n^2\kappa) = \mathcal{O}(n^3\kappa)$ bits. Again as the protocol is quite straight-forward, we skip the formal details.

Computation Phase. The computation phase protocol Compute performs the shared circuit evaluation on a gate-by-gate basis, by maintaining the following *invariant* for each gate of the circuit: given the $[\cdot]$-sharing of the input(s) of a gate, the protocol allows the parties to securely compute the $[\cdot]$-sharing of the output of the gate. The invariant is trivially maintained for the addition (linear) gates in the circuit, thanks to the lineaity property of $[\cdot]$-sharings. For a multiplication gate, the invariant is maintained by applying the Beaver's circuit randomization technique and using a $[\cdot]$-shared multiplication triple from the pre-processing stage (recall from section 2). For a random gate, a $[\cdot]$-shared multiplication triple from the pre-processing stage is considered and the first component of the triple is associated with the random gate. Finally, once the $[\cdot]$-sharing $[y]$ of the circuit output y is generated, the parties execute the reconstruction protocol Rec, reconstruct y and terminate. Again as the protocol is quite standard in the literature (see for example [20]), we omit the complete details.