On Communication-Efficient Asynchronous MPC with Adaptive Security

Annick Chopard, Martin Hirt, and Chen-Da Liu-Zhang[⋆]

{achopard,hirt}@ethz.ch, ETH Zurich
cliuzhan@andrew.cmu.edu, Carnegie Mellon University

Abstract. Secure multi-party computation (MPC) allows a set of n parties to jointly compute an arbitrary computation over their private inputs. Two main variants have been considered in the literature according to the underlying communication model. Synchronous MPC protocols proceed in rounds, and rely on the fact that the communication network provides strong delivery guarantees within each round. Asynchronous MPC protocols achieve security guarantees even when the network delay is arbitrary.

While the problem of MPC has largely been studied in both variants with respect to both feasibility and efficiency results, there is still a substantial gap when it comes to communication complexity of adaptively secure protocols. Concretely, while adaptively secure synchronous MPC protocols with linear communication are known for a long time, the best asynchronous protocol communicates $\mathcal{O}(n^4\kappa)$ bits per multiplication.

In this paper, we make progress towards closing this gap by providing two protocols. First, we present an adaptively secure asynchronous protocol with optimal resilience $t < n/3$ and $\mathcal{O}(n^2\kappa)$ bits of communication per multiplication, improving over the state of the art protocols in this setting by a quadratic factor in the number of parties. The protocol has cryptographic security and follows the CDN approach [Eurocrypt'01], based on additive threshold homomorphic encryption.

Second, we show an optimization of the above protocol that tolerates up to $t < (1-\epsilon)n/3$ corruptions and communicates $\mathcal{O}(n \cdot \mathsf{poly}(\kappa))$ bits per multiplication under stronger assumptions.

1 Introduction

Secure multi-party computation (MPC) allows a set of parties to compute a function of their private inputs, in such a way that the parties' inputs remain secret, and the computed output is correct. This must hold even when an adversary corrupts a subset of the parties.

The problem of MPC [Yao82, GMW87, BGW88, CCD88, RB89] has been studied mostly in the so-called synchronous network model, where parties have access to synchronized clocks and there is an upper bound on the network communication delay. Although this model is theoretically interesting and may be justified in some settings, they fail to model real-world networks such as the Internet, which is inherently asynchronous. This gave rise to the asynchronous network model, where protocols do not rely on any timing assumptions, and messages sent can be arbitrarily delayed.

Asynchronous MPC protocols have received much less attention than their synchronous counterpart, partly because of their inherent difficulty and the weaker achievable security guarantees. In particular, one cannot distinguish between a dishonest party not sending a message, or an honest party that sent a message that was delayed by the adversary. As a result, parties have to make progress in the protocol after seeing messages from $n-t$ parties. This also implies that in this setting it is impossible to consider the inputs of all honest parties, i.e, the inputs of up to t (potentially honest) parties may be ignored. Moreover, one can show that the optimal achievable corruption tolerance in the asynchronous setting is $t < n/3$, even with setup, in both the cryptographic and information-theoretic setting; and perfect security is possible if and only if $t < n/4$.

[⋆] This work was partially carried out while the author was at ETH Zurich.

1.1 Communication Complexity of Asynchronous MPC protocols

The communication complexity in MPC has been the subject of a huge line of works. While the most communication-efficient synchronous MPC solutions without the usage of multiplicative-homomorphic encryption primitives achieve $\mathcal{O}(n\kappa)$ bits per multiplication gate (see e.g. [HN06, DI06, BTH08, BFO12, GLS19, GSZ20]), asynchronous MPC protocols still feature higher communication complexities, most notably when it comes to protocols with adaptive security.

In the adaptive security setting, all protocols are information-theoretic. The first protocol was provided by Ben-Or et al. [BKR94], and later improved by Patra et al. [PCR10, PCR08] to $\mathcal{O}(n^5\kappa)$ per multiplication, and by Choudhury [Cho20] to $\mathcal{O}(n^4\kappa)$ per multiplication.

When considering static security, the most efficient protocols with optimal resilience $t < n/3$ provide cryptographic security. The works by Hirt et al. [HNP05, HNP08] make use of an additive homomorphic encryption, with the protocol in [HNP08] being slightly more efficient and communicating $\mathcal{O}(n^2\kappa)$ per multiplication. The work by Choudhury and Patra [CP15] achieves $\mathcal{O}(n\kappa)$ per multiplication at the cost of using somewhat-homomorphic encryption, and the work by Cohen [Coh16] achieves a communication independent of the circuit size using fully-homomorphic encryption.

Other efficient solutions have been provided for the $t < n/4$ setting. Notable works include the protocols in [SR00, PSR02, CHP13, PCR15], achieving information-theoretic security.

1.2 Contributions

In this paper, we consider the problem of MPC over an asynchronous network with adaptive security. Our contributions can be summarized as follows.

First, we present an adaptively secure protocol with optimal resilience $t < n/3$ and $\mathcal{O}(n^2\kappa)$ bits of communication per multiplication, improving over the state of the art adaptively-secure protocols by a quadratic factor in the number of parties. The protocol follows the CDN approach [CDN01, DN03] and makes use of an additive threshold homomorphic encryption.

Second, we show a protocol that tolerates up to $t < (1-\epsilon)n/3$ corruptions and communicates a $\mathcal{O}(n \cdot \mathsf{poly}(\kappa))$ number of bits per multiplication, assuming secure erasures, non-interactive zero-knowledge proofs, and access to a network providing *atomic send*[1] (see e.g. [BKLZL20]), which guarantees that parties are able to atomically send messages to all other parties, and also guarantees that messages sent by honest parties cannot be retrieved back, even if the sender becomes corrupted. Note that a linear protocol with optimal resilience, and without the usage of any type of multiplicative-homomorphic encryption is not known even for the case of static security.

2 Preliminaries

We consider protocols among a set of n parties $P_1, \ldots, P_n$. We denote by κ the security parameter. Our protocols are proven in the model by Canetti [Can00a]. A summary can be found in appendix A.

2.1 Communication and Adversary Model

Parties have access to a network of point-to-point asynchronous and secure channels (for details of the asynchronous network model, we refer the reader to [CR98]). Asynchronous channels guarantee *eventual* delivery, meaning that messages sent are eventually delivered, and the scheduling of the messages is done by the adversary. In particular, the adversary can arbitrarily (but only finitely) delay all messages sent and deliver them out of order.

[1] This model has also been referred to as *weakly-adaptive* corruption, or simply adaptive corruption model in the literature.

We consider a computationally bounded adversary that can actively corrupt up to t parties in an adaptive manner. That is, as long as the adversary has corrupted strictly less than t parties, it can corrupt any party at any point in time based on the information during the protocol execution.

2.2 Zero-Knowledge Proofs of Knowledge

In this subsection, we introduce the notion of patchable zero-knowledge proof of knowledge. For more details, see [DN03].

Definition 1. *A* 2-party patchable zero-knowledge proof of knowledge *for a predicate Q is a protocol between a prover P and a verifier V where P has as public input an instance z and as secret input a witness x and V has public input the instance z and output in* $\{$accept, reject$\}$*. The protocol needs to satisfy the following properties.*

- Completeness: *On common input z, if P's secret input x is such that $Q(x, z) =$ true, then V accepts.*
- Soundness: *There exists an efficient program K (the knowledge extractor) that can interact with any prover P' such that if P' succeeds to make V accept with non-negligible probability, then K can extract a witness x' from its interaction with P' such that $Q(x', z) =$ true.*
- Zero-Knowledge: *For any efficient verifier V', there exists an efficient simulator S such that for any common input z, S can simulate a run of the protocol with V' in a computationally indistinguishable way.*
- Patchability: *Let z be an arbitrary instance and let $\tilde{t}$ be any step of the protocol. Let $T_{\tilde{t}}^{V'}(z)$ be the communication of the simulator (which might not know a witness to z) with a verifier V' in the simulated run of the protocol until step $\tilde{t}$. We require that there exists an efficient algorithm Pat that takes as input z, $\tilde{t}$, $T_{\tilde{t}}^{V'}(z)$ and a witness x such that $Q(x, z) =$ true and outputs randomness ν which satisfies the following: If an honest prover P executes the protocol with V' up to step $\tilde{t}$ on instance z and witness x using randomness ν, then the communication is identical to $T_{\tilde{t}}^{V'}(z)$. Furthermore, the randomness ν looks uniformly random to V'.*

All zero-knowledge proofs used in our protocol will be 2-party patchable zero-knowledge proofs of knowledge with constant communication complexity.

2.3 Universally Composable Commitments

In this section, we briefly introduce universally composable commitment schemes. A detailed exposition is given in Appendix B.

A commitment scheme allows a party P to commit to a value v towards other parties without revealing information about v. If at any point in time, P wants to reveal v, then it can open the given commitment to v.

A universally composable (UC) commitment scheme is a commitment scheme in the UC framework [Can00b]. Like usual commitment schemes, a UC commitment scheme is hiding and binding. Additionally, it is extractable (that is, the simulator can extract the value a corrupted party committed to from its commitment) and equivocable (that is, the simulator can simulate a commitment on behalf of an honest party towards a corrupted party without knowing the committed value and later open the given commitment to any value it wants). Since in our model we consider an adaptive adversary, we require that when the adversary corrupts a party, the simulator can patch the internal state of that party.

For all the commitments in our protocol, we will use a UC adaptively secure (equivocable and extractable) commitment scheme with constant communication complexity.

2.4 Threshold Homomorphic Encryption

We briefly discuss threshold homomorphic encryption schemes. For a detailed exposition, see Appendix C.

A threshold homomorphic encryption scheme is a tuple (KeyGen, Enc, $\mathsf{DecShare}$, Comb) of four algorithms, where

- KeyGen is a probabilistic algorithm that takes a security parameter κ, the number of parties n and the threshold parameter t as input and outputs a uniformly distributed tuple $(pk, sk_1, \dots, sk_n)$ where the public key pk is given to all parties and the secret key sk_i is given to P_i for all $i \in \{1, \dots, n\}$.
- Enc is an efficient probabilistic non-interactive algorithm that takes as input a public key pk and a message m from the message ring R_{pk} and outputs an encryption $\mathsf{Enc}_{pk}(m)$ of m. If we want to specify the randomness r used in the execution of the algorithm, we write $\mathsf{Enc}_{pk}(m, r)$.
  The Enc algorithm is a homomorphism in the sense that there exists an efficient algorithm that takes as input the public key pk and two encryptions $\mathsf{Enc}_{pk}(m_1, r_1)$ and $\mathsf{Enc}_{pk}(m_2, r_2)$ and outputs $\mathsf{Enc}_{pk}(m_1, r_1) \oplus_{pk} \mathsf{Enc}_{pk}(m_2, r_2) := \mathsf{Enc}_{pk}(m_1 +_{pk} m_2, r_1 \boxplus_{pk} r_2)$, where $+_{pk}$ and $\boxplus_{pk}$ are the group laws in the message space and the randomness space. Similarly, there exists an efficient algorithm that takes as input the public key pk, an encryption $\mathsf{Enc}_{pk}(m, r)$ and a message $c \in R_{pk}$ and outputs a uniquely determined encryption $c \odot_{pk} \mathsf{Enc}_{pk}(m, r)$ of $c \cdot_{pk} m$.
- $\mathsf{DecShare}$ is an efficient algorithm that takes as input an index $i \in \{1, \dots, n\}$, the public key pk, the secret key sk_i and a ciphertext c and outputs a decryption share c_i and a proof that c_i is correctly computed using i, pk, c and sk_i.
- Comb is an efficient algorithm that takes as input the public key pk, a ciphertext c and pairs (c_i, p_i) where each pair has a different index. The algorithm outputs a message m or fails.

The scheme is correct (that is, if at least $t + 1$ distinct decryption shares with valid proofs for the same ciphertext c are given as input to the Comb algorithm, then it outputs the message underlying c) and threshold semantically secure (that is, without the help of at least one honest party, an adversary corrupting at most t parties cannot extract information about the plaintext underlying a given ciphertext). Furthermore, there exists a patchable zero-knowledge proof of plaintext knowledge and a patchable zero-knowledge proof of correct multiplication with constant communication complexity.

From the definition of threshold homomorphic encryption scheme, it follows that there is an algorithm Blind that takes an encryption of a message m and the public key pk as input and outputs a uniformly random encryption of m (without knowing m). For details, see Proposition 2 in the appendix.

For convenience, we introduce the following functions which we will often use. For an encryption M in the ciphertext space, we define

$$\mathsf{Enc}_{pk}^{M} \colon (x, r) \to \mathsf{Enc}_{pk}^{M}(x, r) = (x \odot_{pk} M) \oplus_{pk} \mathsf{Enc}_{pk}(0_{pk}, r).$$

We call a preimage of the function Enc_{pk}^{M} of an encryption y a "preimage of y under (pk, M)". If we do not specify the second argument r of the function, then we implicitly mean that r is uniformly random in the randomness space. So (by the homomorphic property of the encryption scheme and by a similar reasoning as in the proof of Proposition 2) $\mathsf{Enc}_{pk}^{M}(x)$ is a uniformly random encryption of $x \cdot_{pk} m$, where m is the value encrypted by M.

In our protocol, we need the following additional properties of our encryption scheme.

- *Proof of compatible commitment:* Let $Q_{pk}^{M}((m', r_1, r_2), (y, B))$ be the binary predicate that is 1 if and only if $y = \mathsf{Enc}_{pk}^{M}(m', r_1)$ and (m', r_2) is the opening information for the commitment B. We require that for every public key pk and every encryption M, there exists an efficient patchable zero-knowledge proof of knowledge for Q_{pk}^{M} with constant communication complexity.

- *Lagrange arguments:* There exist n distinct elements $\{\alpha_1, \dots, \alpha_n\} \in (R_{pk} \backslash \{0_{pk}\})^n$ such that for all $(i, j) \in \{1, \dots, n\}^2$ we have that $\alpha_i - \alpha_j$ is invertible in R_{pk}. For these elements, the usual Lagrange polynomials and Lagrange coefficients are well-defined.
- *Patch:* Given a public key pk, two encryptions $E = \mathsf{Enc}_{pk}(0_{pk}, r_0)$ and $K = \mathsf{Enc}_{pk}(0_{pk}, r_K)$ of 0_{pk} under key pk and the randomness r_0 and r_K used, there exists an efficient probabilistic algorithm that given any constant x computes randomness r_E such that $E = (x \odot_{pk} K) \oplus_{pk} \mathsf{Enc}_{pk}(0_{pk}, r_E) = \mathsf{Enc}_{pk}^{K}(x, r_E)$.

Remark 1. By the homomorphic property of the encryption scheme, in the Patch property we have that $x \odot_{pk} K = \mathsf{Enc}_{pk}(0_{pk}, r_0 \boxminus_{pk} r_E)$. Since multiplication by a constant is a deterministic algorithm and since the randomness space is a group, this implies that if r_0 is uniformly random from the randomness space, then r_E is also uniformly random from the randomness space.

In Appendix C.1, we present the Paillier threshold encryption scheme which is an instantiation of the definition above.

3 Subprotocols

This section is devoted to the exposition of the subprotocols that will be used in the MPC protocol.

3.1 Agreement protocols

Often, parties need to have agreement on certain values or objects. To achieve this, we use the following primitives in our protocol.

1. *Reliable consensus:* Reliable consensus is a weaker version of asynchronous consensus. It allows the parties to agree on one of the honest parties' input values without requiring termination if there is no pre-agreement. More precisely, every party has a (private) input and the primitive guarantees that if all honest parties have the same input, then all honest parties output their inputs. Furthermore, if an honest parties outputs a value, then all other honest parties output the same value. In Appendix D.1, we discuss the definition of reliable consensus in more details and we present a reliable consensus protocol RC for $t < n/3$. Our protocol is based on Bracha's A-Cast protocol [Bra84] and has communication complexity $\mathcal{O}(n^2\kappa)$, where κ is the size any party's secret input.
2. *A-Cast:* A-Cast is an asynchronous broadcast protocol. It allows the parties to agree on the value of a sender without requiring termination if the sender is corrupted. More precisely, the sender has a private input and the primitive guarantees that if the sender is honest, then all parties output the senders message. Furthermore, if an honest party outputs a value, then all other honest parties output the same value. In Appendix D.2, we discuss the definition of reliable broadcast in more details and we present Bracha's reliable broadcast protocol RBC for $t < n/3$ [Bra84]. The protocol has communication complexity $\mathcal{O}(n^2\kappa)$, where κ is the size of the sender's input. Moreover, we show that if the sender has computationally indistinguishably distributed input, then the RBC protocol maintains computational indistinguishability.
   In some situations, we use Patra's $\mathsf{Multi\text{-}Valued\text{-}Acast}$ protocol [Pat11] which is a reliable broadcast protocol that achieves linear communication complexity for messages of size $\Omega(n^3 \log(n))$. This allows us to improve the efficiency of our MPC protocol.
3. *Byzantine agreement:* Byzantine agreement allows the parties to agree on one of the honest parties' input values. It guarantees that all honest parties terminate and that they output the same value. For $t < n/3$, Byzantine agreement can be achieved with expected communication complexity $\mathcal{O}(n^2)$. For a more detailed definition of Byzantine agreement, see Appendix D.3.

4. *ACS:* The agreement on a common subset (ACS) primitive allows the parties to agree on a set of at least $n-t$ parties that satisfy a certain property (a so-called ACS property). In Appendix D.4, we discuss the definitions of ACS property and ACS protocol in more details and we present an ACS protocol ACS with communication complexity $\mathcal{O}(n^3)$.

3.2 Decryption Protocols

To decrypt ciphertexts of our threshold homomorphic encryption scheme, we use two decryption protocols. The PrivDec protocol is a straightforward private decryption protocol which takes as input the public and private keys $pk, sk_1, \ldots, sk_n$, a ciphertext c and a party P and correctly decrypts c towards P even in the presence of an active adaptive adversary corrupting $t < n/3$ parties. The PubDec protocol is a public decryption protocol which takes as input $pk, sk_1, \ldots, sk_n$, $n-2t$ ciphertexts $c_1, \ldots, c_T$ and uses the PrivDec protocol to correctly publicly decrypt $c_1, \ldots, c_T$ even in the presence of an active adaptive adversary corrupting $t < n/3$ parties. The PubDec protocol has communication complexity $\mathcal{O}(n^2\kappa)$ and thus achieves linear communication complexity per decrypted ciphertext. For details about these two protocols and their guarantees, see Appendix D.5.

Remark 2. Additionally to the properties in the definiton of threshold homomorphic encryption scheme, we require the following from our encryption scheme. Let P be any party and let c_1 and c_2 be two computationally indistinguishably distributed ciphertexts with computationally indistinguishably distributed underlying plaintexts. An instance of the PrivDec protocol with (pk, c_1, P) as public input (and $sk_1, \ldots, sk_n$ as private inputs) is computationally indistinguishably distributed to an instance of the PrivDec protocol with (pk, c_2, P) as public input (and $sk_1, \ldots, sk_n$ as private inputs) even in the presence of an active adaptive adversary corrupting up to $t < n/3$ parties.

Remark 3. By inspection of the PubDec protocol in Appendix D.5, it is clear that the "computational indistinguishable decryption" property also holds for the PubDec protocol.

3.3 Multiplication

In this section, we briefly discuss the multiplication protocol. A detailed description is given in Appendix D.6.

The main idea for the multiplication protocol is to use circuit randomization [Bea92]. To make it more efficient, we apply the ideas of [DN07] and [BTH08], namely we use the PubDec protocol to process up to $T = \lfloor \frac{n-2t}{2} \rfloor$ independent multiplication gates simultaneously. Hence, the multiplication protocol takes as input T independent multiplication gates, their encrypted inputs and their associated multiplication triples and outputs the encrypted outputs of the given gates. The protocol guarantees that if the inputs to the processed multiplication gates are computationally indistinguishably distributed, then the executions of the multiplication protocol are as well (see Proposition 5). Furthermore, it communicates $\mathcal{O}(n^2\kappa)$ bits.

3.4 Triple Generation

This subsection is devoted to the introduction of the Triples protocol which takes as input an integer ℓ and outputs ℓ encrypted multiplication triples. The protocol is based on the multiplication protocol in [DN03] and the KFD-TRIPLES protocol in [HN06]. We first adapted their protocols to the asynchronous setting using the ACS primitive and then improved efficiency by amortizing the cost of the ACS instances over the number of generated triples and using the communication efficient Multi-Valued-Acast protocol.

Protocol Triples

```
1: Every party $P_j$ independently chooses uniformly random elements $a_j^i$ in the message space $R_{pk}$ and $r_j^i$ in
   the randomness space for all $i \in \{1, \ldots, \ell\}$. Then, $P_j$ computes $A_j^i = \mathsf{Enc}_{pk}(a_j^i, r_j^i)$ and uses the Multi-
   Valued-Acast protocol to broadcast $A_j^i$ for all $i \in \{1, \ldots, \ell\}$. Finally, $P_j$ proves to $P_k$ in zero-knowledge
   that it knows the plaintext underlying $A_j^i$ using the "proof of plaintext knowledge" property in Definition
   9 with instance $A_j^i$ and witness $(a_j^i, r_j^i)$ for all $i \in \{1, \ldots, \ell\}$ and all $k \in \{1, \ldots, n\}$.
2: Let $Q$ be the property such that a party $P_k$ satisfies $Q$ towards another party $P_j$ if and only if the broadcasts
   of all $A_k^i$ with $i \in \{1, \ldots, \ell\}$ terminated for $P_j$ and $P_j$ accepted all proofs of plaintext knowledge for $A_k^i$
   with $i \in \{1, \ldots, \ell\}$. The parties run the ACS protocol with $Q$ and obtain a set $S$ of parties.
3: The parties wait until the broadcasts of all parties in $S$ terminated and set $A^i = \bigoplus_{P_k \in S} A_k^i$ for all
   $i \in \{1, \ldots, \ell\}$.
4: Every party $P_j$ independently chooses uniformly random elements $b_j^i$ in the message space $R_{pk}$ and $r_j'^i$ in the
   randomness space for all $i \in \{1, \ldots, \ell\}$. Then, $P_j$ computes $B_j^i = \mathsf{Enc}_{pk}(b_j^i, r_j'^i)$ and $(C_j^i, r_j''^i) = \mathsf{Blind}(b_j^i \odot_{pk}
   A^i)$ and uses the Multi-Valued-Acast protocol to broadcast $B_j^i$ and $C_j^i$ for all $i \in \{1, \ldots, \ell\}$. Finally, $P_j$
   proves to $P_k$ in zero-knowledge that $C_j^i$ was computed correctly using the "proof of correct multiplication"
   property in Definition 9 with instance $(B_j^i, A^i, C_j^i)$ and witness $(b_j^i, r_j'^i, r_j''^i)$ for all $i \in \{1, \ldots, \ell\}$ and all
   $k \in \{1, \ldots, n\}$.
5: Let $Q'$ be the property such that a party $P_k$ satisfies $Q'$ towards another party $P_j$ if and only if the
   broadcast of all $(B_k^i, C_k^i)$ with $i \in \{1, \ldots, \ell\}$ terminated for $P_j$ and $P_j$ accepted all proofs of correct
   multiplication for $(B_k^i, A^i, C_k^i)$ with $i \in \{1, \ldots, \ell\}$. The parties run the ACS protocol with $Q'$ and obtain
   a set $S'$ of parties.
6: The parties wait until the broadcasts of all parties in $S'$ terminated and set $B^i = \bigoplus_{P_k \in S'} B_k^i$ and $C^i =
   \bigoplus_{P_k \in S'} C_k^i$ for all $i \in \{1, \ldots, \ell\}$.
7: Each party outputs $(A^i, B^i, C^i)$ for all $i \in \{1, \ldots, \ell\}$.
```

To prove security of the above Triples protocol, we give the simulator $\mathcal{S}_{\mathsf{Triples}}$ who does not have access to the secret keys of honest parties.

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

The simulator $\mathcal{S}_{\mathsf{Triples}}$ executes the protocol acting honestly on behalf of the honest parties. If the adversary decides to corrupt a party P_i at any point of the protocol, $\mathcal{S}_{\mathsf{Triples}}$ gives all the information it holds on behalf of P_i about the execution of the Triples protocol to the adversary.

Lemma 1. *The* Triples *protocol above satisfies the following:*

– Termination: *All honest parties terminate the protocol and output ℓ triples.*
– Consistency: *All honest parties output the same triples.*
– Correctness: *The output triples are correct.*
– Secrecy: *The plaintexts underlying the output triples are unknown to the adversary. In other words, the adversary has no more information about these plaintexts than that the plaintexts underlying the third components are the multiplication of the plaintexts underlying the corresponding first and second components.*
– Computational Uniform Randomness: *The distribution of the plaintexts underlying any output triple is computationally indistinguishable from the uniform distribution over the set of all triples $(a, b, a \cdot_{pk} b)$ for $a, b \in R_{pk}$.*
– Independence: *The plaintexts underlying any output triple are independent of the plaintexts underlying all other output triples.*
– Privacy: *The adversary's views in the simulation and the protocol are perfectly indistinguishably distributed, i.e. the adversary does not learn anything.*
– Communication complexity: *The protocol communicates $\mathcal{O}(n^2\ell\kappa + n^5 \log(n))$ bits.*

The proof is given in Appendix E.

Remark 4. If we choose $\ell\kappa = \Omega(n^3 \log(n))$, we obtain that the Triples protocol communicates $\mathcal{O}(n^2\kappa)$ bits per triple.

4 Asynchronous Adaptively Secure MPC Protocol

In this section, we present an asynchronous MPC protocol based on the protocols in [CDN01], [DN03] and [BTH08]. Then we informally prove that our protocol is secure against an active adaptive adversary corrupting up to t parties.

4.1 Ideal Functionality

In this subsection, we define the specification that our protocol achieves. The following exposition is based on [BKR94] and [CDN00].
Let $f : \mathbb{N} \times \{0,1\}^* \times (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be an efficiently computable function.

Functionality

```
1: The trusted party receives the security parameter $\kappa \in \{0,1\}^*$ and the number of parties $n \in \mathbb{N}$ as input.
2: Every party $P_i$ gives its input $x_i$ to the trusted party. Corrupted parties are allowed to give wrong input
   or even no input at all. If the adversary corrupts a party $P_j$ at any point in time after this step, then the
   trusted party gives $x_j$ to the adversary.
3: The adversary chooses a set of parties $S \subseteq \mathcal{P}$ of size at least $n - t$ and gives it to the trusted party.
4: The trusted party evaluates the function $f$ on the given inputs of parties in $S$ and using a default input
   $d$ for parties not in $S$. From this, it obtains output $y$.
5: The trusted party sends $y$ to all parties.
6: All honest parties output $y$. Corrupted parties can output whatever they like.
```

Recall that since we are in the asynchronous setting with at least $n - t$ honest parties, the size of the set S of parties whose inputs are considered for the evaluation of f is between $n - t$ and n. Note that it is not guaranteed that all parties in S are honest. However, we require from the adversary that it only includes corrupted parties in S for whom it gave input to the ideal functionality in step 2.

4.2 Informal Explanation of the Protocol

To achieve adaptive security in the asynchronous setting, we proceeded as follows. We started with the statically secure synchronous MPC protocol introduced by Cramer, Damgård and Nielsen [CDN01]. Next, we used circuit randomization [Bea92] to split the protocol into a preparation phase and a computation phase. After that, we adapted the protocol to the asynchronous setting using asynchronous broadcast and agreement on a common subset (ACS). Finally, we made the protocol adaptively secure by applying the techniques from Damgård and Nielsen [DN03], namely redefining the way values are encrypted and randomizing the output ciphertext in a specific way before decrypting it. Concretely, the new rule of encryption is: Given an encryption M and a value v to be encrypted, the encryption is set to $\mathsf{Enc}_{pk}^{M}(v)$. Recall that if we denote the value that M encrypts by m, then by the homomorphic property of the encryption scheme and by definition of the function Enc_{pk}^{M}, $\mathsf{Enc}_{pk}^{M}(v)$ is a uniformly random encryption of $v \cdot_{pk} m$. In the protocol, we will mostly choose $m = 1_{pk}$ to have an encryption of v while in the simulation we will often choose $m = 0_{pk}$ which helps the simulator to provide computationally indistinguishably distributed information. In detail, the idea of the protocol is the following.

Preparation phase:

– *Setup phase (steps* 1–4*)*: The keys for all the keyed primitives used in our protocol (namely the encryption scheme, the commitment scheme and the zero-knowledge proofs) are set up. Each party receives the keys it is entitled to along with public Lagrange arguments $\{\alpha_i\}_{i \in \{1,\dots,n\}}$. Additionally, two public encryptions K and R are set up and given to all

parties. The encryption K is a uniformly random encryption of 1_{pk} and the encryption R is a uniformly random encryption of 0_{pk}. In the simulation, the simulator will cheat by choosing K to be a uniformly random encryption of 0_{pk} and R to be a uniformly random encryption of 1_{pk}. By semantic security of the encryption scheme, this is computationally indistinguishable to the adversary.
Finally, the parties compute the circuit corresponding to the function to be evaluated and generate multiplication triples that will be used in the Evaluation phase to evaluate the multiplication gates of the circuit.

Computation phase:

– *Input phase (steps* 1 *and* 2*)*: The parties receive their inputs x_i needed for the execution and want to give them to an agreed function f. To do so, every party reliably broadcasts an encryption of its input applying the new rule of encryption with $M = K$. While $\mathsf{Enc}_{pk}^{K}(x_i)$ is indeed an encryption of x_i in the real world (recall that in the protocol K is an encryption of 1_{pk}), it is an encryption of 0_{pk} in the simulation as there, K is an encryption of 0_{pk}. Hence, in the simulation all encryptions of inputs will be encryptions of 0_{pk} independently of the inputs of the parties. However, the simulator needs to be able to extract the inputs of corrupted parties because it has to provide those inputs to the ideal functionality on behalf of the corrupted parties. This is why every party commits to its input towards every other party using a UC commitment scheme. The extraction property of UC commitments allows the simulator to extract the correct inputs of corrupted parties (ewnp) and give them to the ideal functionality. To ensure correctness and prevent the adversary in the real world from having more power than an adversary in the ideal world, the parties need to prove in zero-knowledge (using the "proof of compatible commitment" property) that they know a preimage of the reliably broadcasted encryption $\mathsf{Enc}_{pk}^{K}(x_i)$ under (pk, K) and that the first component of this preimage is the same as the value that they committed to. This is important because without these proofs a corrupted party could just wait for the reliable broadcast of another party P_j to terminate and then set its input to the same as the one from P_j without knowing it. This is not possible in the ideal world and therefore, we want to prevent it in the protocol execution. Furthermore, the simulator extracts the inputs of the corrupted parties from the commitments whereas for the computation in the protocol we will use the encryptions. Thus, the simulator needs to ensure that the value underlying the commitment and the first component of the preimage under (pk, K) of the encryption are the same so that it does not give wrong inputs to the ideal functionality on behalf of the corrupted parties. Finally, the parties run the ACS protocol and obtain a set S of size at least $n - t$ of parties that successfully broadcasted an encryption of their input which they committed to. The inputs of the parties in S are the ones that will be taken into account in the evaluation of f. Thus, the ACS protocol needs to ensure that S only contains parties that successfully completed the reliable broadcast of their inputs and all their zero-knowledge proofs towards at least one honest party (so that everything is correct and the simulator can extract the correct inputs ewnp as it received at least one valid commitment to every input of the corrupted parties in S). All inputs of parties that are not in S are set to a default value. Each party then waits until the reliable broadcast for every party in S terminated. It is okay for the parties to wait until the reliable broadcast of the parties in S terminate because we saw that for all parties P_k in S, there exists an honest party for which the reliable broadcast of P_k terminated. By the properties of reliable broadcast this implies that the reliable broadcast of P_k eventually terminates for all honest parties.
The computation of the encryptions, their reliable broadcast, the zero-knowledge proofs and the run of the ACS protocol are summed up in the BrACS protocol in Appendix F.
– *Evaluation phase (step* 3*)*: The parties evaluate the circuit on the encrypted inputs of the parties using the "$+_{pk}$-homomorphic" property, the "Multiplication by constant" property

and the multiplication protocol from Appendix D.6. In the end, the parties get a ciphertext c (called $\mathsf{Enc}_{pk}(s)$ in the protocol and $\mathsf{Enc}_{pk}(\hat{s})$ in the simulation).

– *Randomization phase (steps* 4–7*)*: Before the parties jointly decrypt c, they randomize it. This is done so that the simulator can cheat. In fact, as we saw above, all inputs to the circuit in the simulation are encryptions of 0_{pk}. By the correctness of the gates, this implies that all ciphertexts in the circuit are encryptions of 0_{pk} (not counting the intermediate ciphertexts in the multiplication protocol). Hence, c is also an encryption of 0_{pk} and therefore, we cannot simply honestly decrypt c as otherwise the simulator would fail to provide a computationally indistinguishable simulation with overwhelming probability. Furthermore, our encryption scheme is not adaptively secure which is why we can not decrypt c to anything but 0_{pk} either. Thus, the parties randomize the ciphertext before decrypting it honestly.
To randomize the ciphertext c, the parties do the following. Each party chooses a random r_i and reliably broadcasts the encryption $\mathsf{Enc}_{pk}^{R}(r_i)$. Then the parties agree on a set $\hat{S}$ of parties of size at least $t+1$ of successful broadcasts using the ACS protocol. Denote the indices of the parties in the set $\hat{S}$ by I. Next, the parties consider the unique polynomial p of degree $|I|-1$ that goes through $\mathsf{Enc}_{pk}^{R}(r_i)$ at position α_i for all $i \in I$. They interpolate this polynomial at 0_{pk} and add this to c using the "$+_{pk}$-homomorphic" and the "Multiplication by constant" properties of the encryption scheme. This gives the new ciphertext c' (denoted by $\mathsf{Enc}_{pk}(s)'$ in the protocol and the simulation). In the real execution, R is an encryption of 0_{pk} under pk and therefore, all $\mathsf{Enc}_{pk}^{R}(r_i)$ are encryptions of 0_{pk} under pk. Since interpolation is a linear operation and the encryption scheme is homomorphic, the value of p at 0_{pk} will also be an encryption of 0_{pk} and thus c' will encrypt the same message as c. In the simulation however, R is an encryption of 1_{pk}. This will help the simulator to cheat. Concretely, the simulator will adjust the r_i's of honest parties so that at position 0_{pk}, p will have a uniformly random encryption of the output s (received from the ideal functionality) of the function f evaluated on the inputs of the parties. This is possible since $|I| \geqslant t+1$ and hence, there is at least one honest party whose r_i is taken into account in the randomization and can be chosen by the simulator in the simulation. Since c is an encryption of 0_{pk} in the simulation, we get that c' encrypts s as wanted. But we need to integrate a mechanism that allows the simulator to choose the r_i's of honest parties according to those of corrupted parties. This is done in the following way.
Before reliably broadcasting $\mathsf{Enc}_{pk}^{R}(r_i)$ and agreeing on a set of successful broadcasts, the parties commit to their r_i and use the BrACS protocol to reliably broadcast $\mathsf{Enc}_{pk}^{K}(r_i)$ and agree on a set S' of successful broadcasts (including a successful proof of compatible commitment). By the ACS property we will use and by the guarantees of the ACS protocol, we have that the simulator received at least one valid commitment to r_k for every corrupted party $P_k \in S'$. Thus, it can extract all r_k from corrupted parties in S' ewnp (by the extraction property of UC commitment schemes). Now the simulator can adjust the r_i's of the honest parties as described above. Then the parties execute the BrACS for $\mathsf{Enc}_{pk}^{R}(r_i)$ (see above) but using the same commitments in the zero-knowledge proof as in the previous BrACS (with $\mathsf{Enc}_{pk}^{K}(r_i)$). We obtain a set S'' and encryptions $\mathsf{Enc}_{pk}^{R}(r_i)$ for all $P_i \in S''$. The ACS property the parties use in the second BrACS is slightly modified to ensure that the value used to compute the broadcasted encryption in the first BrACS (the one with $\mathsf{Enc}_{pk}^{K}(r_i)$) and the value used to compute the broadcasted encryption in the second BrACS (the one with $\mathsf{Enc}_{pk}^{R}(r_i)$) is the same except with negligible probability. Concretely, the property ensures that for all $P_i \in S''$ at least one honest party likes P_i for both BrACS executions. Since those BrACS protocols were run with the same commitments, we can be sure that the values used to compute the broadcasted encryptions are the same in both runs of the BrACS protocol. Then we set $\hat{S} = S' \cap S''$ and observe that $\hat{S}$ is of size at least $n - 2t \geqslant t+1$ as wanted.
Note that the simulator has to know the r_k's of corrupted parties in $\hat{S} \subseteq S'$ before the broadcasting of $\mathsf{Enc}_{pk}^{R}(r_i)$ because while it can patch the encryptions and proofs of the first

BrACS (with $\mathsf{Enc}^K_{pk}(r_i)$) due to K being an encryption of 0_{pk}, it can *not* do the same for the second BrACS (with $\mathsf{Enc}^R_{pk}(r_i)$) because R is an encryption of 1_{pk}.
- *Output phase (steps* 8 *and* 9*)*: The parties decrypt c' and obtain s. Then they run the reliable consensus protocol on secret input s as termination procedure. The persistency property of reliable consensus ensures that everyone terminates on the same correct output s.

A detailed description of the protocol can be found in Appendix F.

4.3 Main Theorem

Our protocol achieves the following.

Theorem 1. *The MPC protocol in Appendix F t-securely realizes the ideal functionality in Subsection 4.1 in the* KG*-hybrid model for $t < n/3$. The protocol communicates $\mathcal{O}(c_M n^2\kappa + Dn^2\kappa + n^3\kappa + n^5\log(n))$ bits, where c_M is the number of multiplication gates in the circuit and D is the multiplicative depth of the circuit.*

In Appendix G, the simulator and an informal proof are given.

5 Near-Linear MPC in the Atomic Send Model

In this section, we show how to improve the efficiency of our MPC protocol at the cost of stronger assumptions on the model.
Taking a closer look at the communication complexity of the protocol in Appendix F reveals that the complexity is dominated by the communication in the Triples protocol. While the number of messages sent between the parties per produced triple (and hence per multiplication gate of the circuit) in the Triples protocol is quadratic in the number of parties, the computation phase of the protocol only needs near-linear communication per evaluated gate assuming a shallow circuit (except for the input phase which has quadratic communication complexity per input gate). By considering slightly stronger assumptions on the model, we can reduce the communication complexity of the triple generation and obtain a near-linear MPC protocol.

5.1 Model

In this subsection, we present the model which will be used to achieve better efficiency in the generation of multiplication triples. The subsection is based on [BKLZL20].

As before (see Subsection 2.1), we consider multiparty computation among a set of n parties $P_1, \ldots, P_n$, where every pair of parties is connected by a secure asynchronous communication channel. A protocol in our setting comprises a number of atomic steps.
The adversary in the new setting is computationally bounded and can actively corrupt up to t parties in an atomic send adaptive manner. That is, as long as the adversary has corrupted strictly less than t parties, it can corrupt any party at any point in time considering all the information it has seen so far and make this party behave as it wishes for the remaining steps of the protocol. However, if in some step a party needs to send several messages simultaneously, then the adversary is only allowed to corrupt this party before or after it sent all the messages (that is, the adversary can not corrupt the party in the midst of the sending). Furthermore, messages sent by any honest party P_i are guaranteed to arrive, even if P_i is later corrupted. Once a party is corrupted, the adversary learns its internal state and the party remains corrupted until the end of the protocol.
We assume the existence of non-interactive zero-knowledge (NIZK) proofs and secure erasure. Moreover, we assume the existence of a trusted party that provides the parties with public and private setup information before the execution of a protocol, more details below. The size of the setup is defined to be the sum of the size of the total private setup information and the size of the public setup information (hence, we count the private information of each party separately, but the public information only once for all parties).

5.2 VACS

This subsection is devoted to the introduction of the VACS primitive. We follow the exposition in [BKLZL20].
In the efficient WeakTriples protocol, we need a primitive that allows the parties to agree on a sufficiently large subset of their inputs satisfying a specific predicate. This can be achieved by the VACS primitive.

Definition 2. *Consider a predicate Q and an n-party protocol π, where every party P_i has a secret input m_i and outputs a multiset S of size at most n. Every honest party's input satisfies Q and every party terminates upon generating output. We say that π is a* t-secure Q-validated ACS protocol with q-output quality *if for all adversaries corrupting up to t parties and for all inputs the following is satisfied:*

- Q-Validity: *Let S be the output of an honest party. Then for every $m \in S$, we have $Q(m) = 1$.*
- Consistency: *All honest parties agree on S.*
- q-Output Quality: *The output multiset S of every honest party is of size at least q and contains the inputs of at least $q - t$ parties that were honest at the beginning of the protocol.*

Theorem 2. *Let $0 < \epsilon < 1/3$, $t \leqslant (1-2\epsilon)\cdot n/3$ and $q \leqslant (1+\epsilon/2)\cdot 2n/3$. There exists a t-secure Q-validated ACS protocol $\Pi_{\mathsf{VACS}}^{q,Q}$ with q-output quality, expected setup size $\mathcal{O}(q\kappa^4)$ and expected communication complexity $\mathcal{O}((\mathcal{I}+\kappa^3)\cdot q\kappa n)$, where $\mathcal{I}$ is the size of any party's secret input. In addition to the properties of t-secure Q-validated ACS protocols, the $\Pi_{\mathsf{VACS}}^{q,Q}$ protocol guarantees that the output multiset S contains the inputs of at least $\frac{1}{2}q$ parties that were honest at the beginning of the protocol except with probability smaller than $e^{\frac{-q\epsilon^2}{(2-3\epsilon)(2+\epsilon)}}$.*

The construction of $\Pi_{\mathsf{VACS}}^{q,Q}$ and the proof of the first part of the theorem can be found in [BKLZL20]. The second part of the theorem can be proven using Lemma 24 of [BKLZL20].

5.3 Triple Generation

To obtain an efficient protocol for the triple generation in the atomic send model, we start with our Triples protocol from Subsection 3.4 and make it more efficient using the VACS primitive, NIZK proofs and erasure. The following protocol is inspired by the protocols in [BKLZL20]. It takes as input an integer ℓ and outputs ℓ encrypted multiplication triples.

Protocol WeakTriples

Let ℓ be the number of triples we want to generate . We assume that the parties have access to the setup for two runs of the VACS protocol with output quality κ.

1: Each party P_j independently chooses uniformly random messages $a_j^k \in R_{pk}$ and uniformly random elements r_j^k in the randomness space for all $k \in \{1,\dots,\ell\}$. Then, P_j computes $A_j^k = \mathsf{Enc}_{pk}(a_j^k, r_j^k)$ and an NIZK proof $p_{1,j}^k$ of plaintext knowledge with instance A_j^k and witness (a_j^k, r_j^k) for all $k \in \{1,\dots,\ell\}$. Finally, P_j erases (a_j^k, r_j^k) for all $k \in \{1,\dots,\ell\}$.

2: The parties run an instance of the $\Pi_{\mathsf{VACS}}^{\kappa,Q}$ protocol with output quality κ where every party P_j has input $\{(A_j^k, p_{1,j}^k)\}_{k\in\{1,\dots,\ell\}}$ and $Q(\{(A_j^k, p_{1,j}^k)\}_{k\in\{1,\dots,\ell\}}) = 1$ if and only if $p_{1,j}^k$ is a correct NIZK proof of plaintext knowledge with instance A_j^k for all $k \in \{1,\dots,\ell\}$. The parties obtain a multiset S of size at least κ and define $A^i = \bigoplus_{j:\ \{(A_j^k, p_{1,j}^k)\}_{k\in\{1,\dots,\ell\}}\in S} A_j^i$ for all $i \in \{1,\dots,\ell\}$.

3: Each party P_j independently chooses uniformly random messages $b_j^k \in R_{pk}$ and uniformly random elements $\hat{r}_j^k$ in the randomness space for all $k \in \{1,\dots,\ell\}$. Then, P_j computes $B_j^k = \mathsf{Enc}_{pk}(b_j^k, \hat{r}_j^k)$ and $(C_j^k, \tilde{r}_j^k) = \mathsf{Blind}(b_j^k \odot_{pk} A^k)$, where Blind is the blinding algorithm of the encryption scheme. Furthermore, P_j computes an NIZK proof $p_{2,j}^k$ of correct multiplication with instance (B_j^k, A^k, C_j^k) and witness $(b_j^k, \hat{r}_j^k, \tilde{r}_j^k)$ for all $k \in \{1,\dots,\ell\}$. Finally, P_j erases $(b_j^k, \hat{r}_j^k)$, $\tilde{r}_j^k$ and the information used in the blinding algorithm for all $k \in \{1,\dots,\ell\}$.

4: The parties run an instance of the $\Pi_{\mathsf{VACS}}^{\kappa,Q'}$ protocol with output quality κ where every party P_j has input $\{(B_j^k, C_j^k, p_{2,j}^k)\}_{k\in\{1,\dots,\ell\}}$ and $Q'(\{(B_j^k, C_j^k, p_{2,j}^k)\}_{k\in\{1,\dots,\ell\}}) = 1$ if and only if $p_{2,j}^k$ is a correct NIZK proof of

```
   correct multiplication with instance $(B_j^k, A^k, C_j^k)$ for all $k \in \{1,\ldots,\ell\}$. The parties obtain a multiset $S'$ of
   size at least $\kappa$ and define $B^i = \bigoplus_{j:\ \{(B_j^k, C_j^k, p_{2,j}^k)\}_{k\in\{1,\ldots,\ell\}} \in S'} B_j^i$ and $C^i = \bigoplus_{j:\ \{(B_j^k, C_j^k, p_{2,j}^k)\}_{k\in\{1,\ldots,\ell\}} \in S'} C_j^i$
   for all $i \in \{1,\ldots,\ell\}$.
5: Every party outputs $(A^i, B^i, C^i)$ for all $i \in \{1,\ldots,\ell\}$.
```

Remark 5. Because we want to ensure that all parties who contribute to the triples know the plaintexts underlying their contributions and because the VACS protocol requires Q and Q' (defined in steps 2 and 4) to be predicates on the inputs of the parties to the VACS protocol, we need to use NIZK proofs.

To prove security of the above WeakTriples protocol, we give the simulator $\mathcal{S}_{\mathsf{WeakTriples}}$ who does not have access to the secret keys of honest parties.

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

The simulator $\mathcal{S}_{\mathsf{WeakTriples}}$ executes the protocol acting honestly on behalf of the honest parties. If the adversary decides to corrupt a party P_i at any point of the protocol, $\mathcal{S}_{\mathsf{WeakTriples}}$ gives all the information it holds on behalf of P_i about the execution of the WeakTriples protocol to the adversary.

Lemma 2. *For $0 < \epsilon < 1/3$ and $t \leqslant (1-2\epsilon)\cdot n/3$, the* WeakTriples *protocol above satisfies the following:*

- Termination: *All honest parties terminate the protocol and output ℓ triples.*
- Consistency: *All honest parties output the same triples.*
- Correctness: *The output triples are correct.*
- Secrecy: *The plaintexts underlying the output triples are unknown to the adversary. In other words, the adversary has no more information about these plaintexts than that the plaintexts underlying the third components are the multiplication of the plaintexts underlying the corresponding first and second components.*
- Computational Uniform Randomness: *The distribution of the plaintexts underlying any output triple is computationally indistinguishable from the uniform distribution over the set of all triples $(a, b, a \cdot_{pk} b)$ for $a, b \in R_{pk}$.*
- Independence: *The plaintexts underlying any output triple are independent of the plaintexts underlying all other output triples.*
- Privacy: *The adversary's views in the simulation and the protocol are perfectly indistinguishably distributed, i.e. the adversary does not learn anything.*
- Communication complexity: *The protocol communicates $\mathcal{O}(\ell\kappa^3 n + \kappa^5 n)$ bits.*

The proof is given in Appendix H.

5.4 Main Theorem for the Atomic Send Model

By replacing the instance of the Triples protocol in step 4 of the Preparation Phase of the MPC protocol in Appendix F by the WeakTriples protocol above, we can improve the communication complexity of our MPC protocol and achieve $\mathcal{O}(n \cdot \mathsf{poly}(\kappa))$ bits per multiplication.

Theorem 3. *There exists an MPC protocol that t-securely realizes the ideal functionality in Subsection 4.1 in the* KG*-hybrid atomic send model for $t < n/3$ and communicates $\mathcal{O}(c_M n\kappa^3 + Dn^2\kappa + n^3\kappa + n\kappa^5)$ bits, where c_M is the number of multiplication gates in the circuit and D is the multiplicative depth of the circuit.*

References

[Bea92] Donald Beaver. Efficient multiparty protocols using circuit randomization. In Joan Feigenbaum, editor, *Advances in Cryptology — CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pages 420–432, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg. doi:https://doi.org/10.1007/3-540-46766-1_34.

[BFO12] Eli Ben-Sasson, Serge Fehr, and Rafail Ostrovsky. Near-linear unconditionally-secure multiparty computation with a dishonest minority. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 663–680. Springer, Heidelberg, August 2012. doi:10.1007/978-3-642-32009-5_39.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *20th ACM STOC*, pages 1–10. ACM Press, May 1988. doi:10.1145/62212.62213.

[BHN08] Zuzana Beerliova-Trubiniova, Martin Hirt, and Jesper Buus Nielsen. Almost-asynchronous mpc with faulty minority. Cryptology ePrint Archive, Report 2008/416, 2008. https://eprint.iacr.org/2008/416.

[BKLZL20] Erica Blum, Jonathan Katz, Chen-Da Liu-Zhang, and Julian Loss. Asynchronous byzantine agreement with subquadratic communication. Cryptology ePrint Archive, Report 2020/851, 2020. https://eprint.iacr.org/2020/851.

[BKR94] Michael Ben-Or, Boaz Kelmer, and Tal Rabin. Asynchronous secure computations with optimal resilience (extended abstract). In Jim Anderson and Sam Toueg, editors, *13th ACM PODC*, pages 183–192. ACM, August 1994. doi:10.1145/197917.198088.

[Bra84] Gabriel Bracha. An asynchronous [(n - 1)/3]-resilient consensus protocol. In *Proceedings of the third annual ACM symposium on Principles of distributed computing*, PODC '84, page 154–162, New York, NY, USA, 1984. Association for Computing Machinery. doi:10.1145/800222.806743.

[BTH08] Zuzana Beerliová-Trubíniová and Martin Hirt. Perfectly-secure MPC with linear communication complexity. In Ran Canetti, editor, *TCC 2008*, volume 4948 of *LNCS*, pages 213–230. Springer, Heidelberg, March 2008. doi:10.1007/978-3-540-78524-8_13.

[Can95] Ran Canetti. *Studies in Secure Multiparty Computation and Applications.* PhD thesis, The Weizmann Institute of Science, Rehovot 76100, Israel, 06 1995. URL: http://www.wisdom.weizmann.ac.il/~oded/PSX/ran-phd.pdf.

[Can00a] Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13:143–202, 2000. doi:10.1007/s001459910006.

[Can00b] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067, 2000. https://eprint.iacr.org/2000/067.

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In *20th ACM STOC*, pages 11–19. ACM Press, May 1988. doi:10.1145/62212.62214.

[CDN00] Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. Multiparty computation from threshold homomorphic encryption. Cryptology ePrint Archive, Report 2000/055, 10 2000. https://eprint.iacr.org/2000/055.

[CDN01] Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. Multiparty computation from threshold homomorphic encryption. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 280–299. Springer, Heidelberg, May 2001. doi:10.1007/3-540-44987-6_18.

[CF01] Ran Canetti and Marc Fischlin. Universally composable commitments. In Joe Kilian, editor, *Advances in Cryptology — CRYPTO 2001*, pages 19–40, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[Cho20] Ashish Choudhury. Optimally-resilient unconditionally-secure asynchronous multi-party computation revisited. Cryptology ePrint Archive, Report 2020/906, 2020. https://eprint.iacr.org/2020/906.

[CHP12] Ashish Choudhury, Martin Hirt, and Arpita Patra. Unconditionally secure asynchronous multiparty computation with linear communication complexity. Cryptology ePrint Archive, Report 2012/517, 2012. https://eprint.iacr.org/2012/517.

[CHP13] Ashish Choudhury, Martin Hirt, and Arpita Patra. Asynchronous multiparty computation with linear communication complexity. In *International Symposium on Distributed Computing*, pages 388–402. Springer, 2013.

[Coh16] Ran Cohen. Asynchronous secure multiparty computation in constant time. In Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang, editors, *PKC 2016, Part II*, volume 9615 of *LNCS*, pages 183–207. Springer, Heidelberg, March 2016. doi:10.1007/978-3-662-49387-8_8.

[CP15] Ashish Choudhury and Arpita Patra. Optimally resilient asynchronous MPC with linear communication complexity. In *Proc. Intl. Conference on Distributed Computing and Networking (ICDCN)*, pages 1–10, 2015.

[CR98] Ran Canetti and Tal Rabin. Fast asynchronous byzantine agreement with optimal resilience, 1998. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.8.8120.

[DI06] Ivan Damgård and Yuval Ishai. Scalable secure multiparty computation. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 501–520. Springer, Heidelberg, August 2006. doi:10.1007/11818175_30.

[DJ00] Ivan B. Damgård and Mads J. Jurik. Efficient protocols based on probabilistic encryption using composite degree residue classes. Report Series RS-00-5, BRICS Basic Research in Computer Science, Department of Computer Science, University of Aarhus, Denmark, 03 2000. URL: https://www.brics.dk/RS/00/5/BRICS-RS-00-5.pdf.

[DN03] Ivan Damgård and Jesper Buus Nielsen. Universally composable efficient multiparty computation from threshold homomorphic encryption. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 247–264. Springer, Heidelberg, August 2003. doi:10.1007/978-3-540-45146-4_15.

[DN07] Ivan Damgård and Jesper Buus Nielsen. Scalable and unconditionally secure multiparty computation. In *Advances in Cryptology - CRYPTO 2007, 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007, Proceedings*, volume 4622 of *Lecture Notes in Computer Science*, pages 572–590. Springer, 2007. URL: https://iacr.org/archive/crypto2007/46220565/46220565.pdf, doi:10.1007/978-3-540-74143-5_32.

[FPS01] Pierre-Alain Fouque, Guillaume Poupard, and Jacques Stern. Sharing decryption in the context of voting or lotteries. In Yair Frankel, editor, *Financial Cryptography*, pages 90–104, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[GLS19] Vipul Goyal, Yanyi Liu, and Yifan Song. Communication-efficient unconditional MPC with guaranteed output delivery. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part II*, volume 11693 of *LNCS*, pages 85–114. Springer, Heidelberg, August 2019. doi:10.1007/978-3-030-26951-7_4.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987. doi:10.1145/28395.28420.

[GSZ20] Vipul Goyal, Yifan Song, and Chenzhi Zhu. Guaranteed output delivery comes free in honest majority MPC. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 618–646. Springer, Heidelberg, August 2020. doi:10.1007/978-3-030-56880-1_22.

[HN06] Martin Hirt and Jesper Buus Nielsen. Robust multiparty computation with linear communication complexity. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 463–482. Springer, Heidelberg, August 2006. doi:10.1007/11818175_28.

[HNP05] Martin Hirt, Jesper Buus Nielsen, and Bartosz Przydatek. Cryptographic asynchronous multiparty computation with optimal resilience (extended abstract). In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 322–340. Springer, Heidelberg, May 2005. doi:10.1007/11426639_19.

[HNP08] Martin Hirt, Jesper Buus Nielsen, and Bartosz Przydatek. Asynchronous multi-party computation with quadratic communication. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP 2008, Part II*, volume 5126 of *LNCS*, pages 473–485. Springer, Heidelberg, July 2008. doi:10.1007/978-3-540-70583-3_39.

[MMR15] Achour Mostéfaoui, Hamouma Moumen, and Michel Raynal. Signature-free asynchronous binary byzantine consensus with $t < n/3$, o(n^2) messages, and o(1) expected time. *J. ACM*, 62(4), 8 2015. doi:10.1145/2785953.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *Advances in Cryptology — EUROCRYPT '99*, pages 223–238, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[Pat11] Arpita Patra. Error-free multi-valued broadcast and byzantine agreement with optimal communication complexity. In Antonio Fernàndez Anta, Giuseppe Lipari, and Matthieu Roy, editors, *Principles of Distributed Systems*, pages 34–49, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[PCR08] Arpita Patra, Ashish Choudhury, and C. Pandu Rangan. Efficient asynchronous multiparty computation with optimal resilience. Cryptology ePrint Archive, Report 2008/425, 2008. https://eprint.iacr.org/2008/425.

[PCR10] Arpita Patra, Ashish Choudhary, and C. Pandu Rangan. Efficient statistical asynchronous verifiable secret sharing with optimal resilience. In Kaoru Kurosawa, editor, *ICITS 09*, volume 5973 of *LNCS*, pages 74–92. Springer, Heidelberg, December 2010. doi:10.1007/978-3-642-14496-7_7.

[PCR15] Arpita Patra, Ashish Choudhury, and C. Pandu Rangan. Efficient asynchronous verifiable secret sharing and multiparty computation. *Journal of Cryptology*, 28(1):49–109, January 2015. doi:10.1007/s00145-013-9172-7.

[PSR02] B. Prabhu, K. Srinathan, and C. Pandu Rangan. Asynchronous unconditionally secure computation: An efficiency improvement. In Alfred Menezes and Palash Sarkar, editors, *INDOCRYPT 2002*, volume 2551 of *LNCS*, pages 93–107. Springer, Heidelberg, December 2002.

[RB89] Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In *21st ACM STOC*, pages 73–85. ACM Press, May 1989. doi:10.1145/73007.73014.

[Sha79] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, November 1979. `doi:10.1145/359168.359176`.

[SR00] K. Srinathan and C. Pandu Rangan. Efficient asynchronous secure multiparty distributed computation. In Bimal K. Roy and Eiji Okamoto, editors, *INDOCRYPT 2000*, volume 1977 of *LNCS*, pages 117–129. Springer, Heidelberg, December 2000.

[Yao82] Andrew Chi-Chih Yao. Theory and applications of trapdoor functions (extended abstract). In *23rd FOCS*, pages 80–91. IEEE Computer Society Press, November 1982. `doi:10.1109/SFCS.1982.45`.

Appendix

A MPC Security Model

In this subsection, we briefly summarize the main aspects of the model of Canetti [Can00a] that we use for our protocols. The goal of a protocol is to emulate an *ideal functionality*, which models a trusted party that receives inputs and provides outputs to the parties. Intuitively, a protocol is proven secure if one shows that for any attack that an adversary can perform in the real protocol, one can construct a corresponding ideal adversary which can perform the same attack in the ideal world via what is called the simulator. The simulator runs in the ideal world, interacting only with the ideal functionality and the real adversary, and has to be such that the distributions of messages seen in the real world and ideal world executions are indistinguishable.

Definition 3 (Real world). *Consider an n-party protocol π and an active adaptive adversary $\mathcal{A}$ corrupting up to t parties. Let κ be a security parameter, a be an auxiliary string for the adversary and x be the vector of public and private inputs of the parties. We define $REAL_{\pi,\mathcal{A}}(\kappa, x, a)$ to be the distribution of the private and public outputs of all parties in an execution of π on inputs κ, x and a in the network model described in Subsection 2.1 and in the presence of adversary $\mathcal{A}$ (the probability comes from the randomness used in an execution of π). The distribution ensemble $\{REAL_{\pi,\mathcal{A}}(\kappa, x, a)\}_{\kappa\in\mathbb{N},x\in(\{0,1\}^*)^{2n},a\in\{0,1\}^*}$ is denoted by $REAL_{\pi,\mathcal{A}}$. This setting is called the* real world model.

Let us introduce the notion of an ideal functionality.

Definition 4. *An ideal functionality F is an incorruptible trusted party that is connected via a secure channel to each party, takes input from all parties and gives output to all parties according to a specified input-output relation. In other words, F is a black box achieving a specified input-output relation.*

Definition 5 (Ideal world). *Consider an ideal functionality F and an active adaptive simulator (a.k.a. ideal world adversary) $\mathcal{S}$ corrupting up to t parties. As in the previous definition, let κ be a security parameter, a be an auxiliary string for the simulator and x be the vector of public and private inputs of the parties. We define $IDEAL_{F,\mathcal{S}}(\kappa, x, a)$ to be the distribution of the private and public outputs of all parties in an execution of F on inputs κ, x and a in the presence of the simulator $\mathcal{S}$ (the probability comes from the randomness used in an execution of F). The distribution ensemble $\{IDEAL_{F,\mathcal{S}}(\kappa, x, a)\}_{\kappa\in\mathbb{N},x\in(\{0,1\}^*)^{2n},a\in\{0,1\}^*}$ is denoted by $IDEAL_{F,\mathcal{S}}$. This setting is called the* ideal world model.

Definition 6 (Hybrid world). *Let $F_1, \ldots, F_l$ be ideal functionalities. We call the $(F_1, \ldots, F_l)$-*hybrid model *to be the real world model where the parties additionally have access to the ideal functionalities $F_1, \ldots, F_l$. The distribution of the private and public outputs of all parties in the $(F_1, \ldots, F_l)$-hybrid model with inputs κ, x and a in the presence of an active adaptive $(F_1, \ldots, F_l)$-hybrid-adversary $\mathcal{A}$ corrupting up to t parties is denoted by $REAL^{(F_1,\ldots,F_l)}_{\pi,\mathcal{A}}(\kappa, x, a)$. Like before, the distribution ensemble $\{REAL^{(F_1,\ldots,F_l)}_{\pi,\mathcal{A}}(\kappa, x, a)\}_{\kappa\in\mathbb{N},x\in(\{0,1\}^*)^{2n},a\in\{0,1\}^*}$ is denoted by $REAL^{(F_1,\ldots,F_l)}_{\pi,\mathcal{A}}$. The ()-hybrid model is the real world model.*

The security notion states that the execution of a protocol in the hybrid model does not reveal any more information to the adversary than the ideal functionality that the protocol realizes. This is formalized as follows.

Definition 7. *Consider an n-party protocol π and ideal functionalities $F, F_1, \ldots, F_l$. We say that π* t-securely realizes F in the $(F_1, \ldots, F_l)$-hybrid model *if for all active adaptive $(F_1, \ldots, F_l)$-hybrid-adversaries $\mathcal{A}$ corrupting up to t parties there exists an active adaptive simulator $\mathcal{S}$ corrupting up to t parties such that $REAL^{(F_1,\ldots,F_l)}_{\pi,\mathcal{A}} \overset{c}{\approx} IDEAL_{F,\mathcal{S}}$.*

B Details on Universally Composable Commitment Schemes

This subsection is devoted to the introduction of universally composable commitment schemes and some of their properties. The exposition is based on [CF01]. For more details and instantiations see [CF01] and [DN03].

The authors of [CF01] define the ideal functionality of a universally composable commitment scheme between parties $P_1, \ldots, P_n$ and simulator $\mathcal{S}$ as follows. Let *sid* be a session ID (*sid* is useful when running several copies of F).

Functionality F

1: On input (Commit, *sid*, P_i, P_j, $b \in \{0,1\}$) from P_i, save b and send (Receipt, *sid*, P_i, P_j) to P_j and $\mathcal{S}$. Disregard any further Commit-messages from P_i to P_j associated with *sid*.
2: On input (Open, *sid*, P_i, P_j) from P_i, if there is a saved commitment value b from P_i to P_j associated with *sid*, send (Open, *sid*, P_i, P_j, b) to P_j and $\mathcal{S}$ and halt. Else halt.

Step 1 models a Commit phase where a party P_i commits to $b \in \{0,1\}$ towards a party P_j. The ideal functionality receives the value b, the session ID, the sender and the receiver through the Commit-message and informs the receiver as well as the simulator that the sender committed to some value associated to the session ID by sending the Receipt-message.
Step 2 models an Opening phase where a party P_i opens its commitment towards P_j (if it indeed committed to a value towards P_j).

Definition 8. *A protocol that securely realizes F is called a* universally composable (UC) commitment scheme*. If the scheme is secure in the presence of an adaptive adversary, we call it a* UC adaptively secure commitment scheme*.*

UC adaptively secure commitment schemes achieve the following guarantees.

Proposition 1. *Let π be a UC adaptively secure commitment scheme, let $\mathcal{A}$ be an adversary and let $\mathcal{S}_\pi$ be the corresponding simulator. Then we have the following properties.*

- *The scheme is* hiding *and* binding*.*
- Extraction: *Let P_i be any corrupted party and let T be the transcript of the communication between the adversary and the simulator during a successful Commit phase where P_i commits to a value b (which might be unknown to P_i) towards an honest party P_j. Then the simulator can extract b from T ewnp.*
- Equivocability: *Let P_i be any corrupted party, b be any value and $\mathcal{T}'_b$ be the distribution of the messages from honest parties during the Commit phase in the real execution, where an honest party P_j commits to b towards P_i. Then —without having any information about b— the simulator can efficiently sample messages T from a distribution $\mathcal{T}$ such that $\mathcal{T}$ is computationally indistinguishable from $\mathcal{T}'_b$. Furthermore, upon receiving an Open-message for b from the ideal functionality, the simulator can open the commitment given by T to b, that is it can simulate the messages from honest parties in the Opening phase of the commitment given by T to b in a computationally indistinguishable way.*
- Adaptiveness: *If at any point, the adversary corrupts a party P_i, the simulator can provide computationally indistinguishably distributed information about the internal state of P_i (using the information received from the ideal functionality upon corruption of P_i) and can patch the information sent to the adversary in a computationally indistinguishable way to the information that the simulator receives from the ideal functionality F.*

Proof. The proposition follows directly from the definition of the ideal functionality, the definition of UC commitment schemes and the definition of adaptive security.

C Details on Threshold Homomorphic Encryption Schemes

In this section, we define threshold homomorphic encryption schemes. Our definition is based on the definitions given in [CDN00] and [FPS01]. We use the notation from [CDN00] and [DN03].

Definition 9. *A threshold homomorphic encryption scheme is a tuple (*KeyGen*,* Enc*,* $\mathsf{DecShare}$*,* Comb*) such that the following holds.*

- KeyGen *is a probabilistic algorithm that takes a security parameter κ, the number of parties n and the threshold parameter t as input and outputs a uniformly distributed tuple $(pk, sk_1, \ldots, sk_n)$ where the public key pk is given to all parties and the secret key sk_i is given to P_i for all $i \in \{1, \ldots, n\}$.*
- Enc *is an efficient probabilistic non-interactive algorithm that takes as input a public key pk and a message m and outputs an encryption $\mathsf{Enc}_{pk}(m)$ of m. If we want to specify the randomness r used in the execution of the algorithm, we write $\mathsf{Enc}_{pk}(m, r)$.*
- $\mathsf{DecShare}$ *is an efficient algorithm that takes as input an index $i \in \{1, \ldots, n\}$, the public key pk, the secret key sk_i and a ciphertext c and outputs a decryption share c_i and a proof that c_i is correctly computed using i, pk, c and sk_i. We require that from the output of the* $\mathsf{DecShare}$ *algorithm, a computationally bounded adversary does not learn anything about the secret input used, even under parallel composition.*
- Comb *is an efficient algorithm that takes as input the public key pk, a ciphertext c and pairs (c_i, p_i) where each pair has a different index. The algorithm outputs a message m or fails.*

Furthermore, the scheme needs to satisfy the following properties.

- Correctness (this corresponds to the t-robust property in [FPS01]): *Let $(pk, sk_1, \ldots, sk_n)$ be any output of the* KeyGen *algorithm, m be any message in the message space R_{pk}, r be any randomness from the randomness space, $\mathcal{P}' \subseteq \mathcal{P}$ be any set of parties of size at least $t + 1$ and $\{(c_i, p_i^c)\}_{P_i \in \mathcal{P}'}$ be any set of values where the second component is a valid proof that the first component is a correctly computed decryption share for $i, pk, \mathsf{Enc}_{pk}(m, r)$ and sk_i. We require that even in the presence of an active adversary that corrupted up to t parties before the* KeyGen *algorithm, $\mathsf{Comb}(pk, \mathsf{Enc}_{pk}(m, r), \{(c_i, p_i^c)\}_{i=1,\ldots,n}) = m$ where $\{(c_i, p_i^c)\}_{P_i \notin \mathcal{P}'}$ are arbitrary values or $\perp$ (missing values are set to $\perp$ by default). In particular, this implies that for any set $\mathcal{P}' \subseteq \mathcal{P}$ of parties of size at least $t + 1$, we have $\mathsf{Comb}(pk, \mathsf{Enc}_{pk}(m, r), \{\mathsf{DecShare}(i, pk, sk_i, \mathsf{Enc}_{pk}(m, r))\}_{P_i \in \mathcal{P}'}, \{(c_i, p_i^c)\}_{P_i \notin \mathcal{P}'}) = m$, where again $\{(c_i, p_i^c)\}_{P_i \notin T}$ are arbitrary values or $\perp$.*
- Threshold semantic security: *Let κ be the security parameter and let $(pk, sk_1, \ldots sk_n)$ be the output of the* KeyGen *algorithm on input (κ, n, t). Consider any efficient probabilistic adversary $\mathcal{A}$ that on input κ, set $C \subseteq \mathcal{P}$ of size at most t, public key pk and secret keys $\{sk_i\}_{i \in C}$ of the parties in C outputs two messages m_0 and m_1. Let $c_0 = \mathsf{Enc}_{pk}(m_0)$ be an encryption of m_0 and $c_1 = \mathsf{Enc}_{pk}(m_1)$ an encryption of m_1. We denote the distribution of c_0 over $\kappa \in \mathbb{N}$ and $C \subseteq \mathcal{P}$ with $|C| \leqslant t$ by $X_0(\kappa, C)$ and the distribution of c_1 over the same set as X_0 (namely κ and C) by $X_1(\kappa, C)$. We require that the distributions $\{X_0(\kappa, C)\}_{\kappa \in \mathbb{N}, C \subseteq \mathcal{P}}$ and $\{X_1(\kappa, C)\}_{\kappa \in \mathbb{N}, C \subseteq \mathcal{P}}$ are computationally indistinguishable.*
- Message ring: *Let R_{pk} be the message space for a public key pk. Then $(R_{pk}, +_{pk}, \cdot_{pk}, 0_{pk}, 1_{pk})$ is a commutative ring. Furthermore, knowing pk, it is possible to do computations in R_{pk} efficiently.*
- Randomness space: *The domain for the randomness used in the probabilistic algorithm* Enc *is a group with group operation $\boxplus_{pk}$.*
- $+_{pk}$-homomorphic: *There exists an efficient algorithm that takes as input the public key pk and two encryptions $\mathsf{Enc}_{pk}(m_1, r_1)$ and $\mathsf{Enc}_{pk}(m_2, r_2)$ of m_1 using randomness r_1 respectively of m_2 using randomness r_2 and computes the uniquely determined encryption $\mathsf{Enc}_{pk}(m_1 +_{pk} m_2, r_1 \boxplus_{pk} r_2)$ of $m_1 +_{pk} m_2$ using randomness $r_1 \boxplus_{pk} r_2$. We denote a call to this algorithm with inputs $pk, \mathsf{Enc}_{pk}(m_1)$ and $\mathsf{Enc}_{pk}(m_2)$ by $\mathsf{Enc}_{pk}(m_1) \oplus_{pk} \mathsf{Enc}_{pk}(m_2)$.*

– Multiplication by constant: *There exists an efficient algorithm that takes as input the public key pk, a constant $c \in R_{pk}$ and an encryption $\mathsf{Enc}_{pk}(m)$ of m and computes a uniquely determined encryption $\mathsf{Enc}_{pk}(c \cdot_{pk} m)$ of $c \cdot_{pk} m$. We denote a call to this algorithm with inputs pk, c and $\mathsf{Enc}_{pk}(m)$ by $c \odot_{pk} \mathsf{Enc}_{pk}(m)$.*
– Proof of plaintext knowledge: *Let Q_{pk} be the binary predicate such that $Q_{pk}((m,r),y) = true$ if and only if $y = \mathsf{Enc}_{pk}(m,r)$. We require that for every public key pk, there exists an efficient patchable zero-knowledge proof of knowledge for Q_{pk} with constant communication complexity.*
– Proof of correct multiplication: *Similar to the previous property, let P_{pk} be the binary predicate such that $P_{pk}((\alpha, r_1, r_2),(y_1,y_2,y_3)) = true$ if and only if $y_1 = \mathsf{Enc}_{pk}(\alpha, r_1)$ and $y_3 = (\alpha \odot_{pk} y_2) \oplus_{pk} \mathsf{Enc}_{pk}(0_{pk}, r_2)$. We require that for every public key pk, there exists an efficient patchable zero-knowledge proof of knowledge for P_{pk} with constant communication complexity.*

Remark 6. As noted in [CDN00] we have that if the additive group of the message space R_{pk} can be spanned by 1_{pk}, then the multiplication by constant property is a direct consequence of the $+_{pk}$-homomorphic property.

Furthermore, the Blindable property in [CDN00] follows directly from the slightly modified version of the $+_{pk}$-homomorphic property above:

Proposition 2. *Let* (KeyGen, Enc, DecShare, Comb) *be any threshold homomorphic encryption scheme. Then there exists an efficient probabilistic algorithm* Blind *that takes as input a public key pk and an encryption $y = \mathsf{Enc}_{pk}(m)$ of a message m and outputs an encryption $z = \mathsf{Enc}_{pk}(m,r)$ of m, where r is a uniformly random element from the randomness space, and the randomization factor r_1. More precisely, we define $\mathsf{Blind}(pk,y) := (z, r_1)$, where $z = y \oplus_{pk} \mathsf{Enc}_{pk}(0_{pk}, r_1)$ and r_1 is a uniformly randomly sampled element of the randomness space. If the public key pk is clear from the context, we omit it as an input to the algorithm.*

Proof. Let r_1 be uniformly random in the randomness space and define $z = y \oplus_{pk} \mathsf{Enc}_{pk}(0_{pk}, r_1)$ as above. Then by the $+_{pk}$-homomorphic property of the encryption scheme we have that $z = \mathsf{Enc}_{pk}(m +_{pk} 0_{pk}, r_0 \boxplus_{pk} r_1)$ where r_0 is the randomness such that $y = \mathsf{Enc}_{pk}(m, r_0)$. Clearly $m +_{pk} 0_{pk} = m$ and $r_0 \boxplus_{pk} r_1$ is uniformly random because the randomness space is a group and r_1 is uniformly random in it. Hence, z satisfies the wanted properties.

C.1 Paillier Threshold Encryption Scheme

In this subsection, we present the Paillier encryption scheme (which was first introduced in [Pai99]) and a threshold version of it (introduced in [FPS01] and [DJ00]). The Paillier threshold version is an instantiation of the above definition. Our exposition of the original scheme and the threshold version closely follows the description in [FPS01]. For more details, we refer the reader to [FPS01], [DJ00] and [DN03].

Paillier Encryption Scheme Let us shortly recall the original Paillier encryption scheme.

Key Generation: Let $N = pq$ be an RSA-modulus and let g be an integer such that $\mathrm{ord}(g) = \alpha N \mod N^2$. Let λ be the Carmichael lambda function in $\mathbb{Z}^*_{N^2}$. We define the public key to be the pair (N, g) and the secret key to be $\lambda(N)$.
Encryption: Let m be a message from the message ring $\mathbb{Z}_N$. To encrypt m, choose a uniformly random element $r \in \mathbb{Z}^*_N$ and define the ciphertext to be $c = g^m r^N \mod N^2$.
Decryption: Let $W = \{w \in \mathbb{N}_0 \mid w < N^2 \text{ and } w = 1 \mod N\}$ and define the function $L \colon W \to \mathbb{N}_0$ by $L(w) = \frac{w-1}{N}$ (for the well-definedness of the function L, see [FPS01]). To

decrypt a ciphertext c, compute the message $m = \frac{L(c^{\lambda(N)} \mod N^2)}{L(g^{\lambda(N)} \mod N^2)} \mod N$. Correctness of the decryption algorithm can be checked using two properties of the Carmichael lambda function, namely that for all elements $w \in \mathbb{Z}_{N^2}^*$, we have $w^{\lambda(N)} = 1 \mod N$ and $w^{N\lambda(N)} = 1 \mod N^2$.

For a security analysis of this scheme, see [Pai99].

Paillier Threshold Version We continue with the threshold version. Let n be the number of parties. Define $\Delta = n!$. The key generation algorithm uses the Shamir sharing scheme to share the secret key among all parties. This secret sharing scheme ensures that all sets of at least $t+1$ parties can collectively reconstruct the secret key while no set of up to t parties have any information about the secret key. For an explanation of the Shamir sharing scheme, we refer the reader to Section 3.1 of [FPS01] and to [Sha79] where it was first introduced.

Key Generation: Let N be an integer such that $N = pq$ for $p = 2p' + 1$ and $q = 2q' + 1$ for p' and q' prime and such that $\gcd(N, \varphi(N)) = 1$. We define $\chi = p'q'$ and we choose $(\beta, \gamma, \delta) \in (\mathbb{Z}_N^*)^3$ randomly. Next, we set $g = (1+N)^\gamma \cdot \delta^N \mod N^2$ and we share the secret key $\beta \cdot \chi$ with the Shamir sharing scheme by setting $\ell_0 = \beta \cdot \chi$, randomly choosing coefficients $\ell_i \in \{0, \dots, \chi N - 1\}$ for $i \in \{1, \dots, t\}$, defining the polynomial $\ell(x) = \sum_{i=0}^t \ell_i x^i$ and providing the share $sk_i = \ell(i) \mod \chi N$ of the secret key $\beta \cdot \chi$ to party P_i for $i \in \{1, \dots, n\}$. The public key is $pk = (g, N, \theta = L(g^{\chi\beta}) = \gamma\chi\beta \mod N)$. Finally, let $VK = v$ be a generator of the cyclic group of squares in $\mathbb{Z}_{N^2}^*$ and set the verification keys to $VK_i = v^{\Delta sk_i} \mod N^2$ (the verification keys will be used in zero-knowledge proofs in the decryption process).
Encryption: Let m be a message from the message ring $R_{pk} = \mathbb{Z}_N$. To encrypt m, choose a uniformly random $r \in \mathbb{Z}_N^*$ and define the ciphertext to be $c = g^m r^N \mod N^2$.
Decryption Shares: Every party P_i sends its decryption share $c_i = c^{2\Delta sk_i} \mod N^2$ to everyone and proves that its share is correct, that is that $c^{4\Delta} \mod N^2$ and $v^\Delta \mod N^2$ raised to the same power sk_i yield c_i^2 and v_i. For a description of the proof used to show that a decryption share is correct, see Section 3.2 of [FPS01].
Combination: If at least $t+1$ decryption shares are proven to be correct, consider a set U of $t+1$ correct decryption shares. Define the Lagrange coefficients $\mu_j^U = \Delta \cdot \frac{\prod_{j' \in U \setminus \{j\}} (-j')}{\prod_{j' \in U \setminus \{j\}} (j-j')} \in$ for all $j \in U$ and compute the message $m = L(\prod_{j \in U} c_j^{2\mu_j^U} \mod N^2) \cdot \frac{1}{4\Delta^2\theta} \mod N$. Else (if less than $t+1$ decryption shares are proven to be correct), the algorithm fails.

We assume without proof that for the Paillier threshold encryption scheme, the "proof of plaintext knowledge" property, the "proof of correct multiplication" property, the "proof of compatible commitment" property and Remark 2 hold. All the remaining properties of Definition 9 are proven to be satisfied by the Paillier threshold encryption scheme in [FPS01]. Furthermore, the Paillier threshold encryption scheme satisfies the Patch and Lagrange arguments properties. In fact, for the Patch property, we have that r_E can be computed efficiently as $r_E = r_K^{-x} \cdot r_0 \mod N$, where N is the Paillier public key. For the Lagrange arguments property we have that if $N = pq$ is such that $n < \min\{p, q\}$ and if we set $\alpha_i = i$ for all $i \in \{1, \dots, n\}$, then the α_i's satisfy the condition of the property. In fact, since $n < \min\{p, q\}$, we have for all $(i, j) \in \{1, \dots, n\}^2$ that $|\alpha_i - \alpha_j| \in \{0, \dots, n\}$ and thus $\gcd\{|\alpha_i - \alpha_j|, N\} = 1$. Hence, $|\alpha_i - \alpha_j|$ is invertible in $\mathbb{Z}_N$ and thus so is $\alpha_i - \alpha_j$. Therefore, we will assume from now on, that N is sufficiently larger than n so that $n < \min\{p, q\}$. We can thus conclude that the described Paillier threshold encryption scheme is an instantiation of Definition 9 that also satisfies the additional properties we need for our protocol.

D Details of the Subprotocols

D.1 Reliable Consensus

In this subsection, we discuss reliable consensus which is a weaker version of asynchronous consensus. Our protocol is a slight modification of the A-Cast protocol in [Bra84]. We follow the exposition of the A-Cast protocol in [CR98]. In this whole subsection, an honest party is a party that is never corrupted by the adversary and remains honest during the whole execution of the protocol.

Definition 10. *Consider an n-party protocol π, where each party P_i potentially eventually (possibly not even before it terminates the protocol π) has a secret input m_i that can be influenced by other protocols running in parallel. We say that π is a* t-resilient reliable asynchronous consensus protocol *if for all active adaptive adversaries corrupting up to t parties and for all inputs the following is satisfied:*

– Persistency*: If the honest parties all eventually have the same secret input m (that is if there is pre-agreement on m), then all honest parties output m and terminate.*
– Consistency*: If an honest party P_i outputs a value y and terminates, then all honest parties output y and terminate.*

Reliable consensus does not require that honest parties eventually terminate. It is allowed that the protocol runs forever if there is no pre-agreement.

Protocol RC

Each party P_i acts as follows:

```
1: On input the secret input $m_i$, if $P_i$ has not sent an (echo, $m'$)-message for some $m'$ (potentially equal to
   $m_i$) to all parties yet, it sends (echo, $m_i$) to all parties.
2: On input (echo, $m'$) from $n - t$ parties, if $P_i$ has not sent (ready, $m'$) to all parties yet, it does so.
3: On input (ready, $m'$) from $t + 1$ parties, if $P_i$ has not sent (ready, $m'$) to all parties yet, it does so.
4: On input (ready, $m'$) from $n - t$ parties, $P_i$ outputs $m'$ and terminates.
```

Theorem 4. *The* RC *protocol is a t-resilient reliable asynchronous consensus protocol for $t < n/3$ communicating $\mathcal{O}(n^2\kappa)$ bits, where κ is the size any party's secret input.*

The theorem can be proven along the lines of the proof in [Bra84] for the A-Cast protocol.

D.2 A-Cast

In this subsection, we introduce the so-called A-Cast or reliable broadcast (RBC) which is an asynchronous broadcast protocol that was originally introduced in [Bra84]. We follow the exposition in [CR98]. In the whole subsection (as in the previous subsection), an honest party is a party that is never corrupted by the adversary and remains honest during the whole execution of the protocol.

Definition 11. *Consider an n-party protocol π, where party P_S (the sender) has input m (the message that it wants to broadcast) and all other parties have input P_S. We say that π is a* t-resilient reliable asynchronous broadcast protocol *if for all active adaptive adversaries corrupting up to t parties and for all inputs the following is satisfied:*

– Validity: *If P_S is honest, then all honest parties output m and terminate.*
– Consistency: *If an honest party P_i outputs a value y and terminates, then all honest parties output y and terminate.*

Reliable broadcast (like reliable consensus) does not require that honest parties eventually terminate. It is allowed that the protocol runs forever if the sender is corrupted during or before the execution.

Protocol RBC [Bra84]

```
1: $P_S$ sends $m$ to every party. Denote the value that $P_i$ receives from $P_S$ by $m_i$.
2: The parties run the RC protocol with secret input $m_i$ for all honest parties $P_i$ (malicious parties can choose
   an arbitrary input).
```

Theorem 5. *The* RBC *protocol is a t-resilient reliable asynchronous broadcast protocol for $t < n/3$ communicating $\mathcal{O}(n^2\kappa)$ bits, where κ is the size of the sender's input.*

Proof. First we will prove that validity holds and then we will show that the protocol achieves consistency.

- Validity: Suppose P_S is honest. Then P_S sends m to all parties and all honest parties eventually receive m and use it as their secret input in the run of the RC protocol. Hence, by the persistency property of reliable consensus, we can conclude that all honest parties output m and terminate.
- Consistency: Suppose that there exists an honest party P_i that outputs y and terminates. The consistency property of the reliable consensus protocol ensures that all other honest parties also output y and terminate and thus we can conclude.

For the communication complexity we have that the RC protocol communicates $\mathcal{O}(n^2\kappa)$ bits (see Theorem 4) and hence, it is easy to see that the RBC protocol also communicates $\mathcal{O}(n^2\kappa)$ bits.
Our proof works for an adaptive adversary corrupting at most t parties because the reasoning above is independent of which parties the adversary corrupts at what point in time (we only talk about parties that remain honest during the whole execution of the protocol).

The RBC protocol satisfies the following property.

Proposition 3. *Let m_1 and m_2 be any two computationally indistinguishably distributed messages. Then, even in the presence of an active adaptive adversary corrupting up to $t < n/3$ parties, an execution of the* RBC *protocol where P_S has input m_1 is computationally indistinguishably distributed from an execution of the* RBC *protocol where P_S has input m_2.*

The proposition can be proven by reduction.

Remark 7. In [Pat11], Patra presents a t-resilient reliable asynchronous broadcast protocol Multi-Valued-Acast for $t < n/3$ which achieves linear communication complexity for messages of size $\Omega(n^3 \log(n))$. More precisely, for messages of size ℓ, the protocol communicates $\mathcal{O}(n\ell + n^4 \log(n))$ bits. We use the Multi-Valued-Acast protocol in selected steps of our MPC protocol to reduce the communication complexity.

D.3 Byzantine Agreement

In this subsection, we briefly introduce Byzantine agreement. The following definition is taken from [CR98].

Definition 12. *Consider an n-party protocol π, where each party P_i eventually has a secret input that can be influenced by other protocols running in parallel. The protocol π is a $(1-\epsilon)$-*terminating, t-resilient Byzantine agreement protocol (BA) *if for all active adaptive adversaries corrupting up to t parties and for all inputs the following is satisfied:*

- Termination: *With probability $1-\epsilon$ all parties terminate.*
- Correctness: *All honest parties that terminate have the same output. Moreover, if all parties that remain honest during the whole execution of the protocol have the same input m, then all these parties output m and terminate.*

For $t < n/3$, $(1-\epsilon)$-terminating, t-resilient asynchronous Byzantine agreement with expected communication complexity $\mathcal{O}(n^2)$ is achieved in [MMR15].

D.4 ACS

This subsection is devoted to the introduction of the "agreement on a common subset"-protocol which was first introduced by Canetti in [Can95]. Our exposition is based on Section 4 of [BKR94]. For more details, we refer the reader to [BKR94] and [Can95].

Definition 13. *Let $\mathcal{P}$ be a set of n parties and let Q be a property that can be influenced by multiple protocols running in parallel. Every party $P_i \in \mathcal{P}$ can decide for every party $P_j \in \mathcal{P}$ based on the protocols running in parallel whether P_j satisfies the property towards P_i or not. If it does, we say P_i* likes *P_j* for *Q or simply P_i* likes *P_j if the property Q is clear from the context. We require that once a party likes another party, it cannot unlike it. Such a property Q is called an* ACS property *if for every pair of uncorrupted parties $(P_i, P_j) \in \mathcal{P}^2$ we have that P_i will eventually like P_j.*

While the definition guarantees that all honest parties eventually like each other, a dishonest party can also satisfy Q towards an honest party and then the honest party will like the dishonest party. Furthermore, it is possible for a dishonest party to satisfy Q towards an honest party P_i, but not towards a different honest party P_j.
Additionally, the relation goes only one way in terms of if a party P_i knows that a party P_j does not satisfy Q, it cannot dislike P_j. It will simply never like P_j.
Moreover, as Q can depend on protocols running in parallel, it is not required that at the beginning of the current protocol, all parties already know which parties they like. This can be determined later on.
A simple example of such a property Q is if all parties send a "Present"-message to all other parties to announce that they are in the protocol. A party P_i likes P_j as soon as it receives a "Present"-message from P_j. In this work, Q will usually be a property like "reliable broadcast of an input terminated" or "zero-knowledge proof is accepted".

Definition 14. *The* "agreement on a common subset"-protocol (ACS) *is an adaptively secure asynchronous protocol that takes as input an ACS property Q and outputs a set S of parties of size at least $n - t$ such that for each $P_i \in S$ there exists at least one honest party P_j that likes P_i for Q. The protocol ensures that all honest parties terminate with high probability and agree on S.*

Observe that while all parties in S are liked by at least one honest party, there might be parties (possibly even honest ones) that are liked by an honest party but are not in S. This is not in contradiction to the guarantee given by the ACS protocol.
To achieve ACS, we run n instances of $(1-\epsilon)$ terminating, t-resilient Byzantine agreement (BA), one for each party $P_j \in \mathcal{P}$. If the output of BA_j is 1, we add P_j to S, else we don't.

Protocol ACS

Every P_i does the following:

```
1: For every party $P_j$ that $P_i$ likes, $P_i$ inputs 1 to $\text{BA}_j$.
2: As soon as $n - t$ BA's terminated on output 1, $P_i$ inputs 0 to all the BA's that it hasn't given input yet.
3: After all BA's terminated, $P_i$ defines $S$ to be the set of parties whose BA terminated on 1, that is
   $S = \{P_j$ such that the output of $\text{BA}_j$ is $1\}$.
4: Party $P_i$ outputs $S$.
```

Proposition 4. *The* ACS *protocol is indeed an ACS protocol according to Definition 14. The protocol communicates $\mathcal{O}(n^3)$ bits.*

The proposition can be proven along the lines of the proof of Lemma 2 in [BKR94] (replace $2t + 1$ by $n - t$ in the whole proof).

D.5 Decryption protocols

Private Decryption The private decryption protocol PrivDec takes the public key pk, a ciphertext c and a party P as public input and the secret keys $sk_1, \ldots, sk_n$ as private inputs. The protocol has no public nor private output for all parties except for P, who privately outputs the plaintext underlying c. This section is along the lines of [BTH08].

Protocol PrivDec

```
1: Every party $P_i$ computes $(c_i, p_i^c) = \mathsf{DecShare}(i, pk, sk_i, c)$, sends $(c_i, p_i^c)$ to $P$ and terminates.
2: As soon as $P$ has received at least $t+1$ pairs $(c_k, p_k^c)$ from distinct parties $P_k$ such that $p_k^c$ is a valid proof
   for $c_k$ from $P_k$, $P$ computes $m = \mathsf{Comb}(pk, c, \{(c_k, p_k^c)\}_{k \in \{1,\ldots,n\}})$, where $P$ sets all the values that is has
   not received to $\perp$. Then $P$ outputs $m$.
```

Lemma 3. *Every party that is not corrupted before or during the execution terminates the* PrivDec *protocol. Furthermore, if P is honest at the end of the protocol, then its output m is the correct decryption of c even in the presence of an adaptive adversary actively corrupting up to $t < n/3$ parties. The protocol has communication complexity $\mathcal{O}(n\kappa)$.*

Proof. In this whole proof, an honest party is a party that is never corrupted by the adversary and remains honest during the whole execution of the protocol.
Termination: Clearly all honest parties apart from P terminate as they only need to compute a decryption share and send it to P. Furthermore, if P is honest, then it terminates since all honest parties send correct decryption shares. Hence, P eventually receives at least $n - t \geqslant t + 1$ correct decryption shares from distinct parties, runs Comb and obtains and outputs a message m.
Correctness: As we saw above, P eventually receives at least $t + 1$ correct decryption shares from distinct parties. Hence, thanks to correctness of the threshold homomorphic encryption scheme, we can deduce that P can compute the correct decryption m of c. If P is honest, then it computes and outputs m.
It is easy to see that the communication complexity is indeed $\mathcal{O}(n\kappa)$.
The proof works for an adaptive adversary corrupting at most t parties because the reasoning above is independent of which parties the adversary corrupts at what point in time (we only talk about parties that remain honest during the whole execution of the protocol).

Amortized Public Decryption The public reconstruction protocol PubDec takes the public key pk and $T = n - 2t$ ciphertexts $c_1, \ldots, c_T$ as public inputs and the secret keys $sk_1, \ldots, sk_n$ as private inputs. The protocol publicly outputs the plaintexts $m_1, \ldots, m_T$ underlying the ciphertexts $c_1, \ldots, c_T$. This section is along the lines of [CHP12] and [BTH08].

Protocol PubDec

```
1: Every party defines the polynomial $g(x) = \sum_{j=1}^{T} x^{j-1} \odot_{pk} c_j$ and computes $v_i = g(\alpha_i)$ for all $i \in \{1, \ldots, n\}$.
2: The parties use their secret keys to run $\mathsf{PrivDec}(P_i, v_i)$ for all $i \in \{1, \ldots, n\}$. Let $u_i$ be $P_i$'s private output
   from $\mathsf{PrivDec}(P_i, v_i)$ for all $i \in \{1, \ldots, n\}$.
3: Every party $P_i \in \mathcal{P}$ sends $u_i$ to all other parties.
4: Every party $P_i \in \mathcal{P}$ locally defines a set $\mathcal{P}'_i$ of parties and adds party $P_k$ to $\mathcal{P}'_i$ if $P_i$ received $u'_k$ from $P_k$.
   For $j = 0, 1, \ldots t$, as soon as $|\mathcal{P}'_i| \geqslant T + t + j$, $P_i$ applies an efficient algorithm PolyFind (for example the
   Berlekamp-Welch algorithm) on the points $\{(\alpha_k, u'_k)\}_{P_k \in \mathcal{P}'_i}$ to check whether there exists a polynomial
   $p$ of degree at most $T - 1$ such that at least $T + t$ of the input points lie on $p$. If this is the case, then
   PolyFind outputs this polynomial and $P_i$ outputs $m_1 = p_1, \ldots, m_T = p_T$, where $p(x) = \sum_{j=1}^{T} x^{j-1} \cdot_{pk} p_j$,
   and terminates. Otherwise, $P_i$ proceeds with iteration $j + 1$.
```

Lemma 4. *Every party that is not corrupted before or during the execution terminates the* PubDec *protocol and outputs the correct decryptions of $c_1, \ldots, c_T$ even in the presence of an* *adaptive adversary actively corrupting up to $t < n/3$ parties. The protocol has communication complexity $\mathcal{O}(n^2\kappa)$.*

Proof. In this whole proof, an honest party is a party that is never corrupted by the adversary and remains honest during the whole execution of the protocol.
Termination: (taken from [CHP12]) Since all honest parties participate in the $\mathsf{PrivDec}(P_i, v_i)$ protocols for all $i \in \{1, \ldots, n\}$, termination of the PrivDec protocol implies that all honest parties terminate steps 1–3. Next, define the polynomial $g'(x) = \sum_{j=1}^{T} x^{j-1} \cdot_{pk} m_j$. Since c_j is an encryption of m_j under pk for all $j \in \{1, \ldots, T\}$, the homomorphic property of the encryption scheme implies that $g(x)$ is an encryption of $g'(x)$ under pk for all $x \in R_{pk}$. In particular, this holds for $x = \alpha_k$ for all $k \in \{1, \ldots, n\}$. Hence, by the correctness of the PrivDec protocol and by definition of u_k, we have $u_k = g'(\alpha_k)$ for all honest parties P_k. Now, let P_i be an arbitrary honest party and let $\widehat{j}$ be the first iteration when all honest parties are in $\mathcal{P}'_i$ (note that every honest party eventually includes all honest parties in $\mathcal{P}'_i$ and since there are at most $n = T + 2t$ parties, we have $\widehat{j} \leqslant t$). Then, either PolyFind already found a polynomial in iteration j for $j < \widehat{j}$ and P_i terminated before iteration $\widehat{j}$ or in iteration $\widehat{j}$, $\mathcal{P}'_i$ is of size $T + t + \widehat{j}$ and contains $n - t = T + t$ honest parties. Hence, since g' is a polynomial of degree at most $T - 1$ and at least $T + t$ input points (namely the points from honest parties) lie on g', we can be sure that the PolyFind algorithm finds a polynomial and P_i terminates in step $\widehat{j}$. Hence, after at most $\widehat{j} \leqslant t$ iterations, P_i terminates. Note that if in an iteration j the PolyFind algorithm fails to find a polynomial that passes the checks, then P_i has not received all the $u'_k = u_k$'s from honest parties as otherwise the PolyFind algorithm would have succeeded (see above). Hence, if in an iteration the PolyFind algorithm fails to compute a suitable polynomial, then it is ok for P_i to proceed with the next iteration because it is guaranteed that P_i can eventually add at least one party to $\mathcal{P}'_i$ and as soon as P_i has all the u_k's from honest parties (i.e all honest parties are in $\mathcal{P}'_i$), it can terminate (and this will happen before the tth iteration ended).
Correctness: Let P_i be any honest party. As P_i terminates, it found a polynomial p of degree at most $T - 1$ and a set of parties $\mathcal{P}''_i$ of size at least $T + t$ such that P_i received a message u'_k from all $P_k \in \mathcal{P}''_i$ and $u'_k = p(\alpha_k)$ for all $P_k \in \mathcal{P}''_i$. Since there are at most t corrupted parties, at least T of the parties in $\mathcal{P}''_i$ are honest. In the proof for termination, we saw that for honest parties, $u'_k = u_k = g'(\alpha_k)$. Therefore, there exist T distinct elements α_k with $p(\alpha_k) = g'(\alpha_k)$. Since T points uniquely define a polynomial of degree at most $T - 1$ and both p and g' are polynomials of degree at most $T - 1$, we can conclude that $p = g'$ and P_i can correctly compute and output the messages $m_1, \ldots, m_T$ underlying the ciphertexts $c_1, \ldots, c_T$.
The claim about the communication complexity follows directly from the communication complexity of the PrivDec protocol.
Again, the proof works for an adaptive adversary corrupting at most t parties because the reasoning above is independent of which parties the adversary corrupts at what point in time (we only talk about parties that remain honest during the whole execution of the protocol).

Remark 8. In every execution of the PubDec protocol, each party runs the PolyFind algorithm up to $t + 1$ times. By using local player elimination, we can reduce the number of runs of the PolyFind algorithm in m instances of the PubDec protocol to $t + m$ per party (instead of $m(t + 1)$). More precisely, if in iteration j the run of the PolyFind algorithm of an honest party fails to output a polynomial that passes the checks, then at least $j + 1$ of the inputs must be wrong (otherwise the PolyFind algorithm would have succeeded). Since every party outputs a polynomial satisfying all the checks at latest in round t, each party can then detect which inputs were wrong and can locally eliminate the parties that sent those wrong values. In any future run of the PolyFind algorithm in the PubDec protocol, the party ignores the values sent from parties it locally eliminated (respectively, it does not include parties it locally eliminated in $\mathcal{P}'_i$).

Remark 9. By reduction and by Remark 2, we can deduce that for $c_1^1, \ldots, c_T^1$ and $c_1^2, \ldots, c_T^2$ two computationally indistinguishably distributed sets of T ciphertexts with computationally indistinguishably distributed sets of underlying plaintexts, an instance of the PubDec protocol with $(pk, c_1^1, \ldots, c_T^1)$ as public input (and $sk_1, \ldots, sk_n$ as private inputs) is computationally indistinguishably distributed to an instance of the PubDec protocol with $(pk, c_1^2, \ldots, c_T^2)$ as public input (and $sk_1, \ldots, sk_n$ as private inputs) even in the presence of an active adaptive adversary corrupting up to $t < n/3$ parties.

D.6 Multiplication

This subsection presents the multiplication protocol which is based on the MULTIPLICATION GATE in the Computation Phase protocol of [BTH08]. The protocol uses circuit randomization which was originally introduced in [Bea92].
Let $T = \lfloor \frac{n-2t}{2} \rfloor$. Our multiplication protocol processes up to T independent multiplication gates at the same time. To ensure independence of the gates, every run of the multiplication protocol only considers multiplication gates with a specific multiplicative depth.
The multiplication protocol takes as input T multiplication gates $m_1, \ldots, m_T$ with the same multiplicative depth, the $2T$ inputs $\{(X_i, Y_i)\}_{i\in\{1,\ldots,T\}}$ to the given multiplication gates and the T encrypted multiplication triples $\{(A_i, B_i, C_i)\}_{i\in\{1,\ldots,T\}}$ (encrypting the values $\{(a_i, b_i, a_i \cdot_{pk} b_i)\}_{i\in\{1,\ldots,T\}}$) associated with the given multiplication gates $m_1, \ldots, m_T$. We denote the plaintexts underlying the encryptions $\{(X_i, Y_i)\}_{i\in\{1,\ldots,T\}}$ by (x_i, y_i) for all $i \in \{1, \ldots, T\}$. The protocol publicly outputs T encryptions $\{Z_i\}_{i\in\{1,\ldots,T\}}$, where the underlying plaintexts z_i are equal to $x_i \cdot_{pk} y_i$ for all $i \in \{1, \ldots, T\}$.

Protocol Multiplication

```
1: Every party locally computes $X_i \ominus_{pk} A_i$ encrypting $x_i -_{pk} a_i$ and $Y_i \ominus_{pk} B_i$ encrypting $y_i -_{pk} b_i$ for all
   $i \in \{1, \ldots, T\}$ using the $+_{pk}$-homomorphic property of the encryption scheme.
2: The parties use their secret keys to run PubDec($\{X_i \ominus_{pk} A_i\}_{i\in\{1,\ldots,T\}}, \{Y_i \ominus_{pk} B_i\}_{i\in\{1,\ldots,T\}}$) and obtain
   $x_i -_{pk} a_i$ and $y_i -_{pk} b_i$ for all $i \in \{1, \ldots, T\}$.
3: Each party locally computes $E_i = \mathsf{Enc}_{pk}((x_i -_{pk} a_i) \cdot_{pk} (y_i -_{pk} b_i), e)$ for all $i \in \{1, \ldots, T\}$, where $e$ is
   the neutral element of the randomness space. Then, it computes $Z_i = E_i \oplus_{pk} [(x_i -_{pk} a_i) \odot_{pk} B_i] \oplus_{pk}$
   $[(y_i -_{pk} b_i) \odot_{pk} A_i] \oplus_{pk} C_i$ for all $i \in \{1, \ldots, T\}$.
4: Every party outputs $\{Z_i\}_{i\in\{1,\ldots,T\}}$.
```

Remark 10. 1. If $n-2t$ is odd, then the parties only input $n-2t-1$ ciphertexts to the PubDec protocol in step 2. In that case, the parties additionally give $\mathsf{Enc}_{pk}(0_{pk}, e)$ as input to the PubDec protocol, where e is again the neutral element of the randomness space, obtain the plaintext 0_{pk} as one of the outputs of PubDec and simply disregard it in all further steps.
2. If only $T' < T$ multiplication gates are input to the multiplication protocol (for example when there are less than T multiplication gates with the same multiplicative depth in a given circuit), then the parties execute the protocol normally doing all the computations for indices in $\{1, \ldots, T'\}$ instead of in $\{1, \ldots, T\}$ and adding the encryption $\mathsf{Enc}_{pk}(0_{pk}, e)$ to the inputs of the PubDec protocol $n - 2t - 2T'$ times (where e is again the neutral element of the randomness space).

The multiplication protocol achieves the following.

Proposition 5. *Let $m_1, \ldots, m_T$ be T multiplication gates with the same multiplicative depth and let $\{(A_i, B_i, C_i)\}_{i\in\{1,\ldots,T\}}$ be the encrypted multiplication triples associated with the gates $m_1, \ldots, m_T$. Furthermore, let $\{(X_i^1, Y_i^1)\}_{i\in\{1,\ldots,T\}}$ and $\{(X_i^2, Y_i^2)\}_{i\in\{1,\ldots,T\}}$ be two computationally indistinguishably distributed sets of $2T$ ciphertexts. Then, even in the presence of an active adaptive adversary corrupting up to $t < n/3$ parties, an execution of the multiplication protocol with $\{(X_i^1, Y_i^1)\}_{i\in\{1,\ldots,T\}}$ as inputs to the given gates is computationally indistinguishably distributed from an execution of the multiplication protocol with $\{(X_i^2, Y_i^2)\}_{i\in\{1,\ldots,T\}}$ as inputs to the given gates.*

Proof. Using reduction it is easy to see that step 1 is computationally indistinguishably distributed in both executions (even if the adversary corrupts a party during step 1).
For step 2, by reduction, we know that ciphertexts $(\{X_i^1 \ominus_{pk} A_i\}_{i\in\{1,\dots,T\}}, \{Y_i^1 \ominus_{pk} B_i\}_{i\in\{1,\dots,T\}})$ and $(\{X_i^2 \ominus_{pk} A_i\}_{i\in\{1,\dots,T\}}, \{Y_i^2 \ominus_{pk} B_i\}_{i\in\{1,\dots,T\}})$ are computationally indistinguishably distributed. Furthermore, by Lemma 1, we have that the plaintexts underlying $\{A_i\}_{i\in\{1,\dots,T\}}$ and the plaintexts underlying $\{B_i\}_{i\in\{1,\dots,T\}}$ are computationally uniformly randomly distributed. Therefore, the plaintexts underlying $\{X_i^1 \ominus_{pk} A_i\}_{i\in\{1,\dots,T\}}$, $\{Y_i^1 \ominus_{pk} B_i\}_{i\in\{1,\dots,T\}})$, $\{X_i^2 \ominus_{pk} A_i\}_{i\in\{1,\dots,T\}}$ and $\{Y_i^2 \ominus_{pk} B_i\}_{i\in\{1,\dots,T\}})$ are all computationally uniformly randomly distributed and thus, they are computationally indistinguishably distributed. By Remark 9, we can conclude that step 2 of the multiplication protocol is computationally indistinguishably distributed in both executions, even if the adversary corrupts a party.
As for step 1, a reduction argument shows that steps 3 and 4 maintain computational indistinguishability (even if the adversary corrupts a party during these steps).

Proposition 6. *The multiplication protocol communicates $\mathcal{O}(n^2\kappa)$ bits.*

E Proof of Lemma 1

We will prove each property separately. The proof is inspired by [BTH08], [DN03], [HN06] and [BHN08]. In this whole proof, an honest party is a party that remains honest during the whole execution of the protocol. Furthermore, we use uppercase letters to denote ciphertexts and the corresponding lowercase letters to denote the plaintexts underlying these ciphertexts.

- *Termination and Consistency:* By validity of reliable broadcast and completeness of zero-knowledge proofs, Q and Q' defined in the Triples protocol are indeed ACS properties according to Definition 13. Hence, the guarantees of the ACS primitive ensure that all honest parties eventually terminate the two ACS instances in the protocol and consistently output a set S, respectively S', of parties of size at least $n - t$, where each party in S, respectively S', is liked by at least one honest party for Q, respectively Q'. Therefore, for each $P_k \in S$, respectively $P_k \in S'$, at least one honest party terminated the Multi-Valued-Acast of $\{A_k^i\}_{i\in\{1,\dots,\ell\}}$, respectively $\{(B_k^i, C_k^i)\}_{i\in\{1,\dots,\ell\}}$. Consistency of reliable broadcast then implies that all honest parties eventually terminate the reliable broadcasts of $\{A_k^i\}_{i\in\{1,\dots,\ell\}}$, respectively $\{(B_k^i, C_k^i)\}_{i\in\{1,\dots,\ell\}}$ for all parties $P_k \in S$, respectively $P_k \in S'$, with the same output. Hence, all honest party can compute (A^i, B^i, C^i) as described in the protocol for $i \in \{1, \dots, \ell\}$, output the same triples and terminate.
- *Correctness:* This property can be proven using the properties of the ACS primitive, the definition of Q and a similar reasoning as for the multiplication protocol in [DN03].
- *Secrecy:* We start by showing that the plaintexts underlying A^i are unknown to the adversary for all $i \in \{1, \dots, \ell\}$. Since the set S is of size at least $n - t$ and there are at most $t < n - t$ corrupted parties, we have that there is at least one honest party in S. We choose an arbitrary honest party in S and denote its index by h. Let i be an arbitrary index in $\{1, \dots, \ell\}$. By definition of A^i we have $A^i = \bigoplus_{P_k\in S} A_k^i = A_h^i \oplus_{pk} \bigoplus_{P_k\in S\setminus\{P_h\}} A_k^i$. The guarantees of the ACS primitive and the definition of the ACS property Q ensure that for each $P_k \in S$, at least one honest party accepts the zero-knowledge proof of plaintext knowledge for A_k^i. Therefore, soundness of zero-knowledge proofs implies that with high probability the adversary knows the plaintext a_k^i underlying A_k^i for all corrupted parties P_k in S. By semantic security, the adversary does not know the plaintext a_h^i underlying A_h^i. Thus, the a_k^i's from corrupted parties $P_k \in S$ are independent of a_h^i and we can conclude that the plaintext underlying A^i is unknown to the adversary. Since i was an arbitrary index in $\{1, \dots, \ell\}$, this holds for all $i \in \{1, \dots, \ell\}$.
  The reasoning that the plaintexts underlying B^i are unknown to the adversary for all $i \in \{1, \dots, \ell\}$ is analogous.

To show that the adversary does not have more information about the plaintexts underlying C^i than that they are the multiplication of the plaintexts underlying A^i and B^i for all $i \in \{1, \ldots, \ell\}$, we observe the following. Let i be an arbitrary index in $\{1, \ldots, \ell\}$ and let k be any index such that $P_k \in S'$. If P_k is corrupted at any point of the $\mathsf{Triples}$ protocol, then the adversary knows b_k^i with high probability (by the guarantees of the ACS primitive, the definition of the ACS property Q' and soundness of zero-knowledge proofs). However, by semantic security and because a^i is unknown to the adversary (see above), the adversary still does not know anything more about c_k^i than that it is the multiplication of b_k^i and the unknown plaintext a^i underlying A^i (note that since P_k is in S', the guarantees of the ACS primitive, the definition of the ACS property Q' and soundness of zero-knowledge proofs ensure that with high probability C_k^i is indeed the multiplication of the plaintexts underlying B_k^i and A^i). If P_k remains honest during the whole execution of the protocol, then by semantic security, the adversary does not have more information about c_k^i than that it is the multiplication of the plaintexts underlying B_k^i and A^i which are both unknown to the adversary. Hence, since $C^i = \bigoplus_{P_k \in S'} C_k^i$, $B^i = \bigoplus_{P_k \in S'} B_k^i$ and all B_k^i's are only used to compute B^i, we can conclude that the adversary does not know anything about the plaintext underlying C^i but that it is the multiplication of the plaintexts underlying A^i and B^i. Finally, since i was an arbitrary index in $\{1, \ldots, \ell\}$, we can conclude that Secrecy holds.

- *Computational Uniform Randomness and Independence:* These two properties can be proven using similar arguments as for the Secrecy property and a similar reasoning as for the multiplication protocol in [DN03].
- *Privacy:* It is easy to see that this property holds because the simulator $\mathcal{S}_{\mathsf{Triples}}$ can perfectly imitate the honest parties (no party has any secret input to this protocol). Hence, also if the adversary decides to corrupt any party during the execution of the protocol, the simulator can give perfectly indistinguishably distributed information to the adversary.
- *Communication complexity:* The parties only communicate in steps 1, 2, 4 and 5 of the protocol. Furthermore, it is easy to see that the communication complexity of steps 1 and 4 are in the same complexity class and the communication complexity of steps 2 and 5 are in the same complexity class. Hence, the overall communication complexity is $\mathcal{O}(\{\text{communication complexity of step 1}\} + \{\text{communication complexity of step 2}\})$.
  In step 1, each party uses the $\mathsf{Multi\text{-}Valued\text{-}Acast}$ to broadcast ℓ values (of size κ) and proves bilaterally to each party plaintext knowledge of the ℓ broadcasted values. Hence, the communication complexity of step 1 is $\mathcal{O}(n^2\ell\kappa + n^5\log(n))$. By Appendix D.4, we know that the communication complexity of step 2 is $\mathcal{O}(n^3)$. Thus, we can conclude that the total communication complexity is $\mathcal{O}(n^2\ell\kappa + n^5\log(n))$.

F Protocol

The protocol we present uses a key generation oracle (KG) which sets up all the public and private keys used in our protocol, gives the keys to the entitled parties and provides public Lagrange arguments for all parties. We assume that the simulator has access to an efficient key generation algorithm (KGA) that computes a computationally indistinguishably distributed set of public and private keys and Lagrange arguments. Furthermore, we assume that the parties have access to an encoder and a decoder algorithm that transform values from the message space of the encryption scheme to $\{0,1\}^*$ and vice versa. We do not explicitly mention when the parties use the encoder and decoder algorithms. They are implicitly used whenever a transformation is necessary.
The description of the protocol follows the structure of the FuncEval_f Algorithm in [CDN00].

Protocol

Preparation Phase:

1: Every party P_i receives a security parameter κ, the number of parties n, a secret input $x_i \in \{0,1\}^*$ and a random string $b_i \in \{0,1\}^*$ as input. The adversary is given the inputs κ, n, a random string $b \in \{0,1\}^*$ and an auxiliary string $a \in \{0,1\}^*$.

2: The parties call the key generation oracle KG. Each P_i gets common inputs $pk, K, R, \{K_\nu\}_\nu, \{\alpha_i\}_{i\in\{1,\ldots,n\}}$ and the secret inputs $sk_i, \{K_\chi^i\}_\chi$, where $(pk, sk_1, \ldots, sk_n)$ is a uniformly random threshold encryption key, K is a uniformly random encryption of 1_{pk} under pk, R is a uniformly random encryption of 0_{pk} under pk, $\{K_\nu\}_\nu$ are the public keys used for the zero-knowledge proofs and the commitment scheme, $\{K_\chi^i\}_\chi$ are the private keys of P_i used for the zero-knowledge proofs and the commitment scheme and $\{\alpha_i\}_{i\in\{1,\ldots,n\}}$ are Lagrange arguments.

3: On input pk, every party computes the arithmetic circuit over R_{pk} corresponding to the function f evaluated on n inputs. We denote the gates in the circuit by $H^1_{pk}, \ldots, H^l_{pk}$.

4: Let c_M be the number of multiplication gates in the circuit. The parties execute the Triples protocol with input c_M and obtain a set of triples $\{(A_i, B_i, C_i)\}_{i\in\mathcal{I}}$, where $\mathcal{I}$ is the set of all indices of multiplication gates in the circuit.

Computation Phase:

1: Each party P_i commits to its secret input x_i towards every party P_j for all $j \in \{1,\ldots,n\}$ under the corresponding commitment key. For all $(i,j) \in \{1,\ldots,n\}$, let $C_{i\to j}$ be the commitment to x_i from P_i towards P_j and let (x_i, c_{ij}) be the opening information for $C_{i\to j}$.

2: Every party P_i chooses a uniformly random value r_{x_i} from the randomness space. The parties run the BrACS protocol from Appendix F with public input (pk, K) and secret input $(x_i, r_{x_i}, \{c_{ij}\}_{j\in\{1,\ldots,n\}}, \{C_{i\to j}\}_{j\in\{1,\ldots,n\}}, \{C_{j\to i}\}_{j\in\{1,\ldots,n\}})$ for every party P_i. The parties obtain as output a set S and encryptions $\{\mathsf{Enc}^K_{pk}(x_i)\}_{i:\, P_i\in S}$.

3: Evaluate the circuit as in [CDN00]: While there are gates that have not been evaluated yet, let $J \subseteq \{1,\ldots,l\}$ be the set of non-evaluated gates that are ready to be evaluated. Evaluate all gates in J in parallel by doing for every $j \in J$:

a) If H^j_{pk} is an input gate for a party $P_i \in S$, then every party sets $\mathsf{Enc}_{pk}(h_j) = \mathsf{Enc}^K_{pk}(x_i)$. If H^j_{pk} is an input gate for a party $P_i \notin S$, then every party computes $d \odot_{pk} K$ using the "Multiplication by constant" property of the encryption scheme and sets $\mathsf{Enc}_{pk}(h_j) = d \odot_{pk} K$, where d and is a default value.

b) If H^j_{pk} is a constant input gate for constant c, then every party sets $\mathsf{Enc}_{pk}(h_j) = c \odot_{pk} K$ by using the "Multiplication by constant" property of the encryption scheme.

c) If H^j_{pk} is an addition gate for $\mathsf{Enc}_{pk}(h_{j_1})$ and $\mathsf{Enc}_{pk}(h_{j_2})$, every party sets $\mathsf{Enc}_{pk}(h_j) = \mathsf{Enc}_{pk}(h_{j_1}) \oplus_{pk} \mathsf{Enc}_{pk}(h_{j_2})$ using the "$+_{pk}$-homomorphic" property of the encryption scheme.

d) If H^j_{pk} is a multiplication by a constant gate for values c and $\mathsf{Enc}_{pk}(h_{j_1})$, every party sets $\mathsf{Enc}_{pk}(h_j) = c \odot_{pk} \mathsf{Enc}_{pk}(h_{j_1})$ using the "Multiplication by constant" property of the encryption scheme.

e) If H^j_{pk} is a multiplication gate, the parties wait until all the multiplication gates with the same multiplicative depth as H^j_{pk} are ready to be evaluated. As soon as this is the case, the parties split these multiplication gates into blocks of $\lfloor \frac{n-2t}{2} \rfloor$ gates. For each block, the parties use the multiplication protocol from Appendix D.6 with the following input: the gates in the block, their input ciphertexts and the encrypted multiplication triples associated with the gates in the considered block. From this, the parties obtain the encrypted outputs of all the multiplication gates with the same multiplicative depth as H^j_{pk}.

Let $\mathsf{Enc}_{pk}(s)$ be the output of the evaluated circuit.

4: Every party P_i generates a uniformly random r_i from the message space R_{pk}. Each P_i commits to r_i towards every party P_j for all $j \in \{1,\ldots,n\}$ under the corresponding commitment key. For all $(i,j) \in \{1,\ldots,n\}$, let $B_{i\to j}$ be the commitment to r_i from P_i towards P_j and let (r_i, b_{ij}) be the opening information for $B_{i\to j}$.

5: Every party P_i chooses a uniformly random value $r^K_{r_i}$ from the randomness space. Parties run BrACS (see Appendix F) with public input (pk, K) and secret input $(r_i, r^K_{r_i}, \{b_{ij}\}_{j\in\{1,\ldots,n\}}, \{B_{i\to j}\}_{j\in\{1,\ldots,n\}}, \{B_{j\to i}\}_{j\in\{1,\ldots,n\}})$ for every party P_i. The parties get as output a set S' and encryptions $\{\mathsf{Enc}^K_{pk}(r_i)\}_{i:\, P_i\in S'}$.

6: Every party P_i chooses a uniformly random value $r^R_{r_i}$ from the randomness space. Parties run BrACS with public input (pk, R) and secret input $(r_i, r^R_{r_i}, \{b_{ij}\}_{j\in\{1,\ldots,n\}}, \{B_{i\to j}\}_{j\in\{1,\ldots,n\}}, \{B_{j\to i}\}_{j\in\{1,\ldots,n\}})$ for every party P_i. In this execution of the BrACS, we take a slightly modified ACS property Q, namely to all the conditions described in the BrACS protocol, we add that a party P_j only likes another party P_i if P_j likes P_i for the ACS property of the BrACS execution in step 5 (it is okay if P_j only likes P_i after the BrACS from step 5 terminated and input 0 to BA$_i$ in the ACS of step 5). The parties obtain as output a set S'' and encryptions $\{\mathsf{Enc}^R_{pk}(r_i)\}_{i:\, P_i\in S''}$.

```
7: Let $\hat{S} = S' \cap S''$. Let $I$ be the set of indices of the parties in $\hat{S}$ and let $\{\lambda_i\}_{i\in I}$ be the Lagrange coefficients
   of degree $|I|-1$ over $R_{pk}$ such that for any polynomial $g$ of degree at most $|I|-1$ we have $g(0_{pk}) =$
   $\sum_{i\in I} \lambda_i \cdot_{pk} g(\alpha_i)$ (precisely $\lambda_i = \prod_{j\in I, j\neq i}(0_{pk} - \alpha_j) \cdot_{pk} (\alpha_i - \alpha_j)^{-1}$ for all $i \in I$). Every party $P_i$ locally
   computes $\mathsf{Enc}_{pk}(s)' = \mathsf{Enc}_{pk}(s) \bigoplus_{pk, i\in I} (\lambda_i \odot_{pk} \mathsf{Enc}^R_{pk}(r_i))$.
8: The parties use their secret keys to run $\mathsf{PrivDec}(P_i, \mathsf{Enc}_{pk}(s)')$ for all $i \in \{1,\ldots,n\}$ and all parties obtain
   $s$.
9: The parties run the RC protocol from Appendix D.1 taking as secret input the value $s$ decrypted in the
   previous step (as soon as they obtain it).
```

BrACS In this subsection, we discuss the BrACS protocol used in our MPC protocol. The subprotocol takes as public input the public key pk of the encryption scheme and an encryption M (in our protocol and simulation this is sometimes an encryption of 1_{pk} and other times an encryption of 0_{pk}). The message encrypted by M is denoted by m. For each party P_i the protocol takes as secret input a message a_i, a randomness r_{a_i}, n values c_{ij} and $2n$ commitments $C_{j\to i}$ and $C_{i\to j}$ for $j \in \{1\ldots,n\}$. The $C_{j\to i}$'s represent commitments from P_j towards P_i. If P_i and P_j are both honest, (a_i, c_{ij}) is the opening information for the commitment $C_{i\to j}$ that P_j holds. The protocol publicly outputs a set S of parties and for each party $P_i \in S$ it publicly outputs an encryption of $a_i \cdot_{pk} m$.

Protocol BrACS

```
1: Every party $P_i$ generates an encryption of $a_i \cdot_{pk} m$ by computing $\mathsf{Enc}^M_{pk}(a_i, r_{a_i})$ and reliably broadcasts
   $\mathsf{Enc}^M_{pk}(a_i, r_{a_i})$ using the RBC protocol from Appendix D.2.
2: Every party $P_i$ uses the "proof of compatible commitment" property in Subsection 2.4 and proves to all
   $P_j$ for $j \in \{1,\ldots,n\}$ with instance $(\mathsf{Enc}^M_{pk}(a_i, r_{a_i}), C_{i\to j})$ and witness $(a_i, r_{a_i}, c_{ij})$.
3: Let $Q$ be the property such that a party $P_k$ satisfies $Q$ towards another party $P_j$ if and only if the reliable
   broadcast of $P_k$ in step 1 terminated for $P_j$ and the proof in step 2 was accepted by $P_j$. The parties run
   the ACS protocol with property $Q$ and obtain a set $S \subseteq \mathcal{P}$. Every $P_i$ waits until the reliable broadcast of
   all parties $P_k \in S$ terminated. Then each party outputs $S$ and for each $P_k \in S$ the value received from the
   terminated reliable broadcast.
```

Proposition 7. *The* BrACS *protocol achieves the following properties.*

a) The protocol terminates for all honest parties.

b) All parties agree on the set S and the encryptions of parties in S.

c) The set S is of size at least $n-t$.

d) Every honest party P_i in S succeeds to reliably broadcast a correct encryption $\mathsf{Enc}^M_{pk}(a_i)$ of $a_i \cdot_{pk} m$. This means that the reliable broadcast of $\mathsf{Enc}^M_{pk}(a_i)$ terminates for all honest parties and that at least one honest party P_j accepts the proof given by P_i in step 2, namely that P_i knows a preimage of $\mathsf{Enc}^M_{pk}(a_i)$ under (pk, M) and that the first component of this preimage is equal to the value P_i committed to with $C_{i\to j}$.
Furthermore, for every corrupted party P_i in S, the reliable broadcast of y of P_i in step 1 terminates for all honest parties and at least one honest party P_j accepts the proof (see above) given by P_i in step 2. Hence, with high probability, P_i knows values (a'_i, c'_{ij}) such that $y = \mathsf{Enc}^M_{pk}(a'_i)$ and (a'_i, c'_{ij}) is the opening information to $C_{i\to j}$.

The proof is straightforward and therefore omitted.

G Proof of Theorem 4.3

We will only informally prove the theorem. To do so, we construct the simulator.

G.1 Simulator

The description follows the structure of the simulator in [CDN00]. The simulator receives κ, n, a random string $c \in \{0,1\}^*$ and a as inputs. We denote the set of corrupted parties by C. Every time the adversary decides to adaptively corrupt a party P_i, P_i is added to C and the simulator sends a corruption request for P_i to the ideal functionality (we will not mention this explicitly in the description of the simulator).

Simulator $\mathcal{S}$

Preparation Phase:

1: Give κ and n to the ideal functionality. Then, give κ, n, b and a to the adversary, where b is a prefix of c. If the adversary decides to corrupt a party P_i during or after this step, receive x_i from the ideal functionality and give it to the adversary.

2: Run the key generation algorithm (KGA) and get a threshold key $(pk, sk_1, \ldots, sk_n)$, the public and private keys $\{K_\nu\}_\nu$ and $\{\{K_\chi^i\}_\chi\}_{i\in\{1,\ldots,n\}}$ used for the zero-knowledge proofs and the commitment scheme, the Lagrange arguments $\{\alpha_i\}_{i\in\{1,\ldots,n\}}$ and the uniformly random encryptions K (encrypts 1_{pk}) and R (encrypts 0_{pk}). Then, choose uniformly random r_K and r_R in the randomness space and redefine $K = \mathsf{Enc}_{pk}(0_{pk}, r_K)$ and $R = \mathsf{Enc}_{pk}(1_{pk}, r_R)$. Give $pk, K, R, \{K_\nu\}_\nu, \{\{K_\chi^i\}_\chi\}_{i\in C}, \{\alpha_i\}_{i\in\{1,\ldots,n\}}$ and $\{sk_i\}_{i\in C}$ to the adversary. If the adversary decides to corrupt a party P_i during or after this step, receive x_i from the ideal functionality and give x_i, sk_i and $\{K_\chi^i\}_\chi$ to the adversary.

3: Execute this step honestly on behalf of honest parties. If the adversary decides to corrupt a party P_i during or after this step, give all the information that P_i holds about the execution of this step to the adversary.

4: Run the Triples simulator $\mathcal{S}_{\mathsf{Triples}}$ described in Subsection 3.4. If the adversary decides to corrupt a party during or after this step, the Triples simulator $\mathcal{S}_{\mathsf{Triples}}$ handles what information is given to the adversary about this step.

Computation Phase:

1: For every corrupted party P_j and every honest party P_i, act honestly on behalf of P_i in the commit protocol that allows P_j to commit to a value towards P_i.
For each honest party P_i and every $j \in \{1,\ldots,n\}$, use the simulator of the commitment scheme to simulate a commitment $C_{i\to j}$ from P_i towards P_j (if P_j is honest, act honestly on behalf of P_j). If the adversary decides to corrupt P_i during or after this step, receive x_i from the ideal functionality and patch all the commitments from P_i towards any P_j that were already started or sent before P_i was corrupted to x_i using the adaptiveness property of the commitment scheme for all P_j for $j \in \{1,\ldots,n\}$. Give the information from the patching and all the information P_i holds about commitments made towards P_i to the adversary.

2: The steps in the BrACS protocol are simulated as follows.

1. Act honestly on behalf of honest parties in the reliable broadcasts with corrupted parties as senders. For every honest parties P_i, compute $D_i = \mathsf{Enc}_{pk}(0_{pk}, \widehat{r_{x_i}})$ using a uniformly random value $\widehat{r_{x_i}}$ and reliably broadcast D_i. If the adversary decides to corrupt P_i after the reliable broadcast, receive x_i from the ideal functionality and compute the randomness r_{x_i} such that $D_i = (x_i \odot_{pk} K) \oplus_{pk} \mathsf{Enc}_{pk}(0_{pk}, r_{x_i}) = \mathsf{Enc}_{pk}^K(x_i, r_{x_i})$ using pk, D_i, $\widehat{r_{x_i}}$, K, r_K and the Patch property of the encryption scheme. Then give r_{x_i} to the adversary.
2. Act honestly on behalf of honest parties in the zero-knowledge proofs from corrupted parties towards honest parties.
Use the simulator for zero-knowledge proofs for all zero-knowledge proofs with an honest party as prover. If the adversary decides to corrupt P_i during or after this step, receive x_i from the ideal functionality, compute r_{x_i} and patch the commitments as in the previous steps. For every $P_j \in \mathcal{P}$, let c'_{ij} be the opening information received from the patching of the commitment $C_{i\to j}$ to x_i. Give the instance $(\mathsf{Enc}_{pk}^K(x_i, r_{x_i}), C_{i\to j})$, the witness (x_i, r_{x_i}, c'_{ij}), the step $\tilde{t}$ in the zero-knowledge protocol when the adversary decides to corrupt P_i and the communication for the zero-knowledge proof up to $\tilde{t}$ to the Pat algorithm of the "proof of compatible commitment" zero-knowledge proof and Pat will output randomness ν_j that patches the proof. Give ν_j to the adversary for all $j \in \{1,\ldots,n\}$.
3. Run the ACS honestly with the same ACS property Q as in the protocol and obtain the set $S \subseteq \mathcal{P}$. Wait until all the reliable broadcasts of the corrupted parties in S terminate and set $\mathsf{Enc}_{pk}^K(x_i)$ to the output of the honest parties in the reliable broadcast with P_i as sender for all i such that $P_i \in S \cap C$. Extract x_i from a valid commitment received from P_i to x_i for all i such that $P_i \in S \cap C$. Give all x_i for $P_i \in S \cap C$ to the ideal functionality as inputs on behalf of the corrupted parties in $S \cap C$. Additionally give the set S as input and receive the output s from the ideal functionality. If the adversary decides to corrupt P_i during or after this step, patch the previous steps as described in the steps before and give all the information P_i holds about the running of the ACS in this step to the adversary.

3: Evaluate the circuit as described in step 3 of the protocol acting honestly on behalf of honest parties. If the adversary decides to corrupt a party P_i during or after this step, give all the information P_i holds about the execution of this step to the adversary.
Let $\mathsf{Enc}_{pk}(\hat{s})$ be the output of the evaluated circuit.
4: For every corrupted party P_j and every honest party P_i, act honestly on behalf of P_i in the commit protocol that allows P_j to commit to a value towards P_i.
For each honest party P_i, generate a uniformly random r'_i from the message space. For every $j \in \{1, \dots, n\}$, use the simulator of the commitment scheme to simulate a commitment $B_{i \to j}$ from P_i towards P_j (if P_j is honest, act honestly on behalf of P_j). If the adversary decides to corrupt P_i during or directly after this step, patch all the commitments from P_i towards any P_j that were already started or sent before P_i was corrupted to r'_i using the adaptiveness property of the commitment scheme for all P_j for $j \in \{1, \dots, n\}$. Give r'_i, the information from the patching and all the information P_i holds about commitments made towards P_i to the adversary.
5: The steps in the BrACS protocol are simulated very similarly to the BrACS in step 2.
   1. Act honestly on behalf of honest parties in the reliable broadcasts with corrupted parties as senders.
      For every honest party, compute $V_i = \mathsf{Enc}_{pk}(0_{pk}, \widehat{r^K_{r'_i}})$ using a uniformly random value $\widehat{r^K_{r'_i}}$ and reliably broadcast V_i. If the adversary decides to corrupt P_i directly after the reliable broadcast, patch all the commitments of P_i towards other parties in step 4 to r'_i using the adaptiveness property of the commitment scheme and compute the randomness $r^K_{r'_i}$ such that $V_i = (r'_i \odot_{pk} K) \oplus_{pk} \mathsf{Enc}_{pk}(0_{pk}, r^K_{r'_i}) = \mathsf{Enc}^K_{pk}(r'_i, r^K_{r'_i})$ using pk, V_i, $\widehat{r^K_{r'_i}}$, K, r_K and the Patch property of the encryption scheme. Then give r'_i, $r^K_{r'_i}$, the information from the patching of the commitments and all the information P_i holds about commitments made towards P_i in step 4 to the adversary.
   2. Act honestly on behalf of honest parties in the zero-knowledge proofs from corrupted parties towards honest parties.
      Use the simulator for zero-knowledge proofs for all zero-knowledge proofs with an honest party as prover. If the adversary decides to corrupt P_i during or directly after this step, compute $r^K_{r'_i}$ and patch the commitments as in the previous step. For every $P_j \in \mathcal{P}$, let b'_{ij} be the opening information received from the patching of the commitment $B_{i \to j}$ to r'_i. Give the instance $(\mathsf{Enc}^K_{pk}(r'_i, r^K_{r'_i}), B_{i \to j})$, the witness $(r'_i, r^K_{r'_i}, b'_{ij})$, the step $\tilde{t}$ in the zero-knowledge protocol when the adversary decides to corrupt P_i and the communication for the zero-knowledge proof up to $\tilde{t}$ to the Pat algorithm of the "proof of compatible commitment" zero-knowledge proof and Pat will output randomness ν'_j that patches the proof. Give r'_i, $r^K_{r'_i}$, the information from the patching of the commitments, all the information P_i holds about commitments made towards P_i in step 4 and ν'_j to the adversary for all $j \in \{1, \dots, n\}$.
   3. Run the ACS honestly with the same ACS property Q as in the protocol and obtain the set $S' \subseteq \mathcal{P}$. Wait until all the reliable broadcasts of the corrupted parties in S' terminate and set $\mathsf{Enc}^K_{pk}(r_i)$ to the output of the honest parties in the reliable broadcast with P_i as sender for all i such that $P_i \in S' \cap C$. Extract r_i from a valid commitment received from P_i to r_i for all i such that $P_i \in S' \cap C$. If the adversary decides to corrupt P_i during or directly after this step, patch as described in the previous step and give all the information P_i holds about the running of the ACS along with the information that the adversary would receive upon corrupting P_i directly after the previous step to the adversary.
6: Let I' be the set of indices of all the corrupted parties in S' and let r_i be the value they committed to in step 4. If $|I'| < t$, randomly choose $t - |I'| > 0$ honest parties in S', add their indices to I' and set $r_i = r'_i$ for the added parties. Then choose the unique polynomial p of degree less than or equal to t that at position α_i goes through r_i for $i \in I'$ and at position 0_{pk} goes through s, where s is the output received from the ideal functionality. Set $r_j = p(\alpha_j)$ for all honest parties P_j with $j \notin I'$.
For every honest party P_i, choose a uniformly random value $r^R_{r_i}$ from the randomness space. Patch the commitments of P_i towards other parties in step 4 to $p(\alpha_i)$ using the adaptiveness property and execute the BrACS (with the slightly modified ACS property) honestly for public input (pk, R) and secret input $(r_i = p(\alpha_i), r^R_{r_i}, \{b'_{ij}\}_{j \in \{1,\dots,n\}}, \{B_{i \to j}\}_{j \in \{1,\dots,n\}}, \{B_{j \to i}\}_{j \in \{1,\dots,n\}})$, where $\{(p(\alpha_i), b'_{ij})\}_{j \in \{1,\dots,n\}}$ is the opening information of the commitments $\{B_{i \to j}\}_{j \in \{1,\dots,n\}}$ to $p(\alpha_i)$ learned from the patching. From this execution obtain the set S'' and encryptions $\{\mathsf{Enc}^R_{pk}(r_i)\}_{P_i \in S''}$. If the adversary decides to corrupt P_i at any point during or after this step, then
   – if $i \in I'$, patch as described in step 5.3. and give all the information P_i holds about the execution of the current step along with all the information that the adversary would receive upon corrupting P_i directly after step 5.3 to the adversary.
   – if $i \notin I'$, patch steps 1–3 of the Computation phase as described in the simulator of those steps. Patch the commitments of P_i towards other parties in step 4 to $p(\alpha_i)$ and patch the execution of the BrACS in step 5 to the secret input $r_i = p(\alpha_i)$ in the same way as the execution of the BrACS in step 2 was

patched to x_i (see step 2 of the simulator). Finally give all the information along with the state of P_i in step 6 to the adversary.

7: For all honest parties, execute step 7 as described in the protocol (replacing $\mathsf{Enc}_{pk}(s)$ by $\mathsf{Enc}_{pk}(\hat{s})$). If the adversary decides to corrupt P_i during or after this step, give all the information P_i holds about the execution of this step to the adversary.

8: Run the n instances of the $\mathsf{PrivDec}$ protocol with input ciphertext $\mathsf{Enc}_{pk}(s)'$ acting honestly on behalf of honest parties. Again, if the adversary decides to corrupt a party P_i during or after this step, give all the information P_i holds about the execution of this step to the adversary.

9: Run the RC protocol acting honestly on behalf of honest parties. Again, if the adversary decides to corrupt a party P_i during or after this step, give all the information that P_i holds about the running of the RC protocol to the adversary.

G.2 Informal Proof of Security

Let us informally prove that the distributions of the simulation and the real execution are computationally indistinguishable (note that the distributions are in fact comparable). As in [CDN00], we will show that every step in the protocol is computationally indistinguishably distributed from the corresponding step in the simulation. Hence, after every step, computational indistinguishability is ensured which implies that after step 9 of the Computation phase, we can conclude that the distributions are computationally indistinguishable.

- **Preparation Phase:**
  1. Clearly the adversary receives the exact same inputs in both settings.
  2. Thanks to the guarantees given by the KGA, the threshold key, the Lagrange arguments and the public and private keys used for the zero-knowledge proofs and the commitment scheme are computationally indistinguishably distributed in the real execution and in the simulation. By the semantic security of the encryption scheme, we obtain that the distributions of K and R in the simulation (where they are redefined by the simulator) and the real execution are computationally indistinguishable.
  3. It is easy to see that this step keeps the computational indistinguishability.
  4. Lemma 1 shows that this step is perfectly indistinguishably distributed in the simulation and the real protocol.
- **Computation Phase:**
  1. By the equivocability property, the simulator of the commitment scheme gives a computationally indistinguishably distributed simulation of the commitments from honest parties. Furthermore, the simulator acts honestly on behalf of honest parties in commit protocols that allow a corrupted party to commit to a value towards an honest party. Thus, this step is computationally indistinguishably distributed from the corresponding one in the protocol. If the adversary corrupts a party P_i during or after this step, the simulator can perfectly patch the commitment to the value it receives from the ideal functionality (adaptiveness property) and give the information to the adversary.
  2. 1. Since the simulator acts honestly on behalf of honest parties in the reliable broadcasts with corrupted parties as senders, we have that the communication in these reliable broadcasts is computationally indistinguishably distributed. By semantic security of the encryption scheme, the inputs from honest parties to the reliable broadcasts where they act as senders are also computationally indistinguishably distributed. Hence, Proposition 3 applies and we can deduce that the communication in the reliable broadcasts with honest parties as senders is computationally indistinguishably distributed in the real execution and the simulation. If the adversary corrupts a party P_i during or after this step, the Patch property of the encryption scheme ensures that the simulator can perfectly patch the internal state of P_i to the value that it receives from the ideal functionality and give this information to the adversary. (More precisely, the simulator can compute the randomness r_{x_i} so that the reliably broadcasted encryption is equal to $\mathsf{Enc}_{pk}(x_i, r_{x_i})$ and r_{x_i} is uniformly random in the

randomness space.) Moreover, the simulator executes the reliable broadcasts honestly for all honest parties and hence, it can provide the adversary with computationally indistinguishably distributed information about the state of P_i in all reliable broadcasts.

   2. Since the simulator acts honestly on behalf of honest parties in zero-knowledge proofs with honest parties as verifiers, we have that the communication in these zero-knowledge proofs is computationally indistinguishably distributed. Thanks to the simulator of zero-knowledge proofs giving a computationally indistinguishably distributed view of the zero-knowledge proofs with honest parties as provers, the view of the adversary in the simulation of these zero-knowledge proofs is also computationally indistinguishably distributed from its view in the real execution. As before, if the adversary decides to corrupt party P_i during or after this step, the simulator can patch the commitments and the reliably broadcasted encryption and give this information as a witness to the Pat algorithm who will then patch the zero-knowledge proof. The adversary is given the patched internal state of P_i which is computationally indistinguishably distributed from the one in the real execution.
   3. As the ACS protocol is run honestly and since the ACS property Q is not susceptible to changes of the ciphertexts and the zero-knowledge proofs up to computational indistinguishability, we have that the distributions remain computationally indistinguishable. If the adversary corrupts P_i during or after this step, its view is computationally indistinguishably distributed from the real execution since this was true after the previous step and this step is run honestly and can be efficiently simulated using only information that the adversary knows (reduction).

3. Since the simulator evaluates the circuit as described in the protocol, we have the following. For gates of type a), it is easy to see that they maintain computational indistinguishability. For gates of type b), we can conclude by the semantic security of the encryption scheme. For gates of type c) and d), we can reduce the computational indistinguishability of the distribution of the output to the computational indistinguishability of the distributions of the inputs. Finally, for gates of type e), we obtain computational indistinguishability by Proposition 5. If the adversary decides to corrupt a party during the evaluation of a gate, then the simulator can give computationally indistinguishably distributed information for gates of type a)–d) because it acts honestly on behalf of honest parties and by reduction to the computational indistinguishability of the distributions of the inputs to the considered gate. For gates of type e) we can conclude by Proposition 5 and because the simulator acts honestly on behalf of honest parties.
   The outputs of the circuit $\mathsf{Enc}_{pk}(s)$ and $\mathsf{Enc}_{pk}(\hat{s})$ are computationally indistinguishably distributed because the inputs and outputs to all types of gates are computationally indistinguishably distributed.

4. Again, by the equivocability property, the simulator of the commitment scheme gives a computationally indistinguishably distributed simulation of the commitments from honest parties. Furthermore, the simulator acts honestly on behalf of honest parties in commit protocols that allow a corrupted party to commit to a value towards an honest party. Thus, this step is computationally indistinguishably distributed from the corresponding one in the protocol. If the adversary corrupts a party P_i during or after this step, the simulator can perfectly patch the commitments to r_i' (which is uniformly random like r_i in the protocol) using the adaptiveness property and give the computationally indistinguishably distributed information to the adversary.

5. 1. By the same reasoning as in step 2.1, this step is computationally indistinguishably distributed from the one in the real execution (use Proposition 3 and semantic security of the encryption scheme). Furthermore, if the adversary decides to corrupt a party P_i during or after this step, the Patch property of the encryption scheme

ensures that the simulator can perfectly patch the internal state of P_i to r'_i (as in step 2.1, the randomness learned in the patching of the reliably broadcasted value to r'_i in the protocol and in the simulation is distributed identically, namely uniformly). Hence, the simulator can give indistinguishably distributed information about the state of P_i in the reliable broadcast with P_i as sender to the adversary. Moreover, the simulator executes all reliable broadcasts honestly on behalf of honest parties and hence, it can provide the adversary with computationally indistinguishably distributed information about the state of P_i in all reliable broadcasts.

2. The same reasoning as in step 2.2. applies.
3. The same reasoning as in step 2.3. applies.

6. By the same reasoning as in step 5 (use semantic security of the encryption scheme, Proposition 3, the zero-knowledge property of the proof given and reduction), this step is computationally indistinguishably distributed from the corresponding step in the real execution. It remains to argue that if the adversary decides to corrupt party P_i during or after this step, the simulator can give it computationally indistinguishably distributed information about the state of P_i.

- If $i \in I'$, then by the reasoning in step 5, the simulator can provide computationally indistinguishably distributed information about the state of P_i up to (and including) step 5 (since $r_i = r'_i$ for $i \in I'$, we have that r_i is uniformly random like in the real execution of the protocol). Since step 6 is executed honestly and can be efficiently simulated using only information that the adversary knows (reduction), the distribution of the information given in the simulation is computationally indistinguishable to the one in the real execution.
- If $i \notin I'$, we first want to show that, as in the real execution, the r_i that the adversary receives from the simulator is uniformly random conditioned on the r_l's of the adversary. Let $R_l^{\mathcal{S}}$ be the random variables capturing the values of the r_l's in step 6 of the simulation for $l \in \{1, \ldots, n\}$. With this notation, we want to show that the random variable $R_i^{\mathcal{S}}$ conditioned on $R_l^{\mathcal{S}}$ for $P_l \in C$ is uniformly distributed for all $P_i \notin C$.
  For an index set $\hat{I} \subseteq \{0, \ldots, n\}$ of size $t+1$ and for values $\{a_k\}_{k \in \hat{I}}$, we denote the unique polynomial of degree less than or equal to t passing through all points $\{(\alpha_k, a_k)_{k \in \hat{I} \setminus \{0_{pk}\}}\}$ and $(0_{pk}, a_0)$ by $p_{\{a_k\}_{k \in \hat{I}}}$.
  Since the adversary was still able to corrupt P_i, this implies there exists at least one honest party P_j which is still honest such that $j \in I'$ (otherwise the adversary could corrupt at least $t+1$ parties). Let us denote the set of indices of honest parties in I' by J. By the reasoning above, we have $J \neq \varnothing$.
  Let now $a, b, \{v_k\}_{P_k \in S' \setminus C}$ be arbitrary values in the message space. We will show that

$$\Pr[R_i^{\mathcal{S}} = a \mid R_k^{\mathcal{S}} = v_k \text{ for } P_k \in S' \cap C] = \Pr[R_i^{\mathcal{S}} = b \mid R_k^{\mathcal{S}} = v_k \text{ for } P_k \in S' \cap C].$$

  This equality implies that r_i is uniformly random conditioned on *all* r_l's of the adversary because the simulator defines r_i independently of r_l for $P_l \in C \setminus S'$.

We have

$$\begin{aligned}
&\Pr[R_i^{\mathcal{S}} = a \mid R_k^{\mathcal{S}} = v_k \text{ for } P_k \in S' \cap C] \\
&\quad = \sum_{\{a_k\}_{k\in J}} \Pr[R_i^{\mathcal{S}} = a \mid (R_k^{\mathcal{S}} = v_k \text{ for } P_k \in S' \cap C) \\
&\qquad\quad \wedge (R_k^{\mathcal{S}} = a_k \text{ for } k \in J)] \cdot \Pr[R_k^{\mathcal{S}} = a_k \text{ for } k \in J] \\
&\quad = \sum_{\{a_k\}_{k\in J}} 1_{\{a = p_{s,\{a_k\}_{k\in J},\{v_k\}_{k\in I'\setminus J}}(\alpha_i)\}} \cdot \Pr[R_k^{\mathcal{S}} = a_k \text{ for } k \in J] \\
&\quad = \sum_{\substack{\{a_k\}_{k\in J} \\ a = p_{s,\{a_k\}_{k\in J},\{v_k\}_{k\in I'\setminus J}}(\alpha_i)}} \Pr[R_k^{\mathcal{S}} = a_k \text{ for } k \in J] \\
&\quad = \sum_{\substack{\{a_k\}_{k\in J} \\ a = p_{s,\{a_k\}_{k\in J},\{v_k\}_{k\in I'\setminus J}}(\alpha_i)}} \left(\frac{1}{|R_{pk}|}\right)^{|J|}
\end{aligned}$$

where on the second line we used the law of total probability and where 1_A is the indicator random variable for an event A. The second equality follows from the fact that $R_i^{\mathcal{S}}$ is determined by $\{R_k^{\mathcal{S}}\}_{k\in I'}$ and the last equality holds because by definition of the simulator, $R_k^{\mathcal{S}}$ is uniformly distributed in the message space R_{pk} and independent of all $R_l^{\mathcal{S}}$ for all $k, l \in J$. By the same reasoning, we have

$$\Pr[R_i^{\mathcal{S}} = b \mid R_k^{\mathcal{S}} = v_k \text{ for } P_k \in S' \cap C] = \sum_{\substack{\{a_k\}_{k\in J} \\ b = p_{s,\{a_k\}_{k\in J},\{v_k\}_{k\in I'\setminus J}}(\alpha_i)}} \left(\frac{1}{|R_{pk}|}\right)^{|J|}.$$

The sets $A = \{\{a_k\}_{k\in J} \colon a = p_{s,\{a_k\}_{k\in J},\{v_k\}_{k\in I'\setminus J}}(\alpha_i)\}$ and $B = \{\{a_k\}_{k\in J} \colon b = p_{s,\{a_k\}_{k\in J},\{v_k\}_{k\in I'\setminus J}}(\alpha_i)\}$ have the same cardinality, namely $|A| = |B| = N^{|J|-1} \geqslant 1$ (remember that we argued above that $|J| \geqslant 1$). Hence,

$$\sum_{\substack{\{a_k\}_{k\in J} \\ a = p_{s,\{a_k\}_{k\in J},\{v_k\}_{k\in I'\setminus J}}(\alpha_i)}} \left(\frac{1}{|R_{pk}|}\right)^{|J|} = \sum_{\substack{\{a_k\}_{k\in J} \\ b = p_{s,\{a_k\}_{k\in J},\{v_k\}_{k\in I'\setminus J}}(\alpha_i)}} \left(\frac{1}{|R_{pk}|}\right)^{|J|}$$

and thus,

$$\Pr[R_i^{\mathcal{S}} = a \mid R_k^{\mathcal{S}} = v_k \text{ for } P_k \in S' \cap C] = \Pr[R_i^{\mathcal{S}} = b \mid R_k^{\mathcal{S}} = v_k \text{ for } P_k \in S' \cap C].$$

We can conclude that the distribution of r_i that the adversary sees in the simulation is indistinguishable from the distribution in the real execution.
Hence, by the reasoning in step 4, the simulator can provide computationally indistinguishably distributed information about the state of P_i up to (and including) step 4 (the commitments in step 4 are patched to $r_i = p(\alpha_i)$ instead of r_i', but the reasoning to show computational indistinguishability is the same). Furthermore, by the same reasoning as in step 5 for the case where the adversary corrupts P_i and the simulator patches the internal state of P_i to r_i', we have that the simulator can patch the internal state of P_i for the execution of step 5 to $r_i = p(\alpha_i)$. As step 6 is executed honestly with respect to r_i and can be efficiently simulated using only information that the adversary knows (reduction), the simulator can now provide the adversary with information about the internal state of P_i that is computationally indistinguishably distributed from the one in the real execution.

7. If the adversary decides to corrupt a party P_i during or after this step, the simulator can give computationally indistinguishably distributed information about the internal state of P_i to the adversary because it executes the step honestly and the step can be efficiently executed using only information that the adversary knows (reduction).
8. Since in the simulation R is an encryption of 1_{pk} and K is an encryption of 0_{pk}, now $\mathsf{Enc}^R_{pk}(r_i)$ are encryptions of r_i for all $P_i \in \hat{S}$ (ewnp) and $\mathsf{Enc}_{pk}(\hat{s})$ is an encryption of 0_{pk}. Hence, $\mathsf{Enc}_{pk}(s)'$ is an encryption of $\sum_{i\in I} \lambda_i \cdot_{pk} r_i$ and thus, $\mathsf{Enc}_{pk}(s)'$ and $\mathsf{Enc}_{pk}(\hat{s})$ do not encrypt the same value anymore. By construction in step 6 of the simulator $\sum_{i\in I} \lambda_i \cdot_{pk} r_i = s$ and thus, $\mathsf{Enc}_{pk}(s)'$ is an encryption of s. Note that ewnp the output s of the ideal functionality is identically distributed to the message underlying the output of the circuit in the real execution. This is true because the circuit is correct and because the messages underlying the encrypted inputs of corrupted parties to the circuit in the real execution and the inputs given to the ideal functionality in the simulation are identically distributed ewnp (by the extraction property of UC commitment schemes and by computational indistinguishability of the real execution and the simulation up to the step when corrupted parties give input). Hence, we have that the distributions of the decrypted value in the simulation and the real execution are identical ewnp. Furthermore, by the reasoning in the previous steps, we know that the distributions of $\mathsf{Enc}_{pk}(s)'$ are computationally indistinguishable in the real execution and the simulation. Thus, by Remark 2, the communication in this step is computationally indistinguishably distributed in the real execution and the simulation. Hence, we can conclude that the step maintains computational indistinguishability. If the adversary decides to corrupt P_i during or after this step, then the simulator can give computationally indistinguishably distributed information about the internal state of P_i in this step to the adversary thanks to Remark 2 and because the simulator runs the decryption protocol honestly on behalf of P_i.
9. Since the secret input s of the parties is identically distributed in the real execution and in the simulation of this step (ewnp) and since the simulator acts honestly on behalf of honest parties, computational indistinguishability is maintained. If the adversary decides to corrupt a party P_i during or after this step, the simulator can give computationally indistinguishably distributed information about the internal state of P_i in this step because the simulator executes the step honestly on behalf of P_i.

H Proof of Lemma 2

We will prove each property separately. The proof is inspired by [BTH08], [HN06] and [BHN08] and analogous to the proof of Lemma 1 (see Appendix E). In this whole proof, an honest party is a party that remains honest during the whole execution of the protocol. Furthermore, we use uppercase letters to denote ciphertexts and the corresponding lowercase letters to denote the plaintexts underlying these ciphertexts.

– *Termination:* Since all honest parties terminate the $\Pi^{\kappa,Q}_{\mathsf{VACS}}$ protocol and the $\Pi^{\kappa,Q'}_{\mathsf{VACS}}$ protocol ewnp, we can immediately conclude that they terminate the $\mathsf{WeakTriples}$ protocol ewnp and output ℓ triples.
– *Consistency:* This property holds by consistency of the VACS primitive.
– *Correctness:* By the soundness of zero-knowledge proofs, we know that for every element $\{(B^k_j, C^k_j, p^k_{2,j})\}_{k\in\{1,\ldots,\ell\}}$ in S', the plaintext underlying C^k_j is the multiplication of the plaintexts underlying B^k_j and A^k for all $k \in \{1,\ldots,\ell\}$. By definition of B^k and C^k for $k \in \{1,\ldots,\ell\}$, we can directly conclude that the plaintext underlying C^k is the multiplication of the plaintexts underlying B^k and A^k for all $k \in \{1,\ldots,\ell\}$. Hence, the output triples are correct.
– *Secrecy:* We start by showing that the plaintexts underlying A^i are unknown to the adversary for all $i \in \{1,\ldots,\ell\}$. By Theorem 2, we have that ewnp there is at least one honest party

P_h's input $\{(A_h^k, p_{1,h}^k)\}_{k\in\{1,\ldots,\ell\}}$ in S. Let i be an arbitrary index in $\{1,\ldots,\ell\}$. By definition of A^i we have $A^i = A_h^i \bigoplus_{j\colon \{(A_j^k,p_{1,j}^k)\}_{k\in\{1,\ldots,\ell\}}\in S\setminus\{(A_h^k,p_{1,h}^k)\}_{k\in\{1,\ldots,\ell\}}} A_j^i$. The guarantees of the VACS primitive and the definition of Q ensure that for every $\{(A_j^k, p_{1,j}^k)\}_{k\in\{1,\ldots,\ell\}} \in S$, $p_{1,j}^k$ is a valid zero-knowledge proof of plaintext knowledge for A_j^k for all $k \in \{1,\ldots,\ell\}$. Therefore, soundness of zero-knowledge proofs implies that with high probability the adversary knows the plaintext a_j^i underlying A_j^i for all inputs $\{(A_j^k, p_{1,j}^k)\}_{k\in\{1,\ldots,\ell\}} \in S$ with P_j corrupted at the beginning of the VACS protocol. By semantic security, the adversary does not know the plaintext a_h^i underlying A_h^i. Thus, the a_j^i's from corrupted parties are independent of a_h^i and we can conclude that the plaintext underlying A^i is unknown to the adversary. Since i was an arbitrary index in $\{1,\ldots,\ell\}$, this holds for all $i \in \{1,\ldots,\ell\}$.
The reasoning that the plaintexts underlying B^i are unknown to the adversary for all $i \in \{1,\ldots,\ell\}$ is analogous.
To show that the adversary does not have more information about the plaintexts underlying C^i than that they are the multiplication of the plaintexts underlying A^i and B^i for all $i \in \{1,\ldots,\ell\}$, we observe the following. Let i be an arbitrary index in $\{1,\ldots,\ell\}$ and let $\{(B_j^k, C_j^k, p_{2,j}^k)\}_{k\in\{1,\ldots,\ell\}}$ be any element of S'. If P_j is corrupted before it is told to erase b_j^i, then the adversary knows b_j^i with high probability (by the guarantees of the VACS protocol, the definition of Q' and soundness of zero-knowledge proofs). However, by semantic security and because a^i is unknown to the adversary (see above), the adversary still does not know anything more about c_j^i than that it is the multiplication of b_j^i and the unknown plaintext a^i underlying A^i (note that since $\{(B_j^k, C_j^k, p_{2,j}^k)\}_{k\in\{1,\ldots,\ell\}}$ is in S', the guarantees of the VACS protocol, the definition of Q' and soundness of zero-knowledge proofs ensure that with high probability C_j^i is indeed the multiplication of the plaintexts underlying B_j^i and A^i). If P_j remains honest during the whole execution of the protocol or is corrupted after it already erased b_j^i, then by semantic security, the adversary does not have more information about c_j^i than that it is the multiplication of the plaintexts underlying B_j^i and A^i which are both unknown to the adversary. Hence, since $C^i = \bigoplus_{j\colon \{(B_j^k,C_j^k,p_{2,j}^k)\}_{k\in\{1,\ldots,\ell\}}\in S'} C_j^i$, $B^i = \bigoplus_{j\colon \{(B_j^k,C_j^k,p_{2,j}^k)\}_{k\in\{1,\ldots,\ell\}}\in S'} B_j^i$ and all B_j^i's are only used to compute B^i, we can conclude that the adversary does not know anything about the plaintext underlying C^i but that it is the multiplication of the plaintexts underlying A^i and B^i. Finally, since i was an arbitrary index in $\{1,\ldots,\ell\}$, we can conclude that Secrecy holds.

– *Computational Uniform Randomness and Independence:* These two properties can be proven using similar arguments as for the Secrecy property and a similar reasoning as for the multiplication protocol in [DN03].
– *Privacy:* It is easy to see that this property holds because the simulator $\mathcal{S}_{\mathsf{WeakTriples}}$ can perfectly imitate the honest parties (no party has any secret input to this protocol). Hence, also if the adversary decides to corrupt any party during the execution of the protocol, the simulator can give perfectly indistinguishably distributed information to the adversary.
– *Communication complexity:* The parties only communicate in the executions of the $\Pi_{\mathsf{VACS}}^{\kappa,Q}$ and the $\Pi_{\mathsf{VACS}}^{\kappa,Q'}$ protocols in steps 2 and 4 of the WeakTriples protocol. Hence, by Theorem 2 and since the size of the inputs of the parties in the $\Pi_{\mathsf{VACS}}^{\kappa,Q}$ and the $\Pi_{\mathsf{VACS}}^{\kappa,Q'}$ protocols is $\mathcal{O}(\ell\kappa)$, we have that the communication complexity is $\mathcal{O}(\ell\kappa^3 n + \kappa^5 n)$.