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

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

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 $t < n/4$.

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, BH08, 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. Note, however, that in contrast to the protocol in [Cho20] which is information-theoretic, our protocol has cryptographic security. 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. [BKLL20]), 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 the security parameter by κ and use the abbreviation ewnp for "except with negligible probability". Our protocols are proven in the model by Canetti [Can00a]. A summary can be found in the full version [CHL21].

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.

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].

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

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 satisfies $Q(x, z) =$ 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 (UC) commitment schemes. A detailed exposition is given in the full version [CHL21].

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.

In our MPC protocol, we need the following additional property of our commitment scheme. A detailed discussion about the selective decommitment problem can be found in [DNRS03].

Selective decommitment security: Consider the following security game with an integer $t \in \{1, \dots, n\}$ (representing the corruption threshold) and a message distribution M over R_{pk}^n as parameters.

– The challenger samples a uniform random bit $b \in_R \{0, 1\}$.
– The adversary sends a set of indices $I \subset \{1, \dots, n\}$ of size $t' \in \{0, \dots, t\}$ to the challenger.
– The challenger samples n messages according to the distribution M, enumerates them in the natural way and gives the messages with indices in I to the adversary. Next, for each message with index not in I, the challenger commits to it and gives the computed $n - t'$ commitments to the adversary.
– The adversary can adaptively choose up to $t - t'$ of the given commitments and the challenger gives the underlying messages and the randomness used to obtain the commitments in question to the adversary. As soon as the adversary does not want to choose any more commitments, it sends "EndCorruption" to the challenger.
– Upon receipt of the "EndCorruption"-message or if the adversary has already chosen $t - t'$ commitments, the challenger does the following. Let $I' \subseteq \{1, \dots, n\}$ be the set of indices that are not in I and such that the adversary did not choose the commitments with indices in I'.
  - If $b = 0$, the challenger gives the messages underlying the commitments with indices in I' to the adversary.
  - Let $M_{I'}$ be the distribution M conditioned on the components with indices not in I' being equal to the messages already given to the adversary. If $b = 1$, the challenger samples $|I'|$ messages according to the distribution $M_{I'}$ and gives them to the adversary.
– The adversary outputs a guess b' for the value of the bit b.

The idea in the above game is that every party commits to one value and the adversary can corrupt up to t parties. In doing that, the adversary should not learn anything about the messages underlying the commitments of honest parties. This game can be generalized in a natural way to the case where each party P_i commits to a fixed number ℓ_i of values (and this number can be different for each party). For the sake of simplicity, we do not give the formal description of the more general game. We define the advantage of the adversary in the generalized game by

$$\mathrm{Advtg}_{\{\ell_1,\dots,\ell_n\}}^{M,t} := |\Pr[b' = b] - \frac{1}{2}|.$$

We require from our commitment scheme that for all n-tuples $\{\ell_1, \dots, \ell_n\}$ of integers, all message distributions M and all $t < n/3$, there does not exist any adversary that has non-negligible advantage $\mathrm{Advtg}_{\{\ell_1,\dots,\ell_n\}}^{M,t}$.

For all the commitments in our protocols, we will use a UC adaptively secure (equivocable and extractable) commitment scheme that satisfies the "Selective decommitment security" property above and has constant communication complexity.

2.4 Threshold Homomorphic Encryption

We briefly discuss threshold homomorphic encryption schemes. For a detailed exposition, see the full version [CHL21].

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 $(pk, sk_1, \ldots, sk_n)$ 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 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 $\mathsf{Enc}_{pk}(m_2, r_2)$ of m_1 and m_2 and that outputs an encryption $\mathsf{Enc}_{pk}(m_1, r_1) \oplus_{pk} \mathsf{Enc}_{pk}(m_2, r_2) \coloneqq \mathsf{Enc}_{pk}(m_1 +_{pk} m_2, r_1 \boxplus_{pk} r_2)$ of $m_1 +_{pk} m_2$, 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 $i \in \{1, \ldots, n\}$, 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.
– 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 the full version [CHL21].

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 with respect to 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 because the randomness space is a group) $\mathsf{Enc}_{pk}^{M}(x)$ is a uniformly random encryption of $x \cdot_{pk} m$, where m is the value encrypted by M.

In our MPC 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 there exist efficient patchable zero-knowledge proofs of knowledge for Q_{pk}^{M} with constant communication complexity for all public keys pk and all encryptions M under pk.
– *Lagrange arguments:* There exists an n-tuple $\{\alpha_1, \ldots, \alpha_n\} \in (R_{pk} \setminus \{0_{pk}\})^n$ of distinct elements such that for all $(i, j) \in \{1, \ldots, n\}^2$ 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 $E = (x \odot_{pk} K) \oplus_{pk} \mathsf{Enc}_{pk}(0_{pk}, r_E) = \mathsf{Enc}_{pk}^{K}(x, r_E)$.
– *Selective decryption security:* This property is similar to the "Selective decommitment security" property of our commitment scheme. For a detailed discussion, we again refer the reader to [DNRS03].
Consider the following security game with a message distribution M over R_{pk}^{n} and a randomness distribution Rd over the n product of the randomness space as parameters.
  • The challenger samples a uniform random bit $b \in_R \{0, 1\}$.
  • The adversary sends a set of indices $I \subset \{1, \ldots, n\}$ of size $t' \in \{0, \ldots, t\}$ to the challenger.
  • The challenger samples n messages according to the distribution M and n randomness elements according to the distribution Rd, enumerates them in the natural way and gives the messages and randomness elements with indices in I to the adversary. Next, for each message with index not in I, the challenger encrypts it using the corresponding randomness element (i.e. the randomness element with the same index) and gives the computed $n - t'$ ciphertexts to the adversary.

- The adversary can adaptively choose up to $t-t'$ of the given ciphertexts and the challenger gives the underlying messages and the randomness used to obtain the ciphertexts in question to the adversary. As soon as the adversary does not want to choose any more ciphertexts, it sends "EndCorruption" to the challenger.
- Upon receipt of the "EndCorruption"-message or if the adversary has already chosen $t-t'$ ciphertexts, the challenger does the following. Let $I' \subseteq \{1,\ldots,n\}$ be the set of indices that are not in I and such that the adversary did not choose the ciphertexts with indices in I'.
    * If $b=0$, the challenger gives the messages underlying the ciphertexts with indices in I' to the adversary.
    * Let $M_{I'}$ be the distribution M conditioned on the components with indices not in I' being equal to the messages already given to the adversary. If $b=1$, the challenger samples $|I'|$ messages according to the distribution $M_{I'}$ and gives them to the adversary.
- The adversary outputs a guess b' for the value of the bit b.

The idea in the above game is that every party encrypts one value and the adversary can corrupt up to t parties. In doing that, the adversary should not learn anything about the messages underlying the encryptions of honest parties. This game can be generalized in a natural way to the case where each party P_i encrypts a fixed number ℓ_i of values (and this number can be different for each party). For the sake of simplicity, we do not give the formal description of the more general game. We define the advantage of the adversary in the generalized game by

$$\mathrm{Advant}^{M,Rd}_{\{\ell_1,\ldots,\ell_n\}} := |\Pr[b'=b]-\frac{1}{2}|.$$

We require from our encryption scheme that for all n-tuples of integers $\{\ell_1,\ldots,\ell_n\}$, all message distributions M and all randomness distributions Rd, there does not exist any adversary that has non-negligible advantage $\mathrm{Advant}^{M,Rd}_{\{\ell_1,\ldots,\ell_n\}}$, even if it has access to a simulator for zero-knowledge proofs and the Pat algorithm.

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 the full version [CHL21], 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 the full version [CHL21], 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 the full version [CHL21], 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 the full version [CHL21].
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 the full version [CHL21], 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 $\mathsf{PrivDec}$ protocol is a straightforward private decryption protocol which takes as input the public key pk, the private keys $sk_1, \ldots, sk_n$, 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 $\mathsf{PrivDec}$ 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 A.1.

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 $\mathsf{PrivDec}$ 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 $\mathsf{PrivDec}$ 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 A.1, 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 A.2.

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 [BH08], 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 1). Furthermore, it communicates $\mathcal{O}(n^2\kappa)$ bits.

3.4 Triple Generation

This subsection is devoted to the introduction of the $\mathsf{Triples}$ protocol which takes as input an integer ℓ and outputs ℓ encrypted multiplication triples. The protocol is based on the multiplication protocol in [DN03], the KFD-TRIPLES protocol in [HN06] and on [CP15]. 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^i_j in the message space R_{pk} and r^i_j in the randomness space for all $i \in \{1,\ldots,\ell\}$. Then, P_j computes $A^i_j = \mathsf{Enc}_{pk}(a^i_j, r^i_j)$ and uses the Multi-Valued-Acast protocol to broadcast A^i_j for all $i \in \{1,\ldots,\ell\}$. Finally, P_j proves to P_k in zero-knowledge that it knows the plaintext underlying A^i_j using the "proof of plaintext knowledge" property of the encryption scheme with instance A^i_j and witness (a^i_j, r^i_j) 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^i_k with $i \in \{1,\ldots,\ell\}$ terminated for P_j and P_j accepted all proofs of plaintext knowledge for A^i_k 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 $A^i = \bigoplus_{P_k \in S} A^i_k$ for all $i \in \{1,\ldots,\ell\}$.

4: Every party P_j independently chooses uniformly random elements b^i_j in the message space R_{pk} and r'^i_j in the randomness space for all $i \in \{1,\ldots,\ell\}$. Then, P_j computes $B^i_j = \mathsf{Enc}_{pk}(b^i_j, r'^i_j)$ and $(C^i_j, r''^i_j) = \mathsf{Blind}(b^i_j \odot_{pk} A^i)$ and uses the Multi-Valued-Acast protocol to broadcast B^i_j and C^i_j for all $i \in \{1,\ldots,\ell\}$. Finally, P_j proves to P_k in zero-knowledge that C^i_j was computed correctly using the "proof of correct multiplication" property of the encryption scheme with instance (B^i_j, A^i, C^i_j) and witness (b^i_j, r'^i_j, r''^i_j) 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^i_k, C^i_k) with $i \in \{1,\ldots,\ell\}$ terminated for P_j and P_j accepted all proofs of correct multiplication for (B^i_k, A^i, C^i_k) 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 $B^i = \bigoplus_{P_k \in S'} B^i_k$ and $C^i = \bigoplus_{P_k \in S'} C^i_k$ 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 product 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 computationally 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 the full version [CHL21].

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, BH08]. 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, CDN00].
Let $f \colon \mathbb{N} \times \{0,1\}^* \times (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ 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, no input at all or — as long as the adversary has not specified the core set S in step 3 — change their inputs (for example after corruption of any party). If the adversary corrupts a party P_j at any point in time during or 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 $\mathsf{Enc}^M_{pk}(v)$. 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^M_{pk}, $\mathsf{Enc}^M_{pk}(v)$ 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 of the inputs, their reliable broadcast, the zero-knowledge proofs and the run of the ACS protocol are summed up in the BrACS protocol in Appendix B.

- *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 A.2. 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 cannot 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 $\mathsf{Enc}^R_{pk}(r_i)$. 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 $\mathsf{Enc}^R_{pk}(r_i)$ 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 $\mathsf{Enc}^R_{pk}(r_i)$ 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 $\mathsf{Enc}^R_{pk}(r_i)$ 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 $\mathsf{Enc}_{pk}^{K}(r_i)$) due to K being an encryption of 0_{pk}, it can *not* do the same for the second BrACS (with $\mathsf{Enc}_{pk}^{R}(r_i)$) 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 B.

4.3 Main Theorem

Our protocol achieves the following.

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

The simulator and an informal proof of the theorem are given in the full version [CHL21].

Remark 5. It is straightforward to generalize the above protocol to the case where the function f takes c_I inputs and provides c_O outputs. If a party P_i has multiple inputs, it commits to each one of them, reliably broadcasts a random encryption of each one of them and proves compatible commitment for each one of them (in the BrACS). Furthermore, with multiple outputs, the parties execute steps 4–7 of the protocol for each encrypted output of the circuit and then reconstruct the randomized outputs towards the entitled parties. This results in an increase of the communication complexity by a quadratic factor per input and by a cubic factor per output, which leads to the following theorem.

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

Remark 6. This paper does not focus on round complexity. For information about round-efficient MPC, we refer the reader to [CGHZ16]. Our protocol has a round complexity that depends on the circuit depth.

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 and a slightly lower corruption threshold.
Taking a closer look at the communication complexity of the protocol in Appendix B 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 [BKLL20].

As before (see Subsection 2.1), we consider multiparty computation among a set of n parties $P_1, \dots, P_n$, 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 cannot corrupt the party in the midst of the sending). Furthermore, messages sent by any honest party P_i are guaranteed to arrive eventually, 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 [BKLL20].
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 (VACS) 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 3. *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 $\frac{q}{2}$ 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 $\mathit{\Pi}_{\mathsf{VACS}}^{q,Q}$ and the proof of the first part of the theorem can be found in [BKLL20]. The second part of the theorem can be proven using Lemma 24 of [BKLL20].

5.3 Triple Generation

To obtain an efficient protocol for the triple generation in the atomic send model, we start with our $\mathsf{Triples}$ protocol from Subsection 3.4 and make it more efficient using the VACS primitive, NIZK proofs and erasures. The following protocol is inspired by the protocols in [BKLL20]. It takes as input an integer ℓ and outputs ℓ encrypted multiplication triples.

Protocol $\mathsf{WeakTriples}$

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 $\mathit{\Pi}_{\mathsf{VACS}}^{\kappa,Q}$ 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 VACS protocol $\mathit{\Pi}_{\mathsf{VACS}}^{\kappa,Q'}$ 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' is defined such that $Q'(\{(B_j^k, C_j^k, p_{2,j}^k)\}_{k\in\{1,\dots,\ell\}}) = 1$ 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 $k \in \{1,\dots,\ell\}$. The parties obtain a multiset S' of size at least κ and define $B^i = \bigoplus_{j:\, \{(B_j^k, C_j^k, p_{2,j}^k)\}_{k\in\{1,\dots,\ell\}} \in S'} B_j^i$ and $C^i = \bigoplus_{j:\, \{(B_j^k, C_j^k, p_{2,j}^k)\}_{k\in\{1,\dots,\ell\}} \in S'} C_j^i$ for all $i \in \{1,\dots,\ell\}$.

5: Every party outputs (A^i, B^i, C^i) for all $i \in \{1,\dots,\ell\}$.

Remark 7. 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 product 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 computationally 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 has expected communication complexity $\mathcal{O}(\ell\kappa^3 n + \kappa^5 n)$.*

The proof is given in the full version [CHL21].

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 B 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. Furthermore, using the reliable broadcast protocol presented in [BKLL20] in our BrACS protocol, we can reduce the communication complexity per input and obtain the following theorem.

Theorem 4. *Let $0 < \epsilon < 1/3$ and $t \leqslant (1 - 2\epsilon) \cdot n/3$. There exists an MPC protocol that t-securely realizes the ideal functionality in Subsection 4.1 in the* KG*-hybrid atomic send model and that has expected communication complexity $\mathcal{O}(c_M n\kappa^3 + Dn^2\kappa + c_I n\kappa^2 + c_O n^3\kappa + n^3\kappa + n\kappa^5)$, where D is the multiplicative depth of the circuit, c_M is the number of multiplication gates, c_I is the number of input gates and c_O is the number of (public and private) output gates in 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`.

[BH08] 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`.

[BKLL20] 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`.

[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`.

[CGHZ16] Sandro Coretti, Juan Garay, Martin Hirt, and Vassilis Zikas. Constant-round asynchronous multi-party computation based on one-way functions. In J.H. Cheon and T. Takagi, editors, *Advances in Cryptology*

— *ASIACRYPT 2016*, volume 10032 of *LNCS*. Springer-Verlag, 2016. doi:10.1007/978-3-662-53890-6_33.

[CHL21] Annick Chopard, Martin Hirt, and Chen-Da Liu-Zhang. On communication-efficient asynchronous MPC with adaptive security. Cryptology ePrint Archive, Report 2021/1174, 2021. https://ia.cr/2021/1174.

[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.

[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.

[DNRS03] Cynthia Dwork, Moni Naor, Omer Reingold, and Larry Stockmeyer. Magic functions: In memoriam: Bernard m. dwork 1923–1998. *J. ACM*, 50(6):852–921, November 2003. doi:10.1145/950620.950623.

[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 multi-party 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`.

[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`.

[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 Details of the Subprotocols

A.1 Decryption protocols

Private Decryption The private decryption protocol $\mathsf{PrivDec}$ 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 [BH08, CHP12, CP15].

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 $P_k$, $P$ uses the Comb algorithm to
   compute $m = \mathsf{Comb}(pk, c, \{(c_k, p_k^c)\}_{k \in \{1,\ldots,n\}})$, where $P$ sets all the values that
   is has not received to $\perp$. Then $P$ outputs $m$.
```

Lemma 3. *Every party that remains uncorrupted until the end of the execution terminates the* $\mathsf{PrivDec}$ *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 [DN07, CHP12, BH08, CP15].

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 $\mathcal{P}_i'$ as soon as it receives $u_k'$ 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 decoder) 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 remains uncorrupted until the end of 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 $\mathsf{PolyFind}$ algorithm would have succeeded (see above). Hence, if in an iteration the $\mathsf{PolyFind}$ 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 $\mathsf{PrivDec}$ 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 instance of the PubDec protocol, each party executes the $\mathsf{PolyFind}$ algorithm up to $t+1$ times. By using local player elimination, we can reduce the number of runs of the $\mathsf{PolyFind}$ 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 $\mathsf{PolyFind}$ 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 $\mathsf{PolyFind}$ 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 $\mathsf{PolyFind}$ 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 $c^1_1, \ldots, c^1_T$ and $c^2_1, \ldots, c^2_T$ two computationally indistinguishably distributed sets of T ciphertexts with computationally indistinguishably distributed sets of underlying plaintexts, an instance of the PubDec protocol with $(pk, c^1_1, \ldots, c^1_T)$ as public input (and $sk_1, \ldots, sk_n$ as private inputs) is computationally indistinguishably distributed to an instance of the PubDec protocol with $(pk, c^2_1, \ldots, c^2_T)$ 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.

A.2 Multiplication

This subsection presents the multiplication protocol which is based on [DN07] and the Multiplication Gate in the Computation Phase protocol of [BH08]. 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 $\{(X_i, Y_i)\}_{i\in\{1,\ldots,T\}}$ (encrypting the values $\{(x_i, y_i)\}_{i\in\{1,\ldots,T\}}$) 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 require that the multiplication triples underlying the encrypted triples $\{(A_i, B_i, C_i)\}_{i\in\{1,\ldots,T\}}$ are unknown to the adversary and computationally uniformly and independently distributed over the space of all multiplication triples (the latter is equivalent to the plaintexts underlying the first and second components of the triples being computationally uniformly and independently distributed and the third component being the product of the first two). 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 1. *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 given gates. 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, we know by reduction that the ciphertexts $(\{X_i^1 \ominus_{pk} A_i\}_{i \in \{1,\ldots,T\}}, \{Y_i^1 \ominus_{pk} B_i\}_{i \in \{1,\ldots,T\}})$ and $(\{X_i^2 \ominus_{pk} A_i\}_{i \in \{1,\ldots,T\}}, \{Y_i^2 \ominus_{pk} B_i\}_{i \in \{1,\ldots,T\}})$ are computationally indistinguishably distributed. Furthermore, we know that the plaintexts underlying $\{A_i\}_{i \in \{1,\ldots,T\}}$ and the plaintexts underlying $\{B_i\}_{i \in \{1,\ldots,T\}}$ are unknown to the adversary and computationally uniformly and independently distributed. Therefore, the plaintexts underlying $\{X_i^1 \ominus_{pk} A_i\}_{i \in \{1,\ldots,T\}}$, $\{Y_i^1 \ominus_{pk} B_i\}_{i \in \{1,\ldots,T\}})$, $\{X_i^2 \ominus_{pk} A_i\}_{i \in \{1,\ldots,T\}}$ and $\{Y_i^2 \ominus_{pk} B_i\}_{i \in \{1,\ldots,T\}})$ are all unknown to the adversary and computationally uniformly and independently 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 2. *The multiplication protocol communicates $\mathcal{O}(n^2\kappa)$ bits.*

B 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 party P_i gets the common inputs $pk, K, R, \{K_\nu\}_\nu, \{\alpha_i\}_{i\in\{1,\dots,n\}}$ and the secret inputs $sk_i, \{K^i_\chi\}_\chi$, where $(pk, sk_1, \dots, sk_n)$ 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, $\{K^i_\chi\}_\chi$ are the private keys of P_i used for the zero-knowledge proofs and the commitment scheme and $\{\alpha_i\}_{i\in\{1,\dots,n\}}$ 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 $H^1_{pk}, \dots, H^l_{pk}$.
4: Let c_M be the number of multiplication gates in the circuit. The parties execute the $\mathsf{Triples}$ 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 $j \in \{1,\dots,n\}$ under the corresponding commitment key. For all $(i,j) \in \{1,\dots,n\}$, 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: Each party P_i chooses a uniformly random value r_{x_i} from the randomness space. The parties run the BrACS protocol from Appendix B with public input (pk, K) and secret input $(x_i, r_{x_i}, \{c_{ij}\}_{j\in\{1,\dots,n\}}, \{C_{i\to j}\}_{j\in\{1,\dots,n\}}, \{C_{j\to i}\}_{j\in\{1,\dots,n\}})$ for every party P_i and 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 $J \subseteq \{1,\dots,l\}$ 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 is a default value.
   b) If H^j_{pk} is a constant input gate for a 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 A.2 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 $j \in \{1,\dots,n\}$ under the corresponding commitment key. For all $(i,j) \in \{1,\dots,n\}$, 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 the BrACS protocol (see Appendix B) with public input (pk, K) and secret input $(r_i, r^K_{r_i}, \{b_{ij}\}_{j\in\{1,\dots,n\}}, \{B_{i\to j}\}_{j\in\{1,\dots,n\}}, \{B_{j\to i}\}_{j\in\{1,\dots,n\}})$ 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. Then, the parties run the BrACS protocol with public input (pk, R) and secret input $(r_i, r^R_{r_i}, \{b_{ij}\}_{j\in\{1,\dots,n\}}, \{B_{i\to j}\}_{j\in\{1,\dots,n\}}, \{B_{j\to i}\}_{j\in\{1,\dots,n\}})$ 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 $\mathsf{Enc}_{pk}(s)' = \mathsf{Enc}_{pk}(s) \bigoplus_{i\in I}^{pk} (\lambda_i \odot_{pk} \mathsf{Enc}^R_{pk}(r_i))$.

8: The parties use their secret keys to run $\mathsf{PrivDec}(P_i,\ \mathsf{Enc}_{pk}(s)')$ for all $i \in \{1,\dots,n\}$ and all parties obtain s.

9: The parties run the reliable consensus protocol RC 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.
2: Every 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 3. *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.