Asynchronous Byzantine Agreement with Subquadratic Communication

Erica Blum[1][⋆], Jonathan Katz[2][⋆⋆], Chen-Da Liu-Zhang[3][⋆⋆⋆], and Julian Loss[1][†]

[1] University of Maryland
[2] George Mason University
[3] ETH Zurich

Abstract. Understanding the communication complexity of Byzantine agreement (BA) is a fundamental problem in distributed computing. In particular, as protocols are run with a large number of parties (as, e.g., in the context of blockchain protocols), it is important to understand the dependence of the communication on the number of parties n. Although adaptively secure BA protocols with $o(n^2)$ communication are known in the synchronous and partially synchronous settings, no such protocols are known in the fully asynchronous case.

We show here an asynchronous BA protocol with subquadratic communication tolerating an adaptive adversary who can corrupt $f < (1-\epsilon)n/3$ of the parties (for any $\epsilon > 0$). One variant of our protocol assumes initial setup done by a trusted dealer, after which an unbounded number of BA executions can be run; alternately, we can achieve subquadratic *amortized* communication with no prior setup. We also show that some form of setup is needed for (non-amortized) subquadratic BA tolerating $\Theta(n)$ corrupted parties.

As a contribution of independent interest, we show a secure-computation protocol in the same threat model that has $o(n^2)$ communication when computing no-input functionalities with short output (e.g., coin tossing).

1 Introduction

Byzantine agreement (BA) [26] is a fundamental problem in distributed computing. In this context, n parties wish to agree on a common output even when some f of those parties might be adaptively corrupted. Although BA is a well-studied problem, it has recently received increased attention due to its application to blockchain (aka state machine replication) protocols. Such applications typically involve a large number of parties, and it is therefore critical to understand how the communication complexity of BA scales with n. While protocols with adaptive security and $o(n^2)$ communication complexity have been obtained in both the synchronous [24] and partially synchronous [1] settings, there are currently no such solutions for the *asynchronous* model.[1] This leads us to ask:

Is it possible to design an asynchronous BA protocol with subquadratic communication complexity that tolerates $\Theta(n)$ adaptive corruptions?

We give both positive and negative answers to this question.

[⋆] **Email:** `erblum@cs.umd.edu`
[⋆⋆] **Email:** `jkatz2@gmail.com`
[⋆⋆⋆] **Email:** `lichen@inf.ethz.ch`
[†] **Email:** `lossjulian@gmail.com`

[1] Tolerating $f < n/3$ *static* corruptions is easy; see Section 1.1.

Positive results. We show asynchronous BA protocols with (expected) subquadratic communication complexity that can tolerate adaptive corruption of any $f < (1-\epsilon)n/3$ of the parties, for arbitrary $\epsilon > 0$. (This corruption threshold is almost optimal, as it is known [6] that asynchronous BA is impossible for $f \geq n/3$, even assuming prior setup and static corruptions.) Our solutions rely on two building blocks, each of independent interest:

1. We show a BA protocol Π_{BA} with subquadratic communication, assuming prior setup by a trusted dealer for each execution. Importantly, the total size of the setup is independent of n. (In particular, only a small number of parties are given anything by the dealer.)
2. We also construct an secure-computation protocol Π_{MPC}, assuming subquadratic BA available as a subroutine. Π_{MPC} has subquadratic communication complexity when computing no-input functionalities whose output length is independent of n, regardless of the size of the circuit being computed. Moreover, the number of BA executions needed is independent of n as well as the output length.

We can combine these results to give an affirmative answer to the original question. Specifically, using a trusted dealer, we can achieve an *unbounded* number of BA executions with $o(n^2)$ communication per execution. The idea is as follows. Let L be the number of BA executions required by Π_{MPC} for computing a no-input functionality. The dealer provides the parties with the setup needed for $L+1$ executions of Π_{BA}; the total size of this setup is linear in L but independent of n. Then, each time the parties wish to carry out Byzantine agreement, they will use one instance of their setup to run Π_{BA}, and use the remaining L instances to refresh their initial setup by running Π_{MPC} to simulate the dealer. The total communication complexity is subquadratic in n.

Alternately, we can avoid a trusted dealer by having the parties run an arbitrary adaptively secure MPC protocol to generate the initial setup. This protocol may not have subquadratic communication complexity; however, once it is finished the parties can revert to the solution above which has subquadratic communication per BA execution. Overall, this gives BA with *amortized* subquadratic communication.

Impossibility result. We justify our reliance on a trusted dealer by showing that some form of setup is necessary for (non-amortized) subquadratic BA tolerating $\Theta(n)$ corrupted parties. Moreover, this holds even when secret channels and erasures are available. Our bound extends ideas of Abraham et al. [1], who showed a similar result but in a different adversarial model (see further discussion in the next section).

1.1 Related Work

The problem of BA was introduced by Lamport, Shostak and Pease [26]. Without some form of setup, BA is impossible (even in a synchronous network) when $f \geq n/3$. Fischer, Lynch, and Patterson [27] ruled out deterministic protocols for asynchronous BA even when $f = 1$. Starting with the work of Rabin [34], randomized protocols for asynchronous BA have been studied in both the setup-free setting [12,30] as well as the setting with PKI and a trusted dealer [9].

Dolev and Reischuk [17] show that any BA protocol achieving subquadratic communication complexity (even in the synchronous setting) must be randomized. BA with subquadratic communication complexity was first studied in the synchronous model by King et al., who gave setup-free protocols with polylogarithmic communication complexity for the case of $f < (1 - \epsilon)n/3$ static corruptions [25] and $O(n^{1.5})$ communication complexity for the same number of adaptive corruptions [24]. Subsequently, several works [1, 21, 28, 29, 31] gave improved protocols with subquadratic communication complexity (in the synchronous model with an adaptive adversary) using the "player replaceability paradigm," which requires setup in the form of verifiable random functions.

Abraham et al. [1] show a BA protocol with adaptive security and subquadratic communication complexity in the *partially synchronous* model. They also give a version of the Dolev-Reischuk bound that rules out subquadratic BA (even with setup, and even in the synchronous communication model) against a strong adversary who is allowed to remove messages sent by honest parties from the network after those parties have been adaptively corrupted. Our lower bound adapts their ideas to the standard asynchronous model where honest parties' messages can be arbitrarily delayed, but cannot deleted once they are sent. (We refer to the work of Garay et al. [19] for further discussion of these two models.) In concurrent work [35], Rambaud proves an impossibility result similar to our own; we refer to Section 8 for further discussion.

Cohen et al. [15] show an adaptively secure asynchronous BA protocol with $o(n^2)$ communication, using trusted setup. However, they consider a non-standard asynchronous model in which the adversary is not allowed to reorder messages from honest parties (and is restricted in further ways as well). We work in the standard asynchronous model.

We remark for completeness that asynchronous BA with subquadratic communication complexity for a *static* adversary corrupting $f < n/3$ of the parties is trivial using a committee-based approach, assuming a trusted dealer. Roughly, the dealer simply chooses a random committee of $O(\kappa)$ parties who run BA on behalf of everyone. Achieving subquadratic BA *without* a dealer in the static-corruption model is an interesting open question.

Asynchronous secure multi-party computation (MPC) was proposed by Ben-Or, Canetti and Goldreich [3]. Since then, improved protocols have been proposed with both unconditional [32, 33, 36] and computational [13, 14, 22, 23] security. To the best of our knowledge, all above asynchronous MPC protocols incur a total communication complexity of at least $O(n^3\kappa)$, assuming each party has input of length κ, and the output has length κ. Their protocols achieve optimal output quality. Our MPC protocol gives a trade-off between the communication complexity and the output quality. In particular, our protocol achieves subquadratic communication complexity when the required output quality is sublinear in the number of parties (as in the case of no-input, randomized functions).

1.2 Overview of the Paper

In Section 2 we discuss our model and recall some standard definitions. We show how to achieve reliable consensus and reliable broadcast with subquadratic communication in Section 3. In Section 4 we present our first building block: an asynchronous BA protocol with subquadratic communication complexity, assuming prior setup by a trusted dealer for each execution. Section 6 introduces our second building block: a communication-efficient asynchronous protocol fo secure multi-party computation (MPC). We describe how these components can be combined to give our main results in Section 7. We conclude with our lower bound in Section 8.

2 Preliminaries and Definitions

We denote the security parameter by κ, and assume $\kappa \leq n$. We let PPT stand for probabilistic polynomial time. We use standard digital signatures, where a signature on a message m using secret key sk is computed as $\sigma \leftarrow \mathsf{Sign}_{\mathsf{sk}}(m)$; a signature is verified relative to public key pk by calling $\mathsf{Vrfy}_{\mathsf{pk}}(m, \sigma)$. We treat signatures as idealized objects with perfect unforgeability and correctness. We record some standard concentration inequalities in Appendix B.

2.1 Model

We consider a setting where n parties $P_1, \ldots, P_n$ run a distributed protocol over a network in which all parties are connected via pairwise authenticated channels. We work in the *asynchronous* model, meaning the adversary can arbitrarily schedule the delivery of all messages, so long as all messages are eventually delivered. We consider an *adaptive* adversary that can corrupt some bounded number f of the parties at any point during the execution of some protocol, and cause them to deviate arbitrarily from the protocol specification. However, we assume the standard "atomic send" model, which means that (1) if at some point in the protocol an honest party is instructed to send several messages (possibly to different parties) simultaneously, then the adversary can corrupt that party either before or after it sends all those messages, but not in the midst of sending those messages; and (2) once an honest party sends a message, that message is guaranteed to be delivered eventually even if that party is later corrupted. In addition, we assume secure erasure.

In many cases we assume a incorruptible dealer who can initialize the parties with setup information in advance of any protocol execution. Such setup may include both public information given to all parties, as well as private information given to specific parties; when we refer to the size of a setup, we include the total private information given to all parties but count the public information only once. Weaker forms of setup are also possible; for example, a public key infrastructure (PKI) means that all parties hold the same vector of public keys $(pk_1, \ldots, pk_n)$ and each honest party P_i holds the honestly generated secret key sk_i corresponding to pk_i. When the PKI includes keys for a digital signature scheme, we sometimes let $\langle m \rangle_i$ denote a signature on a message m computed using P_i's secret key sk_i.

2.2 Definitions

We define various types of distributed protocols. In all cases, we implicitly assume the protocols take the security parameter κ as input and properties may fail to hold with probability negligible in κ.

Reliable broadcast allows a sender to consistently distribute a message to a set of parties. In contrast to full-fledged broadcast, reliable broadcast does not require termination.

Definition 1 (Reliable broadcast) *Let Π be a protocol executed by parties $P_1, \ldots, P_n$, where a designated sender P^* initially holds input v^*, and parties terminate upon generating output. Π is an f-*secure reliable broadcast protocol *if the following hold if at most f parties are corrupted:*

- **Validity:** *if P^* is honest at the start of the protocol, then every honest party outputs v^*.*

– **Consistency:** *either no honest party terminates, or all honest parties output the same value.*

Reliable consensus is a similar primitive for the case where there is no designated sender.

Definition 2 (Reliable consensus) *Let Π be a protocol executed by parties $P_1, \ldots, P_n$, where each party P_i holds an input v_i and parties terminate upon generating output. Π is an f-*secure reliable consensus protocol *if the following hold if at most f parties are corrupted:*

– **Validity:** *if every honest party has the same input value v, then every honest party outputs v.*
– **Consistency:** *either no honest party terminates, or all honest parties output the same value.*

Byzantine agreement strengthens reliable consensus by requiring termination.

Definition 3 (Byzantine agreement) *Let Π be a protocol executed by parties $P_1, \ldots, P_n$, where each party P_i holds an input v_i and parties terminate upon generating output. Π is an f-*secure Byzantine agreement protocol *if the following hold if at most f parties are corrupted:*

– **Validity:** *if every honest party has the same input value v, then every honest party outputs v.*
– **Consistency:** *all honest parties output the same value.*

3 Reliable Consensus/Broadcast with Subquadratic Communication

We start by describing a reliable consensus protocol Π_{RC} with subquadratic communication. The protocol can be seen as a variant of Bracha's reliable broadcast protocol [6, 7] for the case where every party has input.

The protocol assumes prior setup initialized by a trusted dealer. The dealer selects two secret committees C_1, C_2 by independently placing each party in C_1 (resp., C_2) with probability κ/n. For each party P_i in C_1 (resp., C_2), the dealer generates a public/private key pair $(\mathsf{pk}_{1,i}, \mathsf{sk}_{1,i})$ (resp., $(\mathsf{pk}_{2,i}, \mathsf{sk}_{2,i})$) for a signature scheme and gives the associated private key to P_i; the public keys (but not the identities of the parties in the committee) are given to all parties. In order to achieve adaptive security, it is crucial that each member of a committee send only a single message. This ensures that once an attacker learns that some party is in a committee, adaptively corrupting that party is useless.

Π_{RC} is described in Figure 1. The protocol begins by having each party in C_1 send its signed input to all parties. The parties in C_2 then send a `ready` message on a value v either (1) upon receiving v from $\kappa - t$ parties in C_1 or (2)upon receiving `ready` messages on v from $t+1$ parties in C_2. All parties terminate upon receiving `ready` messages with the same value from $\kappa - t$ parties in C_2.

Each committee has expected size $O(\kappa)$, and each member of a committee sends a single message to all parties; thus, $O(\kappa n)$ messages are sent (in expectation) during the protocol.

Theorem 4 *Let $\epsilon > 0$ and $f < (1 - 2\epsilon)n/3$. Then Π_{RC} is an f-secure reliable consensus protocol with expected communication complexity $O(\mathcal{I}\kappa n)$, where $\mathcal{I}$ is the size of each party's input. The expected size of the setup is $O(\kappa^2)$.*

Protocol Π_{RC}

We describe the protocol from the point of view of a party P_i with input v_i, assuming as setup random committees C_1, C_2 with public keys as described above:

1. If $P_i \in C_1$: Compute $\sigma = \mathsf{Sign}_{\mathsf{sk}_{1,i}}(v_i)$, erase $\mathsf{sk}_{1,i}$, and send $(\mathtt{echo}, (v_i, \sigma))$ to all parties.
2. If $P_i \in C_2$: Upon receiving $(\mathtt{echo}, (v, \sigma_j))$ on the same value v from $\kappa - t$ distinct parties and with $\mathsf{Vrfy}_{\mathsf{pk}_{1,j}}(v, \sigma_j) = 1$, compute $\sigma = \mathsf{Sign}_{\mathsf{sk}_{2,i}}(v)$, erase $\mathsf{sk}_{2,i}$ and send $(\mathtt{ready}, (v, \sigma))$ to all parties.
3. If $P_i \in C_2$: Upon receiving $(\mathtt{ready}, (v, \sigma_j))$ on the same value v from $t + 1$ distinct parties and with $\mathsf{Vrfy}_{\mathsf{pk}_{2,j}}(v, \sigma_j) = 1$, if no $\mathtt{ready}$ message was sent, compute $\sigma = \mathsf{Sign}_{\mathsf{sk}_{2,i}}(v)$, erase $\mathsf{sk}_{2,i}$ and send $(\mathtt{ready}, (v, \sigma))$ to all parties.
4. Upon receiving $(\mathtt{ready}, (v, \sigma_j))$ on the same value v from $\kappa - t$ distinct parties and with $\mathsf{Vrfy}_{\mathsf{pk}_{2,j}}(v, \sigma_j) = 1$, output v and terminate.

Fig. 1. A reliable consensus protocol. Here, $t = (1 - \epsilon)\kappa/3$.

We prove Theorem 4 using a series of lemmas. The following lemmas make use of the fact that whenever an honest party $P_i \in C_r$ sends a message containing (v, σ), $\sigma = \mathsf{Sign}_{\mathsf{sk}_{r,i}}$, and erases its secret key $\mathsf{sk}_{r,i}$, it is guaranteed that v is the only value that can be eventually received and accepted by honest parties. In particular, even if an adversary corrupts P_i after it sent its messages, it cannot change the value it sent. Throughout the lemmas, we set the protocol parameter equal to $t = \kappa(1 - \epsilon)/3$.

Lemma 5 *Π_{RC} is f-valid.*

Proof. Assume all honest parties start with the same input v. Then every party in C_1 that is honest when executing Step 1 sends $(\mathtt{echo}, (v, \sigma_i))$ to all other parties. Since, by Lemma 32, there are at least $\kappa - t$ such parties in C_1 with overwhelming probability, all parties in C_2 receive $(\mathtt{echo}, (v, \sigma_j))$ from at least $\kappa - t$ parties with valid signatures from C_1. Moreover, because each party receives at most $t < \kappa - t$ messages $(\mathtt{echo}, (v', \sigma_j))$, $v' \neq v$, by Lemma 32, each party $P_i \in C_2$ that is honest when executing Step 2 or 3, sends $(\mathtt{ready}, (v, \sigma_i))$. Finally, all honest parties receive $(\mathtt{ready}, (v, \sigma_j))$ from at least $\kappa - t$ parties with valid signatures from C_2 with overwhelming probability and output v and terminate. □

Lemma 6 *If honest parties P_i, P_j send $(\mathtt{ready}, (v_i, \sigma_i))$ and $(\mathtt{ready}, (v_j, \sigma_j))$, respectively, then $v_i = v_j$.*

Proof. Assume this is not the case, and let P and P' be the first parties that are honest when sending $(\mathtt{ready}, (v, \sigma))$ and $(\mathtt{ready}, (v', \sigma'))$ respectively, with $v \neq v'$. This implies that P (resp. P') received at least $\kappa - t$ messages $(\mathtt{echo}, (v, \sigma_j))$ (resp. $(\mathtt{echo}, (v', \sigma_j))$) from distinct parties with valid signatures from C_1.

We now argue about the number of parties in C_1 that were honest at the time of sending its $\mathtt{echo}$ message. From the $\kappa - t$ messages, at least $\kappa - 2t$ came from honest parties with overwhelming probability by Lemma 32. Since honest parties only sent a single echo message, with overwhelming probability there are $2\kappa - 4t = \frac{\kappa(2+4\epsilon)}{3}$ distinct honest parties, which is in contradiction with Lemma 32. □

Lemma 7 *Π_{RC} is f-consistent.*

Proof. Assume that an honest party P_i outputs v. Then, P_i received $(\texttt{ready}, (v, \sigma_j))$ from at least $\kappa - t$ parties in C_2. Since with overwhelming probability there are at most t corrupted parties in C_2 by Lemma 32, at least $\kappa - 2t \geq t + 1$ parties in C_2 were honest and sent $(\texttt{ready}, (v, \sigma_j))$. Moreover, by Lemma 6, ready messages from (back then) honest parties are unique, so no honest party sent a message $(\texttt{ready}, (v', \sigma_j))$, with $v' \neq v$. As a consequence, all honest parties in C_2 eventually send $(\texttt{ready}, (v, \sigma_j))$ and all honest parties eventually receive $(\texttt{ready}, (v, \sigma_j))$ from $\kappa - t$ parties in C_2 and output v as well.

Reliable broadcast. It is easy to obtain reliable broadcast $\mathit{\Pi}_{\mathsf{RBC}}$ from reliable consensus: the sender simply signs its message and sends it to all parties, who then run reliable consensus on what they received. The communication complexity is $O(\kappa n)$, the same as for reliable consensus. We formally describe a reliable broadcast protocol $\mathit{\Pi}_{\mathsf{RBC}}$ in Figure 2.

Protocol $\mathit{\Pi}_{\mathsf{RBC}}$

The sender is P^* with secret key sk^*.

1. P^* does: compute $\sigma = \mathsf{Sign}_{\mathsf{sk}^*}(v)$, erase sk^*, and send (v, σ) to all parties.
2. Upon receiving a pair (v, σ) such that $\mathsf{Vrfy}_{\mathsf{pk}^*}(v, \sigma) = 1$, input v to the reliable consensus protocol $\mathit{\Pi}_{\mathsf{RC}}$.
3. Upon receiving output v from $\mathit{\Pi}_{\mathsf{RC}}$, output v and terminate.

Fig. 2. A reliable broadcast protocol.

Theorem 8 *Let $\epsilon > 0$ and $f < (1 - 2\epsilon)n/3$. Then $\mathit{\Pi}_{\mathsf{RBC}}$ is a f-secure reliable broadcast protocol with expected communication complexity $O(\mathcal{I}\kappa n)$, where $\mathcal{I}$ is the size of the sender's input. The expected size of the setup is $O(\kappa^2)$.*

We prove Theorem 8 by separately considering validity and consistency.

Lemma 9 *$\mathit{\Pi}_{\mathsf{RBC}}$ is f-valid.*

Proof. Assume that the sender P_s is honest at the start of the execution and sends the messages at Step 1. Then, all honest parties eventually receive a pair (v, σ) with a valid signature from the sender. Note that no other valid pair is received even if P_s is adaptively corrupted, due to the fact that P_s erases its secret key before sending.

As a result, every honest party starts a reliable consensus protocol with input v and by validity of reliable consensus, every honest party outputs v and terminates.

Lemma 10 *$\mathit{\Pi}_{\mathsf{RBC}}$ is f-consistent.*

Proof. This follows trivially by consistency of the underlying reliable consensus protocol.

4 BA with Subquadratic Communication

In this section we describe a BA protocol with subquadratic communication complexity assuming initial setup provided by a trusted dealer. Our protocol, based on ideas by Mostéfaoui et al. [30], consists of a sequence of iterations, where each iteration invokes a graded consensus subprotocol and a coin-flip subprotocol. In each iteration there is a constant probability that honest parties reach agreement; once agreement is reached, it cannot be lost in later iterations. The coin-flip protocol allows parties to adopt the value of a common coin if agreement has not yet been reached (or, at least, if parties are unaware that it has been reached). Reliable consensus is used so parties know when to terminate the protocol. We set $t = (1-\epsilon)\kappa/3$ throughout this section.

4.1 Graded Consensus

Graded consensus [18] can be viewed as a weaker form of Byzantine agreement where parties output a grade along with a value, and agreement is required to hold only if some honest party outputs a grade of 1. Our definition does not require termination.

Definition 11 (Graded consensus) *Let Π be a protocol executed by parties $P_1, \ldots, P_n$, where each party P_i holds an input v_i and is supposed to output a value w_i along with a grade $g_i \in \{0,1\}$. Π is an f-*secure graded consensus protocol* if the following hold if at most f parties are corrupted:*

- **Graded validity:** *if every honest party has the same input value v, then every honest party outputs $(v, 1)$.*
- **Graded consistency:** *if some honest party outputs $(w, 1)$, then every honest party P_i outputs (w, g_i).*

Our graded consensus protocol is an adaptation of Canetti-Rabin graded consensus [12], but with subquadratic communication complexity. The protocol assumes a setup which defines three secretly chosen committees C_1, C_2, C_3, each of expected size κ. Each party in a committee will act as a sender in Π_{RBC}; independent setup is used for each of these. The graded consensus protocol consists of three phases. In phase i, each party in committee C_i uses Π_{RBC} to send a phase-specific message to all parties. Since each set C_i has expected size κ, and the expected communication complexity of Π_{RBC} is $O(\kappa^2 n)$ (since the messages are of size $O(\kappa)$), the graded consensus protocol has expected communication complexity $O(\kappa^3 n)$. We formally describe the graded consensus protocol and prove the following theorem in Appendix C.

Theorem 12 *Let $\epsilon > 0$ and $f < (1-2\epsilon)n/3$. Then Π_{GC} is an f-secure graded consensus protocol with expected communication complexity $O(\kappa^3 n)$. The expected size of the setup is $O(\kappa^3)$.*

5 A Coin-Flip Protocol

We describe a protocol to generate a sequence of coins. We denote the sub-protocol to generate the i-th instance of the coin by $\mathsf{CoinFlip}(i)$.

The coin-flip subroutine ensures that, in each instance i, (1) all honest parties output the same coin and (2) until the first honest party calls $\mathsf{CoinFlip}(i)$, the output of $\mathsf{CoinFlip}(i)$ is uniform.

The coin-flip protocol uses as setup a committee of κ parties, where each party holds a signed coin share (see Appendix A.1 for a definition of secret-sharing). Parties then simply send the coin-share to all parties, and reconstruct the uniquely determined coin value.

Coin-Flip Setup

We describe the protocol from the view of the dealer D. Generate a pair $(\mathsf{sk}, \mathsf{pk})$ of secret and public key of a signature scheme. Send pk to all parties.
For each $i \in [\kappa]$ do:

- For each $P \in [n]$, choose P with probability κ/n. Let E_i denote the set of chosen parties.
- Let $\mathsf{TSS} = (\mathsf{Share}, \mathsf{Rec})$ be a $(\lceil \kappa/3 \rceil, |E_i|)$-threshold secret sharing scheme.
- Sample $b' \leftarrow \{0,1\}$.
- Generate shares $(s_1, \ldots, s_{|E_i|}) \leftarrow \mathsf{Share}(b')$.
- Send $(s_j, \mathsf{Sign}_{\mathsf{sk}}(s_j, i))$ to the j-th member of E_i.
- Send $|E_i|$ to all parties.

Fig. 3. A Setup Protocol for T Coins

Protocol $\mathsf{CoinFlip}(i)$

We describe the protocol from the point of view of a party P_j, assuming setup as described in Figure 3:
Let s_j denote the share held by P_j (where possibly $s_j = \bot$).

1. Set $S = \emptyset$.
2. If $P_j \in E_i$: send s_j to all parties.
3. Upon receiving a share s, set $S = S \cup \{s\}$.
4. Upon receiving at least $\lceil \kappa/3 \rceil$ shares:
   Return $\mathsf{Coin}_i := \mathsf{Rec}(\{s\}_{s \in S})$.

Fig. 4. A coin flip protocol.

Lemma 13 *Coin properties. Let $f < n \cdot \frac{1-2\epsilon}{3}$, $\epsilon > 0$. With overwhelming probability,* $\mathsf{CoinFlip}(i)$ *satisfies the following for every input i:*

1. *all honest parties obtain the same value* Coin_i*,*
2. *until the first honest party invokes* $\mathsf{CoinFlip}(i)$*, the distribution of* Coin_i *appears uniformly random to the adversary,*
3. *the expected setup size is $O(\kappa^2)$, and*
4. *the expected communication complexity is $O(\kappa^2 n)$.*

Proof. It immediately follows from Lemma 32 that with probability at least $1 - p_3$, for any i corresponding to a round in which $\mathsf{CoinFlip}(i)$ is evaluated, there are at most $\kappa(1-\epsilon)/3 < \lceil \kappa/3 \rceil$ dishonest parties that are selected for the i^{th} coin committee E_i. Furthermore, with probability at least $1 - p_1$, there are more than $\kappa(2+\epsilon)/3$ that are selected for E_i in any such round who are honest when they begin running $\mathsf{CoinFlip}(i)$, and accordingly send their share of the coin. Thus, by the secrecy property of TSS, with probability at least $1-p_1-p_3$, every invocation of $\mathsf{CoinFlip}(i)$ yields a coin which looks random to the adversary upon the first honest party sending its share. Moreover, by the reconstruction property of TSS, all honest parties obtain the same value for Coin_i.

5.1 Byzantine Agreement

We present our BA protocol in Figure 5. The total setup for Π_{BA} corresponds to κ setups of graded consensus, a setup for κ coin-flips, and one setup for reliable consensus.

Protocol Π_{BA}

We describe the protocol from the point of view of a party with input $v \in \{0,1\}$.

Set $b := v$, $\mathsf{ready} := \mathsf{false}$, and $k := 1$. Then do:

```
1. Run $\Pi_{\mathsf{GC}}$ on input $b$, and let $(b, g)$ denote the output.
2. $\mathsf{Coin}_k \leftarrow \mathsf{CoinFlip}(k)$.
3. If $g = 0$ then set $b := \mathsf{Coin}_k$.
4. Run $\Pi_{\mathsf{GC}}$ on input $b$, and let $(b, g)$ denote the output.
5. If $g = 1$ and $\mathsf{ready} := \mathsf{false}$: set $\mathsf{ready} := \mathsf{true}$ and input $b$ to $\Pi_{\mathsf{RC}}$.
6. Set $k := k + 1$ and repeat from (1.).
```

Termination: When Π_{RC} terminates with output b', output b' and terminate.

Fig. 5. A subquadratic Byzantine agreement protocol.

The following theorem is proven in Appendix D.

Theorem 14 *Let $\epsilon > 0$ and $f < (1-2\epsilon)n/3$. Then Π_{BA} is an f-secure BA protocol with expected communication complexity $O(\kappa^4 n)$. The expected size of the setup is $O(\kappa^4)$.*

6 MPC with Subquadratic Communication Complexity

Definition of Asynchronous MPC. We recap the definition of Canetti [11] for asynchronous MPC. Let g denote a function, possibly randomized, where if the inputs of the parties are $x_1, \ldots, x_n$ and $(y_1, \ldots, y_n) \leftarrow g(x_1, \ldots, x_n)$, then P_i is supposed to learn y_i. In the real-world execution of a protocol Π computing g, parties initially hold inputs $\mathbf{x} = (x_1, \ldots, x_n)$ and 1^κ, and an adversary $\mathcal{A}$ takes as input 1^κ and auxiliary input z. The parties execute Π, and $\mathcal{A}$ may adaptively corrupt parties during execution of the protocol. At the end of the execution, each honest party outputs its final result, and $\mathcal{A}$ outputs its view. We let $\textsc{real}_{\Pi,\mathcal{A}}(\kappa, \mathbf{x}, z)$ denote the distribution over the resulting vector of outputs as well as the set of corrupted parties.

The security of Π is defined relative to an ideal-world evaluation of g by a trusted party. Here, the parties begin holding inputs $\mathbf{x} = (x_1, \ldots, x_n)$, and an adversary $\mathcal{S}$ is given 1^κ and auxiliary input z. The ideal execution then proceeds as follows:

– **Initial corruption**. $\mathcal{S}$ may adaptively corrupt parties and learn their inputs.

- **Computation with ℓ-output quality**. $\mathcal{S}$ chooses a set $\mathsf{CoreSet} \subseteq \{P_1, \ldots, P_n\}$ of size at least ℓ. For $P_i \notin \mathsf{CoreSet}$, let $x'_i = \perp$; if $P_i \in \mathsf{CoreSet}$ is honest, let $x'_i = x_i$; and if $P_i \in \mathsf{CoreSet}$ is corrupted, then $\mathcal{S}$ can set x'_i arbitrarily.
  The trusted party then computes $(y_1, \ldots, y_n) \leftarrow g(x'_1, \ldots, x'_n)$ and sends y_i to each party P_i.
- **Additional corruption**. $\mathcal{S}$ can corrupt additional parties and learn their outputs.[2]
- **Output stage**. Each honest party P_i outputs y_i, and $\mathcal{S}$ outputs an arbitrary function of its view.

We let $\text{IDEAL}^{\ell}_{g,\mathcal{S}}(\kappa, \mathbf{x}, z)$ be the distribution over the vector of outputs and the set of corrupted parties following an ideal-world execution as above.

Definition 15 *Π* f-secure*ly* computes g with ℓ-output quality *if for any* PPT *adversary $\mathcal{A}$ corrupting up to f parties, there is a* PPT *adversary $\mathcal{S}$ such that:*

$$\{\text{IDEAL}^{\ell}_{g,\mathcal{S}}\}_{\kappa\in\mathbb{N};\mathbf{x},z\in\{0,1\}^*} \approx_c \{\text{REAL}_{\Pi,\mathcal{A}}\}_{\kappa\in\mathbb{N};\mathbf{x},z\in\{0,1\}^*}.$$

Our protocol follows the homomorphic encryption approach [14,22,23] but using threshold *fully* homomorphic encryption (FHE); see Appendix A.2 for appropriate definitions. Our protocol achieves a trade-off between the communication complexity and the output quality. It has subquadratic communication complexity when the output quality, as well as the input and output lengths of the functionality being computed, are sublinear in the number of parties.

Our MPC protocol invokes a *validated asynchronous common subset* (VACS) protocol as a subroutine; we give a definition of VACS, and a VACS protocol with subquadratic communication complexity, in the next section.

6.1 VACS with Subquadratic Communication Complexity

A protocol for the asynchronous common subset (ACS) problem [4,10] allows n parties to agree on a subset of their initial inputs. We consider a *validated* version of ACS, where it is additionally ensured that all values in the output multiset satisfy a given predicate Q. This notion is inspired by the standard notion of validated asynchronous Byzantine agreement [8].

Definition 16 (VACS) *Let Q be a predicate, and let Π be a protocol executed by parties $P_1, \ldots, P_n$, where each party P_i initially holds an input v_i, outputs a multiset of size at most n, and terminates upon generating output. Π is an* **f-secure Q-validated ACS protocol with ℓ-output quality** *if the following hold if at most f parties are corrupted:*

- **Correctness:** *If an honest party outputs S, each value $v \in S$ satisfies $Q(v) = 1$.*
- **Consistency:** *every honest party outputs the same multiset.*
- **ℓ-Output quality:** *if every honest party's input v_i satisfies $Q(v_i) = 1$, then all honest parties output a multiset of size at least ℓ.*

[2] This assumes erasure. Otherwise, $\mathcal{S}$ is given the inputs and outputs of corrupted parties.

We describe a common-subset protocol with subquadratic communication complexity. Let Q be a predicate. The protocol allows n parties, $P_1, \ldots, P_n$, to agree on a common subset of inputs values satisfying the predicate Q. The protocol follows the structure of [4].

The protocol $\Pi_{\mathsf{VACS}}^{\ell,Q}$ uses as setup a secret randomly chosen committee of parties C, where each party is independently added to the committee with probability $p := s/n$, where $s := \frac{3}{2+\epsilon}\ell$ and ℓ is the output-quality. The idea is that parties run a number of instances $\mathsf{RBC}_1, \ldots, \mathsf{RBC}_{|C|}$ of the reliable broadcast protocol introduced in Section 3, where each party $P_i \in C$ acts as the sender in one instance of reliable broadcast, and then run $|C|$ instances of asynchronous byzantine agreement $\mathsf{BA}_1, \ldots, \mathsf{BA}_{|C|}$ to agree on the set of reliable broadcast instances which terminated and satisfy the predicate Q. A formal description of the protocol $\Pi_{\mathsf{VACS}}^{\ell,Q}$ and a proof of the following theorem can be found in Appendix E.

Theorem 17 *Let $\epsilon > 0$, $f < (1-2\epsilon)n/3$, and $\ell \leq (2+\epsilon)n/3$. For Q an arbitrary predicate, $\Pi_{\mathsf{VACS}}^{\ell,Q}$ is an f-secure Q-validated ACS protocol with ℓ-output quality. The expected communication complexity is $O(\ell \cdot (\mathcal{I}\kappa n + \kappa^4 n))$, where $\mathcal{I}$ is the size of each party's input. The expected size of the setup is $O(\ell\kappa^4)$.*

6.2 An MPC Protocol

Let $t = \kappa(1-\epsilon)/3$. The protocol uses the following setup:

- A randomly chosen committee C, where each party is chosen with probability κ/n.
- A threshold fully-homomorphic encryption scheme $\mathsf{TFHE} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ [20] (defined in Appendix A.2) with threshold t. The public key is ek and the secret key $dk = (dk_1, \ldots, dk_{|C|})$, where each party in C holds a secret key share, and all parties know ek and $|C|$. Each party in C also has a setup as sender for an instance of the protocol Π_{RBC}, whereas all other parties have a recipient-setup.
- A common reference string CRS used for universally composable non-interactive zero-knowledge proofs [16].

Fix a (possibly randomized) functionality g the parties wish to compute. We assume without loss of generality that g uses exactly κ random bits (one can always use a PRG to ensure this). Very roughly, the protocol asks each party P_i to encrypt its input $c_i = \mathsf{Enc}_{ek}(x_i)$ and execute the VACS protocol. The VACS allows parties to agree on a common subset of input providers. After that, parties homomorphically evaluate the function over the encrypted inputs to obtain an encrypted output. Finally, parties in C jointly decrypt the output.

We deal with adaptive security by using secure-erasures: Parties will erase the relevant information prior to sending the messages in each of the steps, avoiding in this way the so-called *commitment problem* (and also avoiding the need for an equivocal FHE scheme [**?**]), where the simulator needs to explain previously sent messages in a consistent manner.

In the protocol, parties prove (in zero-knowledge) the correctness of values sent during the protocol execution. For this purpose we use universally composable non-interactive zero-knowledge (NIZK) proofs [16] based on a common reference string (CRS) provided by the dealer. We are interested in NIZK proofs for two relations, parameterized by a threshold FHE scheme with public encryption key ek:

Protocol $\Pi^{\ell}_{\mathsf{MPC}}$

Let $t = \kappa(1-\epsilon)/3$. We describe the protocol from the point of view of a party P_i with input x_i, assuming the setup described above that defines a committee C. Let g be the circuit to be computed, and let R be the number of random bits used by g.

1. Compute $c_i \leftarrow \mathsf{Enc}_{ek}(x_i)$ along with a non-interactive zero-knowledge proof of plaintext knowledge π_i^{popk}. Erase the local randomness used to generate the ciphertext and the proof. Then, execute the VACS protocol $\Pi^{\ell, Q_{\mathsf{popk}}}_{\mathsf{VACS}}$ using input $(i, \mathsf{Sign}_{\mathsf{sk}_i}(i), c_i, \pi_i^{\mathsf{popk}})$ for the predicate Q_{popk} that checks that the signature is correct and π_i^{popk} is a correct proof for c_i. Let S be the output of the VACS containing ciphertexts with valid proofs from at least ℓ parties.
2. Choose uniform $r \in \{0,1\}^R$ and compute $u_i \leftarrow \mathsf{Enc}_{ek}(r)$ along with a non-interactive zero-knowledge proof of plaintext knowledge τ_k^{popk} for u_i. Erase the plaintext and the local randomness used to generate the ciphertext and the proof. Then, run $\Pi^{t+1, Q_{\mathsf{popk}}}_{\mathsf{VACS}}$ using input $(u_i, \tau_i^{\mathsf{popk}})$ for the predicate that checks that τ_i^{popk} is a correct proof for u_i. Let T denote the output of the VACS. Let v be denote the ciphertext computed by adding homomorphically all ciphertexts in T.
3. Let $\mathsf{Circ}(S)$ be the circuit g where for each party P_j such that there is no correct tuple $(j, \sigma_j, \cdot, \cdot) \in S$ (S defined at Step 1), a default value $\perp$ is hard-wired into the circuit, and (the plaintext corresponding to) ciphertext v is used as the random wires in the circuit. Let c_j denote the input ciphertext obtained of party P_j such that $(j, \sigma_j, c_j, \pi_j^{\mathsf{popk}}) \in S$ in Step 1. Each party uses the homomorphism to locally compute $c := \mathsf{Eval}_{ek}(\mathsf{Circ}(S), c_{j_1}, \ldots, c_{j_{|S|}})$.
4. If $P_i \in C$: compute $d_i = \mathsf{Dec}_{dk_i}(c)$ and a non-interactive zero-knowledge proof of correct decryption π_i^{pocd}. Erase the local randomness to generate the proof and the decryption-key-share dk_i. Then, execute a reliable broadcast protocol Π_{RBC} with input $(d_i, \pi_i^{\mathsf{pocd}})$ as the sender.
5. Upon receiving $t+1$ correct messages $(d_i, \pi^{\mathsf{pocd}})$, compute output $y_i = \mathsf{Rec}(\{d_j\})$.
6. Execute a reliable consensus protocol Π_{RC} with input y_i. On output y from Π_{RC}, output y and terminate.

Fig. 6. An MPC protocol with ℓ-output quality, for $\ell \le \frac{2+\epsilon}{3} n$.

1. *Proof of plaintext knowledge:* The statement consists of ek, and a ciphertext c. The witness consists of a plaintext m and randomness r such that $c = \mathsf{Enc}_{ek}(m; r)$.
2. *Proof of correct decryption:* The statement consists of ek, a ciphertext c, and a decryption share d. The witness consists of a decryption key share dk_i, such that $d = \mathsf{Dec}_{dk_i}(c)$.

The following theorem is proven in Appendix F.

Theorem 18 *Let $f < n \cdot \frac{1-2\epsilon}{3}$, $\epsilon > 0$, and $\ell \le \frac{2+\epsilon}{3} n$. Assuming the NIZK proof system is universally composable, and* TFHE *is a threshold FHE scheme, $\Pi^{\ell}_{\mathsf{MPC}}$ f-securely computes g with ℓ-output quality. The expected communication complexity is $O(\ell(\mathcal{I}\kappa n + \kappa^4 n) + \kappa^5 n + \mathcal{O}\kappa^2 n)$, where $\mathcal{I}$ is the size of each party's input and $\mathcal{O}$ is the size of the output. The expected size of the setup is $O(\ell\kappa^4 + \kappa^5)$. The expected number of invocations of BA is $O(\ell + \kappa)$.*

7 Putting it All Together

The BA protocol Π_{BA} from Section 4 requires prior setup by a trusted dealer. Unfortunately, a given instance of the setup can be used for only a *single* BA execution. Using multiple, independent instances of the setup it is, of course, possible to support any *bounded* number of BA executions. But a new idea is needed to support an *unbounded* number of executions.

In this section we discuss how to use the MPC protocol from Section 6 to achieve this goal. The key idea is to use that protocol to *refresh the setup* each time a BA execution is done. We first describe how to modify our MPC protocol to make it suitable for our setting, and then discuss how to put everything together to obtain the desired result.

7.1 Securely Simulating the Dealer

As just noted, we would like to use the MPC protocol from Section 6 to simulate a dealer. Note that in this case we are evaluating a no-input (randomized) functionality, and so do not need any output quality; let $\Pi_{\mathsf{MPC}} = \Pi^0_{\mathsf{MPC}}$. Using Π_{MPC} to simulate a dealer, however, is not straightforward. As described, Π_{MPC} evaluates a functionality where all parties receive the *same* output, but to simulate a dealer we need to compute a functionality where parties receive *different* outputs. The standard approach for adapting MPC protocols to provide parties with different outputs cannot be used in our context: Specifically, using symmetric-key encryption to encrypt the output of each party P_i using a key that P_i provides as part of its input does not work, since Π_{MPC} has no output quality (and even $\Pi^{\ell}_{\mathsf{MPC}}$ only guarantees ℓ-output quality for $\ell < n$) and so not all parties' inputs will be included. Assuming a PKI, we can fix this by using public-key encryption instead (in the same way); this works since the public keys of all parties can be incorporated into the functionality being computed—since they are common knowledge—rather than being provided as inputs.

Even when using public-key encryption as just described, additional issues remain. Π_{MPC} has expected subquadratic communication only when the output length of the functionality being computed is sublinear in the number of parties. Even if the dealer algorithm generates output whose length is independent of n, naively encrypting output for every party (encrypting a "null" value of the appropriate length for parties whose output is empty) would result in output of total length linear in n. Encrypting the output only for parties with non-empty output does not work either since, in general, this might reveal which parties get output—defeating the purpose of the setup altogether!

We can address this difficulty by using *anonymous public-key encryption* [2] (cf. Appendix A.3). Roughly, an anonymous public-key encryption (APKE) scheme has the property that a ciphertext leaks no information about the public key pk used for encryption, except to the party holding the corresponding secret key sk (who is able to decrypt the ciphertext using that key). Using APKE to encrypt the output (using the corresponding public key) only for parties who obtain non-empty output, and then randomly permuting the resulting ciphertexts, allows us to compute a functionality with sublinear output length while hiding which parties receive output. This incurs—at worst—an additional multiplicative factor of κ in the output length.

Summarizing, we can simulate an arbitrary dealer algorithm in the following way. View the output of the dealer algorithm as $\mathsf{pub}, \{(i, s_i)\}$, where pub represents the public output that all parties should learn, and each s_i is a private output that only P_i should learn. Assume the existence of a PKI, and let pk_i denote a public key for an APKE scheme that is held by P_i. Then use Π_{MPC} to compute $\mathsf{pub}, \{\mathsf{Enc}_{\mathsf{pk}_i}(s_i)\}$, where the ciphertexts are randomly permuted. As long as the length of the dealers output is independent of n, the output of this functionality is also independent of n.

7.2 Unbounded Subquadratic BA

We now show how to achieve an unbounded number of subquadratic BA executions. We describe two solutions: one involving a trusted dealer who initializes the parties with a one-time setup, and the other a dealer-free solution that achieves expected subquadratic communication in an amortized sense.

A trusted dealer can initially distribute a PKI (this is done only once and for all) and in addition initialize the parties with the setup for one instance of Π_{BA} and one instance of Π_{MPC}. Importantly, the setup for Π_{MPC} can be used to compute any no-input functionality; in particular, the size of the setup is fixed, independent of the size of the circuit for the functionality being computed, or its output length. For a BA execution, the parties run Π_{BA} and then use Π_{MPC} to refresh their setup by simulating the dealer algorithm. (We stress that they refresh the setup for both Π_{BA} and Π_{MPC}.) The expected communication complexity per BA execution is the sum of the communication complexities of Π_{BA} and Π_{MPC}. The former is subquadratic; the latter is subquadratic if we follow the approach described in the previous section. By proceeding in this way, the parties can run an unbounded number of BA executions while only involving a trusted dealer once.

Alternately, we can avoid a trusted dealer altogether by having the parties simulate the dealer using an arbitrary (setup-free) MPC protocol. The communication complexity of the initial MPC protocol may be high, but all subsequent BA executions will have subquadratic (expected) communication complexity as above. In this way we achieve an unbounded number of BA executions with amortized (expected) subquadratic communication complexity.

8 Lower Bound

We show that some form of setup is necessary for adaptively ecure asynchronous BA with (non-amortized) subquadratic communication complexity. Our bound holds even if we allow secure erasure, and even if we allow secret channels between all the parties. (However, we assume an attacker can tell when a message is sent from one party to another.)

An analogous bound was shown by Abraham et al. [1, Theorem 4]; their bound holds even with prior setup and in the synchronous model of communication. However, their result relies strongly on an adversary who can delete messages sent by honest parties after those parties have been adaptively corrupted. In contrast, our bound applies to the standard communication model where honest parties' messages cannot be deleted once they are sent. In concurrent work [35], Rambaud also shows a similar bound. He considers protocols in the *partially synchronous* model, and rules out subquadratic communication complexity even if the parties have a PKI. His bound, however, is restricted to protocols that treat signatures in an idealized manner, and thus it does not apply, e.g., to protocols using unique signatures for coin flipping. His bound also does not account for secret channels or erasures.

A BA protocol is (f, δ)-secure if the properties of Definition 3 simultaneously hold with probability at least δ when f parties are corrupted.

Theorem 19 *Let $\frac{2}{3} < \delta < 1$ and $f \geq 2$. Let Π be a setup-free BA protocol that is (f, δ)-secure in an asynchronous network. Then the expected number of messages that honest parties send in Π is at least $(\frac{3\delta-2}{8\delta})^2 \cdot (f-1)^2$.*

We first provide an outline of the proof Let Π be a setup-free protocol for asynchronous BA with subquadratic communication complexity. We show an efficient attacker $\mathcal{A}$ who succeeds in violating the security of Π. The attacker exploits the fact that with high probability, a uniform (honest) party P will communicate with only $o(n)$ other parties during an execution of Π. The adversary $\mathcal{A}$ can use this to "isolate" P from the remaining honest parties in the network and cause an inconsistency. In more detail, consider an execution in which P holds input 1, and the remaining honest parties S' all hold input 0. $\mathcal{A}$ tricks P into thinking that it is running in an alternate (simulated) execution of Π in which all parties are honest and hold input 1, while fooling the parties in S' into believing they are running an execution in which all honest parties hold 0 and at most f (corrupted) parties abort. By validity, P will output 1 and the honest parties in S' will output 0, but this contradicts consistency.

To "isolate" P as described, $\mathcal{A}$ runs two simulated executions of Π alongside the real execution of the protocol. (Here, it is crucial that Π is setup-free, so $\mathcal{A}$ can run the simulated executions on behalf of all parties.) $\mathcal{A}$ delays messages sent by honest parties to P in the real execution indefinitely; this is easy to do in the asynchronous setting. When a party $Q \in S'$ sends a message to P in the simulated execution, $\mathcal{A}$ corrupts Q in the real execution and then sends that message on Q's behalf. Analogously, when P sends a message to some honest party $Q \in S'$ in the real execution, $\mathcal{A}$ "intercepts" that message and forwards it to the corresponding party in the simulation. (A subtlety here is that messages sent between two honest parties cannot be observed via eavesdropping, because we allow secret channels, and can not necessarily be observed by adaptively corrupting the recipient Q after it receives the message, since we allow erasure. Instead, $\mathcal{A}$ must corrupt Q *before* it receives the message sent by P.) It only remains to argue that, in carrying out this strategy, $\mathcal{A}$ does not exceed the corruption bound.

The above omits several technical details, but conveys the main ideas.

Proof (Theorem 19). If $f \geq n/3$ the theorem is trivially true (as asynchronous BA is impossible); thus, we assume $f < n/3$ in what follows. We present the proof assuming f is even and show that in this case, the expected number of messages is at least $c^2 f^2$. The case of odd f can be reduced to the case of even f since any (f, δ)-secure protocol is also an $(f-1, \delta)$-secure protocol.

Let $c = \frac{3\delta - 2}{8\delta}$. Fix an (f, δ)-secure protocol Π whose expected number of messages is less than $c^2 f^2$. Fix a subset $S \subset [n]$ with $|S| = \frac{f}{2}$. Let S' denote the remaining parties. Consider an execution (Ex1) of Π that proceeds as follows: At the start of the execution, an adversary corrupts all parties in S and they immediately abort. The parties in S' remain honest and run Π using input 0. By δ-security of Π we have:

Lemma 20 *In Ex1 all parties in S' output 0 with probability at least δ.*

Now consider an execution (Ex2) of Π involving an adversary $\mathcal{A}$. (As explained in the proof intuition, $\mathcal{A}$'s goal is to make P believe it is running in an execution in which all parties are honest and have input 1, and to make the honest parties in S' believe they are running in Ex1.) At the start of the execution, $\mathcal{A}$ chooses a uniform $P \in S$ and corrupts all parties in S except for P. All parties in S' are initially honest and hold input 0, while P holds input 1. $\mathcal{A}$ maintains two simulated executions that we label *red* and *blue*. (See

Figure 7.) In the blue execution, $\mathcal{A}$ plays the role of all parties other than P; all these virtual parties run Π honestly with input 1. In the red execution, $\mathcal{A}$ simulates an execution in which all parties in S immediately abort, and all parties in S' run Π honestly with input 0. $\mathcal{A}$ uses these two simulations to determine how to interact with the honest parties in the real execution. Specifically, it schedules delivery of messages as follows:

- **S' to P, real execution.** Messages sent by honest parties in S' to P in the real execution are delayed, and delivered only after all honest parties have generated output.
- **P to S', real execution.** When P sends a message to an honest party $Q \in S'$ in the real execution, $\mathcal{A}$ delays the message and then corrupts Q. Once Q is corrupted, $\mathcal{A}$ delivers the message to Q in the real execution (and can then read the message). $\mathcal{A}$ also delivers that same message to Q in the blue simulation.
- **S' to P, blue execution.** When a party $Q \in S'$ sends a message m to P in the blue execution, $\mathcal{A}$ corrupts Q in the real execution (if Q was not already corrupted), and then sends m to P (on behalf of Q) in the real execution. (Messages that Q may have sent previously to P in the real execution continue to be delayed.)
- **S to P, blue execution.** When a party $Q \in S$ sends a message m to P in the blue execution, Q sends m to P in the real execution (recall that parties in $S \setminus \{P\}$ are corrupted in Ex2).
- **S' to S', real execution.** Messages sent by honest parties in S' to other parties in S' in the real execution are delivered normally. If the receiver is corrupted, the message is relayed to $\mathcal{A}$, who simulates this same message in the red execution.
- **S' to $S \setminus \{P\}$, real execution.** Messages sent by honest parties in S' to the (corrupted) parties in $S \setminus \{P\}$ in the real execution are ignored.
- **S' to S', red execution.** If a party $Q \in S'$ is corrupted in the real execution, then whenever a message m is sent by a party Q to another party in S' in the red execution, Q sends m in the real execution.

If $\mathcal{A}$ would ever need to corrupt more than f parties in total, then it simply aborts. (However, the real execution continues without any further interference from $\mathcal{A}$.)

Lemma 21 *In Ex2, the distribution of the joint view of all parties in S' who remain uncorrupted is identical to the distribution of their joint view in Ex1. In particular, with probability at least δ in Ex2 all parties in S' who remain uncorrupted output 0.*

Proof. The only messages received by the parties in S' in either Ex1 or Ex2 are those that arise from an honest execution of Π among the parties in S', all of whom hold input 0. Moreover, in Ex2 the decision as to whether or not a party in S' is corrupted is independent of the joint view of all uncorrupted parties in S'. The final statement follows from Lemma 20. □

We also show that with positive probability, $\mathcal{A}$ does not abort.

Lemma 22 *In Ex2, $\mathcal{A}$ does not abort with probability at least $1 - 4c$.*

Proof. $\mathcal{A}$ aborts if it would exceed the corruption bound. Initially, only the $f/2$ parties in S are corrupted. Let M denote the total number of messages sent either by the parties in S'

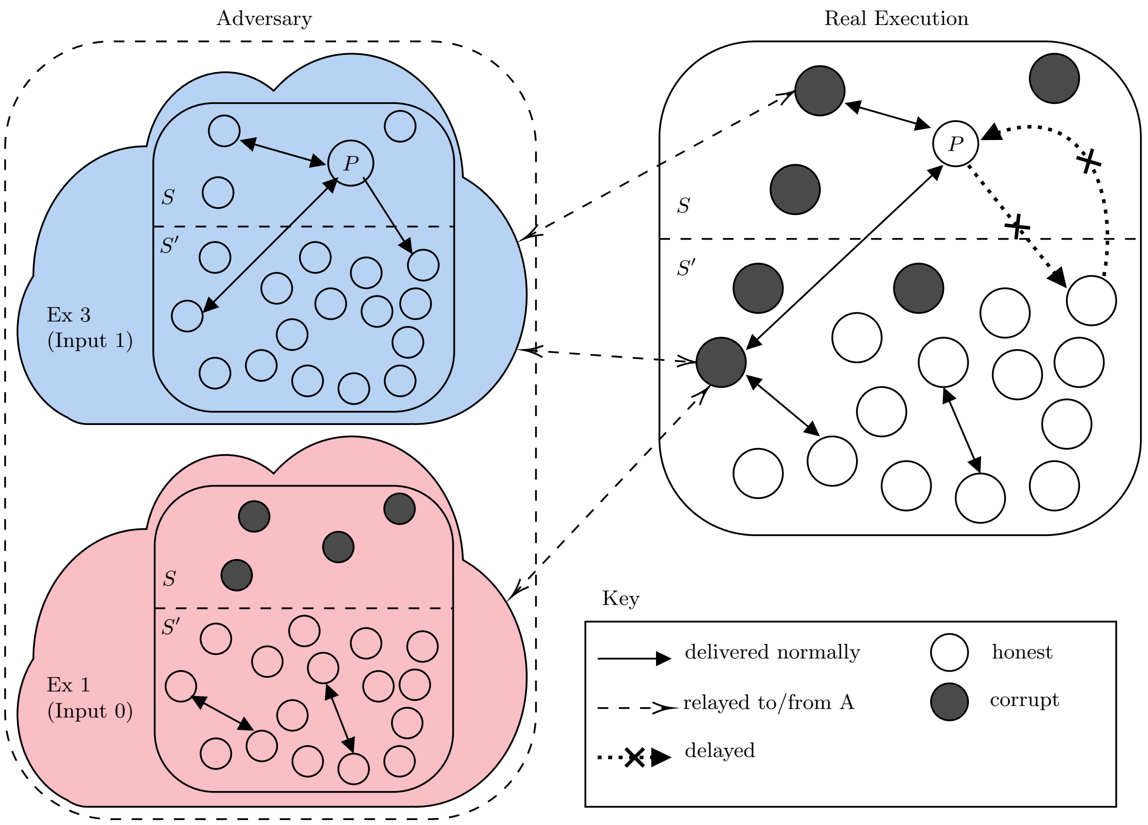

Fig. 7. Adversarial strategy in Ex2. In the real execution (shown at right) corrupted parties in S interact with P as if they are honest with input 1, and ignore honest parties in S'. Corrupted parties in S' interact with P as if they are honest with input 1, and interact with S' as if they are honest with input 0. All messages between P and honest parties in S' are delayed indefinitely. The adversary maintains two simulated executions (shown at left) to determine which messages corrupted parties will send in the real execution.

to the parties in S or by parties in S to parties in S' in the blue execution. By assumption, $\mathbf{Exp}[M] < c^2 f^2$. Let X be the event that $M \leq \frac{c}{2} f^2$. Lemma 30 implies that

$$\Pr[X] \geq \Pr\left[M \leq \frac{\mathbf{Exp}[M]}{2c}\right] \geq 1 - 2c.$$

Let Y be the event that, among the first $cf^2/2$ messages sent by parties in S' to parties in S or vice versa, a uniformly chosen $P \in S$ sends and/or receives at most $f/2$ of those messages. By the pigeonhole principle, at most cf parties in S can receive and/or send $f/2$ or more of those messages, and so $\Pr[Y] \geq 1 - cf/|S| = 1 - 2c$.[3] Thus, $\Pr[X \wedge Y] = \Pr[X] + \Pr[Y] - \Pr[X \cup Y] \geq (1-2c) + (1-2c) - 1 = 1 - 4c$. The lemma follows by observing that when X and Y occur, at most $f/2$ parties in S' are corrupted. $\square$

[3] It is convenient to view the communication between S and S' as an undirected, bipartite multi-graph in which each node represents a party and an edge (U, V) represents a message sent between parties $U \in S$ and $V \in S'$. As the number of edges in this graph is at most $cf^2/2$, there can not be more than cf nodes in S whose total degree is at least $f/2$.

Finally, consider an execution (Ex3) in which a uniform $P \in S$ is chosen and then Π is run honestly with all parties holding input 1.

Lemma 23 *In Ex2, conditioned on the event that $\mathcal{A}$ does not abort, the view of P is distributed identically to the view of P in Ex3. In particular, with probability at least δ in Ex2, P outputs 1.*

Proof. In Ex2, the view of P is determined by the virtual execution in which all parties run Π honestly using input 1. The final statement follows because in Ex3, (f, δ)-security of Π implies that P outputs 1 with probability at least δ. □

We now complete the proof of the theorem. In execution Ex2, let Z_1 be the event that $\mathcal{A}$ does not abort; by Lemma 22, $\Pr[Z_1] \geq 1 - 4c$. Let Z_2 be the event that P does not output 0 in Ex2; using Lemma 23 we have

$$\Pr[Z_2] \geq \Pr[Z_2 \mid Z_1] \cdot \Pr[Z_1] \geq \delta \cdot (1 - 4c).$$

Let Z_3 be the event that all uncorrupted parties in S' output 0 in Ex2. By Lemma 21, $\Pr[Z_3] \geq \delta$. Recalling that $2/3 < \delta < 1$, we see that

$$\Pr[Z_2 \wedge Z_3] = \Pr[Z_2] + \Pr[Z_3] - \Pr[Z_2 \cup Z_3] \geq 2\delta - 4c\delta - 1 = \frac{\delta}{2} > \frac{1}{3} > 1 - \delta,$$

contradicting (f, δ)-security of Π. □

References

1. Ittai Abraham, T.-H. Hubert Chan, Danny Dolev, Kartik Nayak, Rafael Pass, Ling Ren, and Elaine Shi. Communication complexity of byzantine agreement, revisited. In Peter Robinson and Faith Ellen, editors, *38th ACM PODC*, pages 317–326. ACM, July / August 2019.
2. Mihir Bellare, Alexandra Boldyreva, Anand Desai, and David Pointcheval. Key-privacy in public-key encryption. In Colin Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 566–582. Springer, Heidelberg, December 2001.
3. Michael Ben-Or, Ran Canetti, and Oded Goldreich. Asynchronous secure computation. In *25th ACM STOC*, pages 52–61. ACM Press, May 1993.
4. 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.
5. Dan Boneh, Rosario Gennaro, Steven Goldfeder, Aayush Jain, Sam Kim, Peter M. R. Rasmussen, and Amit Sahai. Threshold cryptosystems from threshold fully homomorphic encryption. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 565–596. Springer, Heidelberg, August 2018.
6. G. Bracha. Asynchronous Byzantine agreement protocols. *Information and Computation*, 75:130–143, 1987.
7. Gabriel Bracha and Sam Toueg. Asynchronous consensus and broadcast protocols. *Journal of the ACM*, 32(4):824–840, 1985.
8. Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. Secure and efficient asynchronous broadcast protocols. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 524–541. Springer, Heidelberg, August 2001.
9. Christian Cachin, Klaus Kursawe, and Victor Shoup. Random oracles in constantipole: practical asynchronous byzantine agreement using cryptography (extended abstract). In Gil Neiger, editor, *19th ACM PODC*, pages 123–132. ACM, July 2000.

10. Ran Canetti. Studies in secure multiparty computation and applications. *Journal of Cryptology*, 1996.
11. Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, January 2000.
12. Ran Canetti and Tal Rabin. Fast asynchronous byzantine agreement with optimal resilience. In *25th ACM STOC*, pages 42–51. ACM Press, May 1993.
13. Ashish Choudhury and Arpita Patra. Optimally resilient asynchronous mpc with linear communication complexity. In *Proceedings of the 2015 International Conference on Distributed Computing and Networking*, pages 1–10, 2015.
14. 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.
15. Shir Cohen, Idit Keidar, and Alexander Spiegelman. Not a COINcidence: Sub-quadratic asynchronous Byzantine agreement WHP. 2020.
16. Alfredo De Santis, Giovanni Di Crescenzo, Rafail Ostrovsky, Giuseppe Persiano, and Amit Sahai. Robust non-interactive zero knowledge. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 566–598. Springer, Heidelberg, August 2001.
17. Danny Dolev and Rüdiger Reischuk. Bounds on information exchange for Byzantine agreement. *Journal of the ACM*, 32(1):191–204, 1985.
18. Paul Feldman and Silvio Micali. Optimal algorithms for byzantine agreement. In *20th ACM STOC*, pages 148–161. ACM Press, May 1988.
19. Juan A. Garay, Jonathan Katz, Ranjit Kumaresan, and Hong-Sheng Zhou. Adaptively secure broadcast, revisited. In Cyril Gavoille and Pierre Fraigniaud, editors, *30th ACM PODC*, pages 179–186. ACM, June 2011.
20. Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009.
21. Yue Guo, Rafael Pass, and Elaine Shi. Synchronous, with a chance of partition tolerance. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part I*, volume 11692 of *LNCS*, pages 499–529. Springer, Heidelberg, August 2019.
22. 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.
23. 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.
24. Valerie King and Jared Saia. Breaking the $O(n^2)$ bit barrier: scalable byzantine agreement with an adaptive adversary. In Andréa W. Richa and Rachid Guerraoui, editors, *29th ACM PODC*, pages 420–429. ACM, July 2010.
25. Valerie King, Jared Saia, Vishal Sanwalani, and Erik Vee. Scalable leader election. In *17th SODA*, pages 990–999. ACM-SIAM, January 2006.
26. Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(382–401), July 1982.
27. M.Fischer, N. Lynch, and M. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, April 1985.
28. Silvio Micali. Very simple and efficient byzantine agreement. In Christos H. Papadimitriou, editor, *ITCS 2017*, volume 4266, pages 6:1–6:1, 67, January 2017. LIPIcs.
29. Silvio Micali and Vinod Vaikuntanathan. Optimal and player-replaceable consensus with an honest majority. Technical report, MIT, 2017.
30. Achour Mostéfaoui, Moumen Hamouma, and Michel Raynal. Signature-free asynchronous byzantine consensus with $t < n/3$ and $O(n^2)$ messages. In Magnús M. Halldórsson and Shlomi Dolev, editors, *33rd ACM PODC*, pages 2–9. ACM, July 2014.
31. Rafael Pass and Elaine Shi. The sleepy model of consensus. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part II*, volume 10625 of *LNCS*, pages 380–409. Springer, Heidelberg, December 2017.
32. Arpita Patra, Ashish Choudhury, and C. Pandu Rangan. Efficient asynchronous multiparty computation with optimal resilience. Cryptology ePrint Archive, Report 2008/425, 2008. `http://eprint.iacr.org/2008/425`.

33. 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.
34. Michael O. Rabin. Randomized byzantine generals. In *24th FOCS*, pages 403–409. IEEE Computer Society Press, November 1983.
35. Matthieu Rambaud. Lower bounds for authenticated randomized byzantine consensus under (partial) synchrony: the limits of standalone digital signatures. Unpublished Manuscript.
36. 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.

A Additional Definitions

A.1 Secret Sharing

We recall the standard notion of perfect secret sharing.

Definition 24 *A (t, n)-threshold secret sharing scheme* TSS *is a tuple of algorithms* $(\mathsf{Share}, \mathsf{Rec})$ *such that:*

- *On input a message s, the randomized* secret sharing algorithm Share *outputs shares $s_1, \ldots, s_n$.*
- *On input shares $s_{i_1}, \ldots, s_{i_{t+1}}$, the deterministic* reconstruction algorithm Rec *outputs a reconstructed value s or $\bot$.*

We require that TSS *satisfy:*

Perfect secrecy: *For any $I \subset [n]$ with $|I| = t$, the distribution*

$$\{s_1, \ldots, s_n \leftarrow \mathsf{Share}(s) : \{s_i\}_{i \in I}\}$$

is independent of s. (Note that because we require perfect secrecy, this holds even if I is chosen adaptively.)

Reconstruction: *For any s, and any $I \subset [n]$ with $|I| = t + 1$*

$$\Pr[s_1, \ldots, s_n \leftarrow \mathsf{Share}(s) : \mathsf{Rec}(\{(i, s_i)\}_{i \in I}) = s] = 1.$$

A.2 Threshold Fully Homomorphic Encryption

The following definitions are adapted from Gentry's thesis [20].

Definition 25 *A fully homomorphic encryption scheme (FHE) consists of four algorithms:*

- *Key generation: $(ek, dk) = \mathsf{KGen}(1^\kappa)$, where ek is the public encryption key and dk is the decryption key.*
- *Encryption: $c = \mathsf{Enc}_{ek}(m; r)$ denotes an encryption with key ek of a plaintext m with randomness r, to obtain ciphertext c.*
- *Decryption: $m = \mathsf{Dec}_{dk}(c)$ denotes a decryption of ciphertext c with key dk to obtain plaintext m.*
- *Homomorphic evaluation: $c = \mathsf{Eval}_{ek}(C, c_1, \ldots, c_n)$ denotes the evaluation of a circuit C over a tuple of ciphertexts $(c_1, \ldots, c_n)$ to obtain c.*

As usual, we require that the FHE is correct when evaluating a circuit C, and compact, meaning intuitively that the ciphertext size should not grow through homomorphic operations [**?**, **?**].

Definition 26 *A FHE scheme is correct and compact if it satisfies the following properties:*

- *Correctness: For any circuit C and ciphertexts c_i, where $m_i = \mathsf{Dec}_{dk}(c_i)$:*

$$\Pr[\mathsf{Dec}_{dk}(\mathsf{Eval}_{ek}(C, c_1, \ldots, c_n)) = C(m_1, \ldots, m_n)] = 1 - \mathtt{neg}(\kappa),$$

  where $(ek, dk) = \mathsf{KGen}(1^\kappa)$.
- *Compactness: There is a polynomial p, such that for any key pair $(ek, dk) = \mathsf{KGen}(1^\kappa)$, any circuit C and all ciphertexts c_i, the size of the output $\mathsf{Eval}_{ek}(C, c_1, \ldots, c_n)$ is not more than $p(\kappa)$ bits, independent of the depth and the size of the circuit.*

In our protocol we require in addition that the FHE scheme to have threshold decryption (e.g. [5]), meaning that KGen generates a public key ek as well as a threshold secret sharing of the secret key $(dk_1, \ldots, dk_n)$ such that decrypting c using dk_i outputs a share of the message.

Definition 27 *A threshold FHE scheme is an FHE scheme with the following two additional properties:*

- *The key generation algorithm is parameterized by (t, n) and outputs $(ek, dk) = \mathsf{KGen}_{(t,n)}(1^\kappa)$, where dk is a (t, n)-threshold secret sharing of the secret key $(dk_1, \ldots, dk_n)$.*
- *Given a ciphertext c and a share dk_i, there is a share-decryption algorithm $d_i = \mathsf{DecShare}_{dk_i}(c)$, such that $(d_1, \ldots, d_n)$ is a (t, n)-threshold secret sharing of the plaintext $m = \mathsf{Dec}_{dk}(c)$. We denote the reconstruction algorithm that receives $t + 1$ decryption shares by $m = \mathsf{Rec}(\{d_i\})$.*

A.3 Anonymous Public-Key Encryption

We recall the definition of anonymous public-key encryption from [2].

Definition 28 *A public-key encryption scheme $\mathcal{PE} = (\mathcal{G}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ consists of four algorithms:*

- *Common-key generation: $I = \mathcal{G}(1^\kappa)$, where I is a common key, which might be just the security parameter κ or include some additional information.*
- *Key generation: $(\mathsf{pk}, \mathsf{sk}) = \mathcal{K}(I)$, takes as input the common key I and returns a pair of keys, where pk is the public key and sk is the secret key.*
- *Encryption: $c = \mathcal{E}_{\mathsf{pk}}(m; r)$ denotes an encryption with key pk of a plaintext m with randomness r, to obtain ciphertext c.*
- *Decryption: $m = \mathcal{D}_{\mathsf{sk}}(c)$ denotes a decryption of ciphertext c with key sk to obtain plaintext m.*

Definition 29 (IK-CPA) *Let $\mathcal{PE} = (\mathcal{G}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ be an encryption scheme. Let $b \in \{0, 1\}$, and let A_{cpa} be an adversary that runs in the experiment described in Figure 8. The scheme $\mathcal{PE}$ is IK-CPA secure if:*

$$\Pr[\mathbf{Exp}^1_{\mathcal{PE}, A_{\mathsf{cpa}}}(\kappa) = 1] - \Pr[\mathbf{Exp}^0_{\mathcal{PE}, A_{\mathsf{cpa}}}(\kappa) = 1] \leq \mathtt{neg}(\kappa).$$

Experiment $\mathbf{Exp}^{b}_{\mathcal{PE},A_{\mathsf{cpa}}}(\kappa)$

```
1. $I = \mathcal{G}(\kappa)$
2. $(\mathsf{pk}_0, \mathsf{sk}_0) = \mathcal{K}(I)$; $(\mathsf{pk}_1, \mathsf{sk}_1) = \mathcal{K}(I)$
3. $(x, s) = A_{\mathsf{cpa}}(\mathtt{find}, \mathsf{pk}_0, \mathsf{pk}_1)$
4. $y = \mathcal{E}_{\mathsf{pk}_b}(x)$
5. $d = A_{\mathsf{cpa}}(\mathtt{guess}, y, s)$
6. Return $d$
```

Fig. 8. Security game for IK-CPA.

B Concentration Inequalities

We briefly recall the following well known concentration inequalities.

Lemma 30 (Markov Bound) *Let X be a non-negative, real-valued random variable. Then*

$$\Pr[X \geq a] \leq \frac{E[X]}{a}.$$

Lemma 31 (Chernoff Bound) *Let $X_1, ..., X_n$ be independent Bernoulli random variables with parameter p. Let $X := \sum_i X_i$ and $\mu := E[X] = p \cdot n$. Then, for $\delta \in [0, 1]$*

- $\Pr[X \leq (1-\delta) \cdot \mu] \leq e^{-\delta^2 \mu/2}$.
- $\Pr[X \geq (1+\delta) \cdot \mu] \leq e^{-\delta^2 \mu/(2+\delta)}$.

We prove a useful lemma. Let $\chi_{s,n}$ denote the distribution that samples a subset of the parties $P_1, \ldots, P_n$, where each party is chosen independently with probability s/n.

Lemma 32 *Fix $s \leq n$ and $0 < \epsilon \leq 1/2$, and let $f < n \cdot \frac{1-2\epsilon}{3}$ be an upper bound on the number of corrupted parties. Let $C \leftarrow \chi_{s,n}$. Then:*

1. *C has more than $\frac{s(2+\epsilon)}{3}$ honest parties except with probability at most $p_1(s) := e^{-\frac{\epsilon^2 s}{12(1+\epsilon)}}$.*
2. *C has fewer than $\frac{s(2+4\epsilon)}{3}$ honest parties except with probability at most $p_2(s) := e^{-\frac{2\epsilon^2 s}{6+9\epsilon}}$.*
3. *C has fewer than $\frac{s(1-\epsilon)}{3}$ corrupted parties except with probability at most $p_3(s) := e^{-\frac{\epsilon^2 s}{6-9\epsilon}}$.*

Proof. Let $H \subseteq [n]$ be the indices of the honest parties. Let X_j be the Bernoulli random variable indicating if $P_j \in C$, so $\Pr[X_j = 1] = s/n$. Let $X := \sum_{j \in H} X_j$, and $X' := \sum_{j \notin H} X_j$. Since $|H| \geq n - f$, the expected number of honest parties in C is $\mu := E[X] \geq s \cdot \frac{n-f}{n} = s \cdot \frac{2+2\epsilon}{3}$. Similarly, the expected number of corrupted parties in C is $\mu' := E[X'] \geq s \cdot \frac{f}{n} = s \cdot \frac{1-2\epsilon}{3}$. Thus:

1. Setting $\delta = \frac{\epsilon}{2+2\epsilon}$ in Lemma 31 and simplifying yields:

$$\Pr\left[X \leq \frac{s(2+\epsilon)}{3}\right] \leq e^{-\frac{\epsilon^2 s}{12(1+\epsilon)}} = p_1(s).$$

2. Setting $\delta = \frac{\epsilon}{\epsilon+1}$ in Lemma 31 and simplifying yields:

$$\Pr\left[X \geq \frac{s(2+4\epsilon)}{3}\right] \leq e^{-\frac{2\epsilon^2 s}{6+9\epsilon}} = p_2(s).$$

3. Setting $\delta = \frac{\epsilon}{1-2\epsilon}$ in Lemma 31 and simplifying yields:

$$\Pr\left[X' \geq \frac{s(1-\epsilon)}{3}\right] \leq e^{-\frac{\epsilon^2 s}{6-9\epsilon}} = p_3(s).$$

□

C Graded Consensus

Our graded consensus protocol is described in Figure 10. The protocol is based on the graded consensus protocol by Canetti and Rabin [12]. In the first phase, parties in C_1 reliably broadcast their input value. All parties wait to complete $\kappa - t$ broadcasts, and then set their $\mathsf{prepare}_2$ value as the majority value among these outputs. In the second phase, parties in C_2 reliably broadcast their $\mathsf{prepare}_2$ value. All parties wait for $\kappa - t$ broadcasts that output consistent values with the broadcasted inputs from the first phase to terminate, and sets the $\mathsf{prepare}_3$ value as the majority of such outputs. In the third phase, parties in C_3 reliably broadcast their $\mathsf{prepare}_3$ value. Then, parties wait for $\kappa - t$ broadcasts that are consistent with the received $\mathsf{prepare}_2$ values to terminate. The idea is that if all $\mathsf{prepare}_2$ values received by a party are the same value b (this party outputs $(b, 1)$), then it is guaranteed that every other honest party eventually receives the same $\mathsf{prepare}_2$ values (and so outputs $(b, 1)$) as well, or eventually receives the same $\mathsf{prepare}_3$ values (and outputs $(b, 0)$).

Graded Consensus Setup

The dealer does, for $i \in 1, 2, 3$:

- For each $P \in [n]$, choose P with probability κ/n. Let C_i denote the set of chosen parties.
- Each party in C has available a setup for Π_{RBC} as sender.

Fig. 9. A Setup Protocol for GC

Lemma 33 *Let P_i and P_j be honest parties, and denote as $S_{1,j}, S_{1,i}$ the respective sets S_1 of those parties in an execution of Π_{GC}. Then with overwhelming probability, eventually $S_{1,j} \subseteq S_{1,i}$.*

Proof. Suppose that $(b_\ell, \ell) \in S_{1,j}$. With overwhelming probability, this implies that P_j output $(\mathsf{prepare}_1, \ell, b_\ell)$ in RBC_ℓ where P_ℓ in C_1. By the consistency property of RBC, P_i either eventually outputs $(\mathsf{prepare}_1, \ell, b_\ell)$ in RBC_ℓ and hence adds (b_ℓ, ℓ) to $S_{1,i}$ or terminates Π_{GC} (with overwhelming probability). Thus, every value in $S_{1,j}$ is eventually added to $S_{1,i}$ (and hence $S_{1,j} \subseteq S_{1,i}$), with overwhelming probability.

Lemma 34 *Let P_i and P_j be honest parties, and denote as sets $S_{2,j}, S_{2,i}$ the respective sets S_2 of those parties in an execution of Π_{GC}. Then with overwhelming probability, eventually $S_{2,j} \subseteq S_{2,i}$.*

Proof. Denote as $S_{1,j}, S_{1,i}$ the respective sets S_1 of parties P_i and P_j and suppose that $(b_\ell, \ell) \in S_{2,j}$. With overwhelming probability, this implies that P_j output $(\mathsf{prepare}_2, \ell, \hat{S}_{1,\ell}, b_\ell)$

Protocol Π_{GC}

We describe the protocol from the point of view of a party with input $v \in \{0,1\}$.

1. Initialize $\hat{S}_1 = \hat{S}_2 = S_1 = S_2 = S_3 := \emptyset, b_1 := v, b_2 = b_3 := \bot$
2. If $P_i \in C_1$: participate in RBC_i as the sender with input $(\mathsf{prepare}_1, i, b_1)$. Participate in each protocol RBC_j, $j \neq i, j \in C_1$, as the receiver.
3. Upon receiving output $(\mathsf{prepare}_1, j, b_j)$ in RBC_j where $j \in C_1$, add (b_j, j) to S_1.
4. When $|S_1| = \kappa - t$, do: Set $\hat{S_1} = S_1$ and set b_2 to the majority bit among values in $\hat{S}_1$. Participate in RBC_i as the sender with input $(\mathsf{prepare}_2, i, \hat{S}_1, b_2)$ if $P_i \in C_2$. Participate in each protocol RBC_j, $j \neq i, j \in C_2$, as the receiver.
5. Upon receiving output $(\mathsf{prepare}_2, j, \hat{S}_{1,j}, b_j)$ in RBC_j do: if $j \in C_2$, $\hat{S}_{1,j} \subseteq S_1$, and b_j is the majority bit among $\hat{S}_{1,j}$, add (b_j, j) to S_2.
6. When $|S_2| = \kappa - t$, do: Set $\hat{S_2} = S_2$ and set b_3 to the majority bit among values in $\hat{S}_2$. Participate in RBC_i as the sender with input $(\mathsf{prepare}_3, i, \hat{S}_2, b_3)$ if $P_i \in C_3$. Participate in each protocol RBC_j, $j \neq i, j \in C_3$, as the receiver.
7. Upon receiving output $(\mathsf{prepare}_3, j, \hat{S}_{2,j}, b_j)$ in RBC_j do: if $j \in C_3$, $\hat{S}_{2,j} \subseteq S_2$, and b_j is the majority bit among $\hat{S}_{2,j}$, add (b_j, j) to S_3.
8. When $|S_3| = \kappa - t$, do:
   - If there exists $b \in \{0,1\}$ s.t. for all $(b_j, j) \in \hat{S}_2$ it holds that $b_j = b$, then output $(b, 1)$.
   - Else if there exists $b \in \{0,1\}$ s.t. for all $(b_j, j) \in S_3$ it holds that $b_j = b$, then output $(b, 0)$.
   - Else output $(0, 0)$.

Fig. 10. A protocol for graded consensus.

in RBC_ℓ where P_ℓ in C_2, $\hat{S}_{1,\ell} \subseteq S_{1,j}$, and b_ℓ is the majority bit among values in $\hat{S}_{1,\ell}$. By the consistency property of RBC and the previous lemma, P_i either eventually outputs $(\mathsf{prepare}_2, \ell, b_\ell)$ in RBC_ℓ and $\hat{S}_{1,\ell} \subseteq S_{1,j} \subseteq S_{1,i}$ or or terminates Π_{GC} (with overwhelming probability). Once the former happens, P_i adds (b_ℓ, ℓ) to $S_{2,i}$. Thus, every value in $S_{2,j}$ is eventually added to $S_{2,i}$ (and hence $S_{2,j} \subseteq S_{2,i}$), with overwhelming probability.

Lemma 35 *With overwhelming probability, for every honest party P_i, S_1, S_2, and S_3 eventually are of size $\kappa - t$.*

Proof. Let P_i be an honest party. We analyze the size of the sets in sequence.

- S_1: By validity, P_i outputs in all RBC instances corresponding to honest parties in C_1 with overwhelming probability and adds a corresponding tuple to S_1 as a result. Since by Lemma 32, at least $\kappa - t$ parties in C_1 are honest, the claim for S_1 follows.
- S_2: Since all honest parties S_1 sets eventually become of size $\kappa - t$, all honest parties P_j in C_2 eventually send a message $(\mathsf{prepare}_2, \ell, \hat{S}_{1,j}, b_j)$ in RBC_j. By Lemma 33, $\hat{S}_{1,j} \subseteq S_i$, with overwhelming probability, eventually. This implies that all checks for the instance RBC_j are satisfied in Step 5 with overwhelming probability at some point. By validity of RBC and Lemma 32, P_i eventually adds at least $\kappa - t$ tuples of the form (b_j, j) to S_2 in this manner, with overwhelming probability.
- S_3: The claim on S_3 follows in an analogous fashion as the previous one.

Lemma 36 *If all honest parties P_i in C_1 send $(\mathsf{prepare}_1, i, b)$ in RBC_i, then no honest party adds a tuple $(1 - b, j)$ to S_2, with overwhelming probability.*

Proof. Assume toward a contradiction that all honest parties P_i in C_1 send $(\mathsf{prepare}_1, i, b)$ in RBC_i and there is an honest party P that adds a tuple $(1-b, j)$ to S_2. This implies that it received $(\mathsf{prepare}_2, j, \hat{S}_{1,j}, 1-b)$ in RBC_j, where $j \in C_2$ and $1-b$ is the majority value among values in $\hat{S}_{1,j}$, and $|\hat{S}_{1,j}| \geq \kappa - t$. By assumption, $\hat{S}_{1,j} \subseteq S_1$, and so P outputs in all instances of RBC that correspond to $\hat{S}_{1,j}$. By validity and since all honest parties in C_1 send $(\mathsf{prepare}_1, i, b)$ in RBC_i, at least $\kappa - 2t > t$ of the tuples in $\hat{S}_{1,j}$ are of the form (b, j) and at most most t tuples in $\hat{S}_{1,j}$ are of the form $(1-b, j)$, with overwhelming probability. This contradicts that b is the majority value among values in $\hat{S}_{1,j}$.

Lemma 37 *If an honest party P_i has $\hat{S}_2$ such that all $(b_j, j) \in \hat{S}_2$, $b_j = b$ for some $b \in \{0,1\}$, then each honest party P_j has for all $(b_j, j) \in S_3$, $b_j = b$, with overwhelming probability.*

Proof. Let $\hat{S}_2$ be the set of P_i at Step 6 that contains consistently the same value, i.e., such that all $(b_j, j) \in \hat{S}_2$, $b_j = b$.

We argue that the set S_3 of P_j consistently contains b as well, since any set $\hat{S}_{2,k}$ that P_j accepts in Step 7 has the majority bit b.

Assume that there exists a set $\hat{S}_{2,k}$ that has a different majority bit than b. Then, the sets $\hat{S}_2$ of P_i and $\hat{S}_{2,k}$ both have size at least $\kappa - t$. Since there are at most t dishonest parties by Lemma 32, and by validity of the reliable broadcast, at least $\kappa - 2t$ values came from $\mathsf{prepare}_2$ messages from honest parties.

Since honest parties only send one prepare value, this implies that there are $2\kappa - 4t = 2\kappa/3 \cdot (1 + 2\epsilon)$ distinct honest parties, which is in contradiction with Lemma 32. Hence, the majority bits in all accepted sets $\hat{S}_2^{2,k}$ is the same and the statement follows.

Lemma 38 *Π_{GC} satisfies graded-validity.*

Proof. By Lemma 35, every honest party accumulates a set S_3 of size $\kappa - t$ and hence outputs a value and a grade.

Assume all honest parties have input v. This implies that all honest parties P_i in C_1 send $(\mathsf{prepare}_1, i, v)$. By Lemma 36, no honest parties add $(1-v, j)$ to S_2. Hence, every honest party outputs $(v, 1)$.

Lemma 39 *All honest parties generate output in Π_{GC}.*

Proof. This follows from the fact that every honest party eventually accumulates S_3 with size $\kappa - t$ by Lemma 35.

Lemma 40 *Π_{GC} satisfies graded consistency.*

Proof. Termination is argued in Lemma 39. Let P_i and P_j be two honest parties. Assume that P_i outputs $(v, 1)$.

Let $\hat{S}_2^i$ denote the set $\hat{S}_2$ that P_i accumulates in Step 6. Then $\hat{S}_2^i$ contains consistently the same bit. By Lemma 34, P_j cannot output $(1-v, 1)$. Moreover, by Lemma 37, the set $\hat{S}_3^j$ consistently contains v, and hence P_j outputs $(v, 1)$ or $(v, 0)$.

D Proof of Theorem 14

Security of $\mathit{\Pi}_{\mathsf{BA}}$ follows immediately from the following lemmas. In the following, we say that parties input a value when they decide on their input, even though parties may not actually send a message upon inputting (if they are not on the appropriate committee). For example, we consider P_i to have input b to $\mathit{\Pi}_{\mathsf{RC}}$ upon reaching step 5 of $\mathit{\Pi}_{\mathsf{BA}}$, because that is the point when P_i's input value for $\mathit{\Pi}_{\mathsf{RC}}$ is determined (even if P_i does not actually send a message in $\mathit{\Pi}_{\mathsf{RC}}$ until later).

Lemma 41 *If at most f parties are corrupted during an execution of $\mathit{\Pi}_{\mathsf{BA}}$, then some honest party sets $\mathsf{ready} = \mathsf{true}$ after an expected constant number of iterations, and some honest party sets $\mathsf{ready} = \mathsf{true}$ with overwhelming probability after at most κ iterations.*

Proof. Assume at most f parties are corrupted, and consider an iteration k of $\mathit{\Pi}_{\mathsf{BA}}$ such that no honest party set $\mathsf{ready} = \mathsf{true}$ in any round $k' < k$. (This is trivially true of the first iteration). We begin by showing that in such an iteration, all honest parties input the same value v to the second execution of $\mathit{\Pi}_{\mathsf{GC}}$ with probability at least $1/2$. Call this event $E1$. We consider two cases. In the first case, suppose that some honest party outputs $(b, 1)$ in the first execution of $\mathit{\Pi}_{\mathsf{GC}}$ during iteration k. (Call this event $E2$.) Graded consistency of $\mathit{\Pi}_{\mathsf{GC}}$ guarantees that each other honest party outputs $(b, 1)$ or $(b, 0)$ during that execution. We can see that b is chosen independently of Coin_k with overwhelming probability, because b is determined prior to the first honest party invoking $\mathsf{CoinFlip}(i)$. Therefore, with probability $1/2$, $\mathsf{Coin}_k = b$. Hence, all parties (regardless of output grade) input b to the second execution of $\mathit{\Pi}_{\mathsf{GC}}$. Thus, the probability that $E1$ occurs given that $E2$ occurs is at least $1/2$. For the second case, suppose that no honest party outputs with grade 1 in the first execution $\mathit{\Pi}_{\mathsf{GC}}$. (Call this event $E3$.) Then all honest parties input Coin_k to the second execution of $\mathit{\Pi}_{\mathsf{GC}}$, and by Lemma 13, Coin_k is the same for all honest parties. Thus, the probability that $E1$ occurs given that $E3$ occurs is 1. Because either $E2$ or $E3$ must occur and the two events are disjoint, we can see that $E1$ occurs in this round with probability at least $1/2$.

If $E1$ does occur in iteration k and no party has yet set $\mathsf{ready} = \mathsf{true}$, then by graded validity, certainly at least one honest party will output $(g, 1)$ from the second execution of $\mathit{\Pi}_{\mathsf{GC}}$, and that party will then set $\mathsf{ready} = \mathsf{true}$. Even if $E1$ does not occur in round k, it is still possible that some honest party will output $(g, 1)$, and thus set $\mathsf{ready} = \mathsf{true}$. Thus, in any iteration k of $\mathit{\Pi}_{\mathsf{BA}}$ such that no honest party set $\mathsf{ready} = \mathsf{true}$ in any round $k' < k$, the probability that some honest party sets $\mathsf{ready} = \mathsf{true}$ is at least $\Pr[E1] = 1/2$. Therefore, after an expected constant number of iterations, some honest party sets $\mathsf{ready} = \mathsf{true}$, and some honest party sets $\mathsf{ready} = \mathsf{true}$ with overwhelming probability after at most κ iterations.

Lemma 42 *If at most f parties are corrupted during an execution of $\mathit{\Pi}_{\mathsf{BA}}$, and if some honest party inputs b to $\mathit{\Pi}_{\mathsf{RC}}$ in iteration k, then (1) all honest parties who input a value to an execution of $\mathit{\Pi}_{\mathsf{GC}}$ in iteration $k' \geq k + 1$ must input b, and (2) all honest parties who input a value to $\mathit{\Pi}_{\mathsf{RC}}$ in iteration $k' \geq k$ must input b.*

Proof. Assume at most f parties are corrupted during an execution of $\mathit{\Pi}_{\mathsf{BA}}$. Consider the first iteration k in which some honest party P_i sets $\mathsf{ready} = \mathsf{true}$, and let b be P_i's input to $\mathit{\Pi}_{\mathsf{RC}}$. P_i must have received $(g, 1)$ from the second execution of $\mathit{\Pi}_{\mathsf{GC}}$ during round k. By graded consistency, all other honest parties who receive output from this execution of $\mathit{\Pi}_{\mathsf{GC}}$ must receive (b, g), with $g = 0$ or $g = 1$. Thus, any parties who input to $\mathit{\Pi}_{\mathsf{RC}}$ in this iteration must input b. Moreover, any honest parties who decide on an input to the first execution of $\mathit{\Pi}_{\mathsf{GC}}$ in iteration $k+1$ must input b. Graded validity thus ensures that a party who receives output from that execution of $\mathit{\Pi}_{\mathsf{GC}}$ will receive $(b, 1)$, causing them to input b to the next execution, and so on for as long as they continue to receive output and provide input; thus, we see that an honest party who provides input to $\mathit{\Pi}_{\mathsf{RC}}$ in *any* future iteration must also input b.[4]

Lemma 43 *If at most f parties are corrupted during an execution of $\mathit{\Pi}_{\mathsf{BA}}$, and if some honest party sets* $\mathsf{ready} = \mathsf{true}$ *and inputs a bit b to $\mathit{\Pi}_{\mathsf{RC}}$, then all honest parties terminate with output b.*

Proof. Let k be the first iteration in which some honest party sets $\mathsf{ready} = \mathsf{true}$ and inputs a bit b to $\mathit{\Pi}_{\mathsf{RC}}$. By Lemma 42, we see that any honest party who inputs a value to $\mathit{\Pi}_{\mathsf{RC}}$ must input b, and furthermore, all honest parties who input a value to an execution of $\mathit{\Pi}_{\mathsf{GC}}$ in iteration $k' \geq k+1$ must input b. We consider two cases: either no honest party terminates before all honest parties receive output from the second execution of $\mathit{\Pi}_{\mathsf{GC}}$ in iteration $k+1$, or some honest party terminates before then.

In the first case, graded validity of $\mathit{\Pi}_{\mathsf{GC}}$ ensures that all honest parties eventually receive $(b, 1)$ as output, and thus all parties either input b to $\mathit{\Pi}_{\mathsf{RC}}$ at this point, or they have already done so. By validity of $\mathit{\Pi}_{\mathsf{RC}}$, all honest parties eventually output b and terminate.[5]

In the second case, some honest party P_i terminates before all honest parties have received output from $\mathit{\Pi}_{\mathsf{GC}}$ in iteration $k+1$. By validity of $\mathit{\Pi}_{\mathsf{RC}}$, P_i must have output b. It is possible that $\mathit{\Pi}_{\mathsf{GC}}$ will lose liveness; however, consistency of $\mathit{\Pi}_{\mathsf{RC}}$ guarantees that all parties will eventually output b.

Lemma 44 *$\mathit{\Pi}_{\mathsf{BA}}$ is an f-secure BA protocol.*

Proof. Assume at most f parties are corrupted during an execution of $\mathit{\Pi}_{\mathsf{BA}}$. By Lemma 41, some honest party eventually sets $\mathsf{ready} = \mathsf{true}$ and inputs a bit b to $\mathit{\Pi}_{\mathsf{RC}}$. It immediately follows from Lemma 43 that all honest parties terminate. Furthermore, they terminate with output b, proving f-consistency.

Next, at most f parties are corrupted during an execution of $\mathit{\Pi}_{\mathsf{BA}}$, and assume all honest parties input v. By graded validity of $\mathit{\Pi}_{\mathsf{GC}}$, all honest parties receive $(v, 1)$ as output from the first execution of $\mathit{\Pi}_{\mathsf{GC}}$ in iteration 1, ignore the coin, and input v to the second execution of $\mathit{\Pi}_{\mathsf{GC}}$. Again, graded validity implies that if a party receives output (b, g) from the second execution of $\mathit{\Pi}_{\mathsf{GC}}$, we must have $(b, g) = (v, 1)$. Consider the first honest party P_i to receive output from the second execution $\mathit{\Pi}_{\mathsf{GC}}$ in iteration 1. P_i will set $\mathsf{ready} = \mathsf{true}$ and input v to $\mathit{\Pi}_{\mathsf{RC}}$. By Lemma 43, we see that all honest parties will output v. This proves f-validity.

[4] Note that since $\mathit{\Pi}_{\mathsf{GC}}$ is an asynchronous protocol, even if some parties terminate, security of $\mathit{\Pi}_{\mathsf{GC}}$ ensures that parties will only ever input b in future rounds. However, if some parties terminate, it is possible for $\mathit{\Pi}_{\mathsf{GC}}$ to lose liveness for the remaining parties, in which case they may not input anything.

[5] In this case, we have assumed that no honest parties terminate before the second execution $\mathit{\Pi}_{\mathsf{GC}}$, and therefore we do not have to worry about $\mathit{\Pi}_{\mathsf{GC}}$ losing liveness.

E VACS Protocol and Analysis

We formally describe the protocol $\Pi_{\mathsf{VACS}}^{\ell,Q}$ sketched in Section 6.1, and prove it secure.

Protocol $\Pi_{\mathsf{VACS}}^{\ell,Q}$

We describe the protocol from the point of view of a party P_i with input v_i. Given as set up the secret random committee C where each party is independently added to the committee with probability $\frac{3}{(2+\epsilon)n}\ell$:

1. If $P_i \in C$: participate in RBC_i as the sender. Otherwise, participate in each protocol RBC_j, $j \neq i$, as the receiver.
2. On output v from RBC_j, if $Q(v) = 1$ and an input has not yet been provided to BA_j, then input 1 to BA_j.
3. When ℓ of the protocols BA have output 1, input 0 to each instance BA_j if it has not yet been provided input.
4. Once all instances of BA have been completed, let $\mathsf{CoreSet}$ be the indices of each BA that delivered 1. Wait for the output v_j for each RBC_j, $j \in \mathsf{CoreSet}$. Finally output $\cup_{j\in\mathsf{CoreSet}}\{v_j\}$.

Fig. 11. A VACS protocol with ℓ-output quality and predicate Q.

Lemma 45 *$\Pi_{\mathsf{VACS}}^{\ell,Q}$ is f-correct.*

Proof. This trivially follows from the fact that validity of BA implies that any instance BA_j outputs 1 only if an honest party inputs 1 to that instance. Since honest parties only input 1 to BA_j if the corresponding value from RBC_j satisfies Q, the statement follows.

Lemma 46 *$\Pi_{\mathsf{VACS}}^{\ell,Q}$ is f-consistent.*

Proof. Since each instance of BA satisfies consistency, all parties obtain the same set of indices $\mathsf{CoreSet}$. Now we argue that all parties output the same set of values. First, an instance BA_j outputs 1 only if there is an honest party that inputs 1 to BA_j (by validity, if all honest parties input 0, the output of BA is 0). Moreover, an honest party inputs 1 only if RBC_j terminated for that party. Due to consistency of RBC_j, all honest parties eventually output and terminate with the same value. Hence, all parties obtain the same set of values.

Lemma 47 *$\Pi_{\mathsf{VACS}}^{\ell,Q}$ has ℓ-output quality.*

Proof. We prove that there are at least ℓ instances of BA that output 1. Due to Lemma 32, there are at least $\frac{2+\epsilon}{3} \cdot \frac{3}{2+\epsilon}\ell = \ell$ honest parties in C at Step 1 of the protocol with overwhelming probability. All these parties give input to an instance of RBC. By validity of RBC, all these instances eventually terminate.

F Proof of Theorem 18

The claims regarding the communication complexity and setup follow from the fact that the MPC protocol runs κ reliable broadcast protocols (in expectation) in Step 3 on the output size $\mathcal{O}$, and two VACS protocols: the first one with ℓ-output quality and input size $\mathcal{I}$, and the second one with $(t+1)$-output quality and input size $\mathcal{R} = O(\kappa)$.

We sketch the simulator in each of the protocol steps:

- Setup: Generate the setup as in the real protocol, send the keys corresponding to dishonest parties to the adversary.
- Input: Set $c_i = \mathsf{Enc}_{ek}(0)$, for each honest party P_i. Compute a correct proof of plaintext knowledge π_i^{popk} for the ciphertext c_i. Then, emulate the messages of the VACS protocol with input $(c_i, \pi_i^{\mathsf{popk}})$. Let S be the output of the VACS.
- Random input wires $\mathcal{R} = O(\kappa)$: For each honest party P_i, sample uniformly at random $r_i \in \{0,1\}^R$, set $u_i = \mathsf{Enc}_{ek}(r_i)$ and compute a correct proof of plaintext knowledge τ_i^{popk}. Then, emulate the messages of the VACS protocol with input $(u_i, \tau_i^{\mathsf{popk}})$. Let T be the output of the VACS. Let v be the homomorphic addition of the ciphertexts in T.
- Computation: Let $\mathsf{Circ}(S)$ be the circuit g where for each party P_j that did not contribute a ciphertext into S (S defined at Step 1), a default value $\perp$ is hard-wired into the circuit, and ciphertext v is hard-wired as the input random wires in the circuit. Let c_j denote the input ciphertext obtained for party $P_j \in S$ in Step 1. The simulator locally computes $c^i := \mathsf{Eval}_{ek}(\mathsf{Circ}(S), c_{j_1}, \ldots, c_{j_{|S|}})$.
- Threshold decryption: emulate the messages of the reliable broadcast, where the decryption shares (and non-interactive proofs of correct decryption) from honest parties are set such that (any $t+1$ shares in) $(d_1, \ldots, d_{|C|})$ forms a secret sharing of the output value y.
- Termination: emulate the messages of the reliable consensus protocol with input y.
- *Emulation of VACS with input provider size s and output-quality ℓ.* Before the emulation of the protocol, the simulator emulates the messages of its setup. That is, the committee C and notification to each dishonest party about membership on the committee. Then, on behalf of each honest party P_i, emulate the party. Let v_i be the input with which the party is emulated (in the MPC protocol, it is $v_i = (c_i, \pi_i^{\mathsf{popk}})$), where c_i is an encryption of 0.
  - If $P_i \in C$: Emulate a reliable broadcast protocol RBC_i where P_i is the sender with input v_i. Emulate the messages of the other protocols RBC_j as the receiver.
  - On output v from RBC_j, if $Q(v) = 1$ and an input has not yet been provided to BA_j, then emulate the party P_i with input 1 to BA_j.
  - When ℓ of the protocols BA have output 1, emulate P_i with input 0 to each instance BA_j if it has not yet been provided input.
  - Once all instances of BA have been completed, let $\mathsf{CoreSet}$ be the indices of each BA that delivered 1. Wait for the output v_j for each RBC_j, $j \in \mathsf{CoreSet}$. Finally output $\cup_{j \in \mathsf{CoreSet}} \{v_j\}$.
- *Emulation of reliable consensus.* The simulator first simulates the setup fo reliable consensus, i.e., the public keys for the sender and for the committees C_1 and C_2. Then:
  - On behalf of each honest party P_i with input v_i, do as follows:
    * If $P_i \in C_1$: send (v_i, σ), where $\sigma = \mathsf{Sign}_{\mathsf{sk}_{1,i}}(v_i)$, and send $(\texttt{echo}, (v_i, \sigma))$ to all parties.
    * If $P_i \in C_2$: Upon receiving $(\texttt{echo}, (v_j, \sigma_j))$ on the same value v from $\kappa - t$ distinct parties with correct signatures, compute $\sigma = \mathsf{Sign}_{\mathsf{sk}_{2,i}}(v)$ and send $(\texttt{ready}, (v, \sigma_j))$ to all parties.

    - ∗ If $P_i \in C_2$: Upon receiving $(\texttt{ready}, (v, \sigma_j))$ on the same value v from $t + 1$ distinct parties, if no `ready` message was sent, compute $\sigma = \mathsf{Sign}_{\mathsf{sk}_{2,i}}(v)$ and send $(\texttt{ready}, (v, \sigma_j))$ to all parties.
    - ∗ Upon receiving $(\texttt{ready}, (v, \sigma_j))$ on the same value v from $\kappa - t$ distinct parties, output v and terminate.

– *Emulation of reliable broadcast.* Before the emulation of the protocol, the simulator emulates the messages of its setup, consisting of the public key setup for the sender $(\mathsf{sk}_s, \mathsf{pk}_s)$.
  - If the sender is honest at the start of the execution, let v denote the input with which the sender is emulated. Then, compute $\sigma = \mathsf{Sign}_{\mathsf{sk}_s}(v)$ and send (v, σ) to all parties.
  - On behalf of each honest party P_i, do as follows:
    - ∗ Upon receiving a pair (v, σ) such that $\mathsf{Vrfy}_{\mathsf{pk}_s}(v, \sigma) = 1$, emulate the messages of a reliable consensus protocol.

– *Emulation of BA.* Before the emulation of the protocol, the simulator emulates the messages of its setup. That is, for each invocation to the sub-protocol $\mathit{\Pi}_{\mathsf{GC}}$, the public keys for the committees C_1, C_2 and C_3. Moreover, for each coin-flip sub-protocol invocation, emulate the setup for the secret sharing scheme, which includes a committee and shares for random bits distributed among the members of the committee. Then, on behalf of each honest party P_i, emulate the party with the input chosen by the simulator v_i. We describe the emulation for an iteration k of the BA protocol.
  - Emulate the messages of the graded consensus protocol $\mathit{\Pi}_{\mathsf{GC}}$ on input v_i. Let (b, g) denote the output.
  - Emulate the messages of the k-th coin-flip protocol. For that, if the party is in C_k, send the share. Once the party collects enough shares, it reconstructs the output. Let Coin_k be the output.
  - If $g < 1$ then set $b := \mathsf{Coin}_k$.
  - Emulate $\mathit{\Pi}_{\mathsf{GC}}$ on input b, and let (b, g) denote the output.
  - If $g = 1$ emulate $\mathit{\Pi}_{\mathsf{rc}}$ with input b, and when it terminates with output b', output b' and terminate.

– *Corruption Requests.* Upon a corruption request of a party P_i, corrupt the party in the ideal world, learn its input x_i and deliver it with the corresponding values from all setups that have not been erased so far (according to the protocol description) to the adversary.

Security proof. We define a series of hybrid experiments, starting from the real-world execution and ending with the ideal-world execution, and argue that each is computationally indistinguishable from the next.

Hybrid 1. This corresponds to the real world execution. Here, the simulator knows the inputs and keys of all honest parties.

Hybrid 2. We modify the previous hybrid, so that whenever the simulator has to compute a non-interactive zero-knowledge proof on behalf of an honest party, it computes a simulated proof without using the witness.

Hybrid 3. Here the computation of the decryption shares is different. In this case, the simulator obtains the output y, computes the decryption shares of corrupted parties, and then adjusts the decryption shares of honest parties in C such that any subset of $t+1$ decryption shares $(d_1, \ldots, d_{|C|})$ form a secret sharing of the output value y. That is, here the simulator does not need to know the secret key share of honest parties to compute the decryption shares.

Hybrid 4. We modify the previous hybrid in the Input Stage. Here, the honest parties, instead of sending an encryption of the actual input, they send an encryption of 0.

Hybrid 5. This corresponds to the ideal world execution.

In order to prove that no environment can distinguish between the real world and the ideal world, we prove that no environment can distinguish between any two consecutive hybrids.

Claim 1. No efficient environment can distinguish between Hybrid 1 and Hybrid 2.

Proof. This follows from the security of the non-interactive zero-knowledge proof, the fact that honest parties always send valid proofs and the fact that honest parties erase the randomness to generate any proof. This last point ensures that the adversary cannot distinguish between both hybrids even if he adaptively corrupts a party from Hybrid 1 after it sent a proof.

Claim 2. No efficient environment can distinguish between Hybrid 2 and Hybrid 3.

Proof. This follows from properties of a secret sharing scheme and the security of the threshold encryption scheme. Given that the threshold is t, any number of corrupted decryption shares up to t does not reveal anything about the output y. By Lemma 32, there are no more than t corrupted parties in C. Moreover, one can find shares for honest parties such that $(d_1, \ldots, d_{|C|})$ is a sharing of y. Further note that, since each honest party erases its decryption key-share as soon as it sends the decryption share, an adaptive adversary cannot collect more than t decryption key-shares either.

Claim 3. No efficient environment can distinguish between Hybrid 3 and Hybrid 4.

Proof. This follows from the semantic security of the encryption scheme, and the fact that honest parties erase the randomness to compute the ciphertext, prior to sending the ciphertext. In particular, note that even an adaptive adversary cannot distinguish between both hybrids by corrupting a party that sent a ciphertext.

Claim 4. No efficient environment can distinguish between Hybrid 4 and Hybrid 5.

Proof. This follows, because the simulator in the ideal world and the simulator in Hybrid 5 emulate internally the joint behavior of the ideal assumed functionalities, exactly the same way.

We conclude that the real world and the ideal world are indistinguishable.