Proactive Secure Multiparty Computation with a Dishonest Majority

Karim Eldefrawy[1(✉)], Rafail Ostrovsky[2], Sunoo Park[3], and Moti Yung[4]

[1] Computer Science Laboratory, SRI International, Menlo Park, USA
karim.eldefrawy@sri.com
[2] Department of Computer Science and Department of Mathematics, UCLA, Los Angeles, USA
[3] Department of Computer Science, MIT, Cambridge, USA
[4] Department of Computer Science, Columbia University, New York City, USA

Abstract. Secure multiparty computation (MPC) protocols enable n distrusting parties to perform computations on their private inputs while guaranteeing confidentiality of inputs (and outputs, if desired) and correctness of the computation, as long as no adversary corrupts more than a threshold t of the n parties. Existing MPC protocols assure perfect security for $t \leq \lceil n/2 \rceil - 1$ active corruptions with termination (i.e., robustness), or up to $t = n - 1$ under cryptographic assumptions (with detection of misbehaving parties). However, when computations involve secrets that have to remain confidential for a long time such as cryptographic keys, or when dealing with strong and persistent adversaries, *such security guarantees are not enough*. In these situations, *all* parties may be corrupted over the lifetime of the secrets used in the computation, and the threshold t may be violated over time (even as portions of the network are being repaired or cleaned up). *Proactive MPC (PMPC)* addresses this stronger threat model: it guarantees correctness and input privacy in the presence of a *mobile adversary* that controls a changing set of parties over the course of a protocol, and could corrupt *all parties* over the lifetime of the computation, as long as no more than t are corrupted in each time window (called a *refresh period*). The threshold t in PMPC represents a tradeoff between the adversary's penetration rate and the cleaning speed of the defense tools (or rebooting of nodes from a clean image), rather than being an absolute bound on corruptions. Prior PMPC protocols only guarantee correctness and confidentiality in the presence of an *honest majority* of parties, an adversary that corrupts even a single additional party beyond the $n/2-1$ threshold, even if only passively and temporarily, can learn all the inputs and outputs; and if the corruption is *active* rather than passive, then the adversary can even compromise the correctness of the computation.

In this paper, we present *the first feasibility result for constructing a PMPC protocol secure against a dishonest majority.* To this end, we develop a new PMPC protocol, robust and secure against $t < n - 2$ passive corruptions when there are no active corruptions, and secure but non-robust (but with identifiable aborts) against $t < n/2 - 1$ active corruptions when there are no passive corruptions. Moreover, our protocol

D. Catalano and R. De Prisco (Eds.): SCN 2018, LNCS 11035, pp. 200–215, 2018.
https://doi.org/10.1007/978-3-319-98113-0_11

is secure (with identifiable aborts) against *mixed* adversaries controlling, both, passively and actively corrupted parties, provided that if there are k active corruptions, there are less than $n - k - 1$ total corruptions.

1 Introduction

Secure multiparty computation (MPC) protocols allow a set of distrusting parties, each holding private inputs, to jointly and distributedly compute a function of the inputs while guaranteeing correctness of its evaluation, and privacy of inputs (and outputs, if desired) for honest parties. The study of secure computation has been combining distributed computing paradigms and security methodologies. It was initiated by [Yao82] for two parties and [GMW87] for many parties, and both of these works relied on cryptographic primitives. The information-theoretic setting was introduced by [BGW88, CCD88] which, assuming private channels, constructed information-theoretically secure MPC protocols tolerating up to $n/3$ malicious parties. Assuming a broadcast channel, [RB89] constructs a protocol that can tolerate up to $n/2$ malicious parties. These thresholds, $n/3$ and $n/2$, are optimal in the information-theoretic setting, in their respective communication models. In the context of public key cryptography, schemes for enhancing distributed trust, e.g., threshold encryption and threshold signatures, are a special case of MPC, e.g., [FGMY97a, FGMY97b, Rab98, CGJ+99, FMY01, Bol03, JS05, JO08, ADN06]. Also, when the computation to be performed via MPC involves private keys, e.g., for threshold decryption or signature generation, it is of utmost importance for trustworthy operation to guarantee the highest possible level of corruption tolerance, since confidentiality of cryptographic keys should be ensured for a long time (e.g., years).

Constructing MPC protocols that guarantee security against stronger adversaries and at the same time satisfy low communication and computation complexity bounds has seen significant progress, e.g., [IKOS08, DIK+08, DIK10, BFO12, OY91, BELO14, BELO15]. While enforcing an honest majority bound on the adversary's corruption limit renders the problem (efficiently) solvable, it is often criticized, from a distributed systems point of view, as unrealistic for protocols that require long-term security of shared secrets used in the computation, or for very long computations (i.e., reactive operation, typical in systems maintenance), or may be targeted by nation-state adversaries (often called "Advanced Persistent Threats"). With advancements of cloud hosting of security services, and online exchanges for cryptocurrencies which require trustworthy services protected by their distributed nature, the above criticism makes sense. This concern is especially relevant when considering so-called "reactive" functionalities that never stop executing, e.g., continuously running control loops that perform threshold decryption or signature generation via a secret shared key. Such long-running reactive functionalities will become increasingly important for security in always-on cloud applications: example settings could include the use of MPC to compute digital signatures in online financial transactions between large institutions, or to generate securely co-signed cryptocurrency transactions via secret-shared (or otherwise distributed) keys [GGN16]. In both these cases, one should

expect persistent strong adversaries to continuously attack the parties involved in the MPC protocol, and given enough time vulnerabilities in underlying software (or even some hardware) will eventually be found, and the cryptographic keys may be compromised.

An approach to deal with an adversary's ability to eventually corrupt all parties is the *proactive security model* [OY91]. This model introduces the notion of a *mobile* adversary, motivated by the persistent corruption of participating parties in a distributed computation and the continuous race between parties' corruption and recovery. A mobile adversary is one that *can corrupt all parties in a distributed protocol* over the course of a protocol execution but with the following limitations: (1) only a constant fraction of parties can be corrupted during any round, and (2) parties periodically get rebooted to a clean initial state—in a fashion designed to mitigate the total adversarial corruption at any given time—guaranteeing that some fraction of honest parties will be maintained as long as the corruption rate is not more than the reboot rate[1]. The [OY91] model also assumes that an adversary does not have the ability to predict or reconstruct the randomness used by parties in any uncorrupted period of time, as demarcated by rebooting; in other words, a reboot entails erasing all previous state.

This paper's main goal is to answer the following basic question: *Is it feasible to construct a proactive MPC protocol for the* ***dishonest majority*** *setting?*

1.1 Contributions

We answer this question in the affirmative by developing *the first proactive secure multiparty computation (PMPC) protocol that is secure in the presence of a dishonest majority.* Our new protocol is, first, secure and robust against $t < n-2$ passive adversaries (parties which follow the protocol but leak what they know) when there are no active corruptions (arbitrarily misbehaving parties), and when parties are serially rebooted. Secondly, the same protocol preserves secrecy but is unfair (with identifiable aborts) against $t < n/2 - 1$ active adversaries when there are no additional passive corruptions. Thirdly, the protocol is also secure (but non-robust with identifiable aborts) against mixed adversaries that control a combination of passively and actively corrupted parties such that if there are k active corruptions there are less than $n - k - 1$ total corruptions[2]. We note that the number of parties we start from is $n - 1$ and not n because we assume that parties may be serially rebooted and need recovery from the rest of the $n - 1$ parties. The threshold t is $n - 3$ and not $n - 2$ because in the refresh protocol, the secret being shared in the randomizing polynomial is always 0, so the free coefficient in those polynomials is always an additional point that the adversary knows, hence we can tolerate one less corruption than in the non-proactive gradual secret sharing case.

[1] We model rebooting to a clean initial state to include required global information, e.g., circuit representation of the function to be computed, identities of parties, access to secure point-to-point and broadcast channels.

[2] The threshold in this case is actually the minimum of $n - 3$ and $n - k - 1$.

Our design and analysis require new ideas, since the security guarantees of all existing PMPC protocols *do not apply* in the case of a dishonest passive majority, or in the case of mixed adversaries that may form a majority as described above. Our PMPC protocol can be based on any one-way function and oblivious transfer (the same assumptions as the classic [GMW87] protocol, and formally requires only oblivious transfer which implies the existence of one-way functions). The secret sharing scheme underlying our PMPC protocol is an adaptation of [DEL+16], which recently constructed the first stand-alone proactive secret sharing scheme secure against a dishonest majority. The [DEL+16] scheme makes use of discrete-logarithm-based verification of secret shares (similar to [Fel87]); for our PMPC protocol (being a portion of a more general protocol), we replace this component with another technique (described below as "mini MPC") to overcome problematic proactive simulation issues in the security proof. Computing on secret-shared data (with security against mobile dishonest-majority adversaries) is a topic unaddressed by prior work. Our addition and multiplication sub-protocols are the building blocks that enable the parties to jointly compute a secret sharing of the desired output value. Addition of two secret-shared values can be performed by local addition of shares (as in many common secret sharing schemes), but multiplication requires more work. Our multiplication sub-protocol makes use of the [GMW87] protocol for standard MPC to perform a "mini MPC" on the proactive secret shares held by the parties, in order to obtain a proactive secret sharing of the multiplication of two secrets. (More generally, the multiplication sub-protocol can be instantiated based on *any* standard MPC protocol Φ secure against a dishonest majority, and inherits the efficiency properties from Φ.)

To build in security against mobile adversaries, we intersperse the execution of the addition and multiplication sub-protocols with a *refresh* sub-protocol that "refreshes" the shares held by all parties: informally, each time shares are refreshed, any knowledge of shares from previous "pre-refresh" sharings becomes useless to the adversary. This effectively prevents the adversary from learning sensitive information by putting together shares obtained from corruptions that occur far apart in time. Whenever a party is de-corrupted (rebooted), its memory contents are erased, so it needs to "recover" the necessary share information, this is achieved using our *recovery* sub-protocol which is triggered dynamically each time a memory loss occurs. The number of parties that can simultaneously lose memory is a parameter of our protocol, which trades off with the number of corruptions allowed per phase. This sensitive trade-off is inherent, if $n - \tau$ parties can *restore* the shares of τ parties who lost memory, then they could also collude to *learn* the shares of those τ parties.

As an additional contribution we provide the first (formal) *definition* of secure PMPC in the presence of a dishonest majority consisting of passively and actively corrupted parties in the full version [EOPY]. Prior security definitions for PMPC only addressed the honest majority setting, so they did not have to address potential failures of robustness and fairness. Moreover, no existing definitions considered PMPC security with *mixed* adversaries. Our ideal functionality for the dishonest majority setting models robustness and fairness as a fine-grained

function of the passive and active corruptions that *actually occur* during a protocol execution (rather than a coarser-grained guarantee depending on adherence to a corruption threshold that is fixed as a protocol parameter), by adapting for the proactive setting the *multi-thresholds* paradigm that was introduced by [HLM13] in the context of standard (not proactive) MPC.

1.2 Related Work

To the best of our knowledge there are currently only two generic PMPC protocols, [OY91] (requires $O(Cn^3)$ communication, where C is the size of the circuit to be computed via MPC) and [BELO14] (requiring $O(C\log^2(C)\text{polylog}(n) + D\text{poly}(n)\log^2(C))$, where C is the size of the circuit to be computed via MPC and D its depth). These PMPC protocols are inherently designed for an honest majority and it seems difficult to redesign them for a dishonest majority; the reason is that the underlying secret sharing scheme stores secrets as points on polynomials of degree less than $n/2$, so the only adversary structure that can be described is one in terms of a fraction of the degree of the polynomial and once the adversary compromises enough parties (even if only passively), it can reconstruct the polynomial and recover the secret.

1.3 Outline

The rest of the paper is organized as follows. Section 2 outlines the terminology of proactively secure computation, communication and adversary models; corresponding formal definitions are in Appendix A in the full version [EOPY]. Section 3 presents details of our PMPC protocol. The security proofs are provided in Appendix B in the full version [EOPY].

2 Model and Definitions

We consider n parties (p_i where $i \in [n]$) connected by a synchronous network and an authenticated broadcast channel. Protocol communication proceeds in discrete rounds which are grouped into consecutive blocks called *stages*. We consider a mobile adversary with polynomially bounded computing power, which "moves around" and chooses a (new) set of parties to corrupt per stage, subject to a maximum threshold of corruptions for any stage. Note that parties once corrupted do not necessarily remain so for the remainder of the protocol, which means that over the course of protocol execution, *the adversary can corrupt all the parties, although not all at the same time.*

2.1 Phases and Stages of a Proactive Protocol

We adopt terminology from previous formalizations of proactive protocols [ADN06, BELO14].

Phases. The rounds of a proactive protocol are grouped into *phases* $\varphi_1, \varphi_2, \ldots$. A phase φ consists of a sequence of consecutive rounds, and every round belongs to exactly one phase. There are two types of phases, *refresh* phases and *operation* phases. The protocol phases alternate between refresh and operation phases; the first and last phase of the protocol are both operation phases. Each refresh phase is furthermore subdivided into a *closing period* consisting of the first k rounds of the phase, followed by an *opening period* consisting of the final $\ell - k$ rounds of the phase, where ℓ is the total number of rounds in the phase.

In non-reactive MPC, the number of operation phases can be thought to correspond to the depth of the circuit to be computed. Intuitively, each operation phase serves to compute a layer of the circuit, and each refresh phase serves to re-randomize the data held by parties such that combining the data of corrupt parties across different phases will not be helpful to an adversary.

Stages. A *stage* σ of the protocol consists of an *opening period* of a refresh phase, followed by the subsequent operation phase, followed by the *closing period* of the subsequent refresh phase. Thus, a stage spans (but does not cover) three consecutive phases, and the number of stages in a protocol is equal to its number of operation phases. In the case of the first and last stages of a protocol, there is an exception to the alternating "refresh-operation-refresh" format, the first stage starts with the first operation phase, and the last stage ends with the last operation phase.

Corruptions. If a party p_i is corrupted by the adversary ($\mathcal{A}$) during an operation phase of a stage σ_j, then $\mathcal{A}$ learns the view of p_i starting from its state at the beginning of stage σ_j. If the corruption is made during a refresh phase between consecutive stages σ_j and σ_{j+1}, then $\mathcal{A}$ learns p_i's view starting from the beginning of stage σ_j. Moreover, in the case of a corruption during a refresh phase, p_i is considered to be corrupt in both stages σ_j and σ_{j+1}. Finally, a party p_i that is corrupt during the closing period of a refresh phase in stage σ_j may become *decorrupted*. In this case, p_i is considered to be no longer corrupt in stage σ_{j+1} (unless $\mathcal{A}$ corrupts him again before the end of the next closing period). A decorrupted party immediately rejoins the protocol as an honest party, if it was passively corrupted, then it rejoins with the correct state according to the protocol up to this point; or if it was actively corrupted, then it is restored to a clean default state (which may be a function of the current round). Note that in restoring a party to the default state, its randomness tapes are overwritten with fresh randomness: this is important since otherwise, any once-corrupted party would be deterministic to the adversary. In terms of modeling, parties to be decorrupted are chosen arbitrarily from the corrupt set by the environment.

Erasing State. In our model, parties erase their internal state (i.e., the content of their tapes) between phases. The capability of erasing state is necessary in the proactive model, if an adversary could learn all previous states of a party upon corruption, then achieving security would be impossible, since over the course of a protocol execution a mobile adversary would eventually learn the state of *all* parties in certain rounds.

2.2 Mixed Corruption Model

We consider *mixed* adversaries [HLM13] which can perform two distinct types of corruptions. The adversary can *passively* corrupt a set of parties (P) and only read their internal state; the adversary may also *actively* corrupt some of these parties (A) and make them deviate arbitrarily from the protocol. We assume that $\mathrm{A} \subseteq \mathrm{P}$. In traditional MPC, a common notation is to denote the number of parties by n, and the maximum threshold of corrupt parties by t. For mixed adversaries, there are distinct thresholds for active and passive corruptions. We write t_a and t_p to denote the thresholds of active and passive corruptions, respectively, i.e., $|\mathrm{A}| \leq t_a$ and $|\mathrm{P}| \leq t_p$. Note that since we have defined each active corruption to be also a passive corruption, each active corruption counts towards both t_a and t_p. Following the notation of [HLM13,DEL+16], in order to model security guarantees against incomparable maximal adversaries, we consider *multi-thresholds* $T = \left\{(t_a^1, t_p^1), \ldots, (t_a^k, t_p^k)\right\}$ which are sets of pairs of thresholds (t_a, t_p). Security properties are guaranteed if $(\mathrm{A}, \mathrm{P}) \leq (t_a, t_p)$ for some $(t_a, t_p) \in T$, where $(\mathrm{A}, \mathrm{P}) \leq (t_a, t_p)$ is a shorthand for $|\mathrm{A}| \leq t_a$ and $|\mathrm{P}| \leq t_p$. If this condition is satisfied, we write that $(\mathrm{A}, \mathrm{P}) \leq T$.

We define our MPC protocols in terms of four security properties: correctness, secrecy, robustness, and fairness.[3] The security properties which are guaranteed in any given protocol execution is a function of the number of actually corrupted parties. Accordingly, we consider four multi-thresholds T_c, T_s, T_r, T_f. Correctness (with agreement on abort) is guaranteed if $(\mathrm{A}, \mathrm{P}) \leq T_c$, secrecy is guaranteed if $(\mathrm{A}, \mathrm{P}) \leq T_s$, robustness is guaranteed if $(\mathrm{A}, \mathrm{P}) \leq T_r$, and fairness is guaranteed if $(\mathrm{A}, \mathrm{P}) \leq T_f$. Note that $T_r \leq T_c$ and $T_f \leq T_s \leq T_c$, since secrecy and robustness are not well-defined without correctness, and secrecy is a precondition of fairness.[4]

2.3 New PMPC and Security Definitions

Formal definitions for a proactive MPC protocol and the corresponding ideal functionality, and security for mixed mobile adversaries and dishonest majorities can be found in Appendix A in the full version [EOPY] due to space constraints. These definitions are new to this work; they do not exist in prior proactive MPC literature since the dishonest majority setting is unaddressed. One notable

[3] These terms are standard in the MPC literature. *Correctness* means that all parties that output a value must output the correct output value with respect to the set of all parties' inputs and the function being computed by the MPC. *Secrecy* means that the adversary cannot learn anything more about honest inputs and outputs than can already be inferred from the corrupt parties' inputs and outputs (more formally, secrecy requires that the adversary's view during protocol execution can be simulated given only the corrupt parties' input and output values). *Robustness* means that the adversary must not be able to prevent honest parties from learning their outputs. Finally, *fairness* requires that either all honest parties learn their own output values, or no party learns its own output value.

[4] We write $T \leq T'$ if $\forall (t_a, t_p) \in T,\ \exists (t'_a, t'_p) \in T'$ such that $t_a \leq t'_a$ and $t_p \leq t'_p$.

difference of the proactive dishonest majority definition we develop compared to the dishonest majority model for standard MPC is that in the standard model, it is acceptable to exclude parties found to be corrupt and simply restart the protocol with the remaining parties, whereas in the proactive setting this could result in the exclusion of *all* parties even though the adversary cannot actually corrupt all parties simultaneously. Thus, exclusion of misbehaving parties in our proactive model is only temporary, and the protocol is guaranteed to make progress in any phase when the adversary does not cause a majority of parties to deviate from the protocol (otherwise, the phase is restarted). An adversary could cause multiple restarts of a phase and delay protocol execution—which seems unavoidable in a dishonest majority model with a mobile adversary—but cannot cause a phase to have an incorrect output. Due to the definitions' length and notational complexity, we have opted for a less formal protocol description in the limited space in the body.

3 Construction of a PMPC Protocol for Dishonest Majorities

3.1 Intuition and Overview of Operation

Our PMPC protocol consists of six sub-protocols. `GradualShare` allows a dealer to share a secret s among n parties. `Reconstruct` allows parties to reconstruct the underlying secret s based on shares that they hold. `Refresh` is executed between two consecutive phases, w and $w+1$, and generates new shares for phase $w+1$ that encode the same secret as the shares in phase w. `Recover` allows parties that lost their shares to obtain new shares encoding the same secret s, with the help of other honest parties. `Add` allows parties holding shares of two secrets s and s' to obtain shares that encode the sum $s+s'$. `Mult` allows parties holding shares of two secrets s and s' to obtain shares that encode the product $s \times s'$.

The overall operation of the PMPC protocol is as follows. First, each party uses `GradualShare` to distribute its private input among the n parties (including itself). The circuit to be computed via PMPC is public, and consists of multiple layers each comprised of a set of `Add` or `Mult` gates which are executed via the corresponding sub-protocols (layer by layer). Between circuit layers, the shares of all parties are refreshed via `Refresh`. Decorrupted parties obtain new shares encoding the same shared secrets corresponding to the current state of the MPC computation, i.e., the output of the current circuit layer and any shard values that will be needed in future layers, by triggering the `Recover` sub-protocol as soon as they find themselves rebooted. When the (secret-shared) output of the final layer of the circuit is computed, parties use `Reconstruct` to reconstruct the final output.

In order to tolerate a dishonest majority, it is not enough to directly store the inputs of the parties (the secrets to be computed on, and which will at the end be transformed into the outputs) in the free term, or as other points on a

polynomial. What is needed is to encode the secrets, and compute using them, in a different form resistant to a dishonest majority of say up to $n-1$ parties. At a high level, this can be achieved by first additively sharing the secret into $d = n-1$ random additive summands (this provides security against $t = n-3$ passive corruptions), then sharing each summand using polynomial-based secret sharing for a range of different reconstruction thresholds: this is the key insight of the "gradual secret sharing" scheme of [DEL+16].

We develop protocols to add and multiply shares to perform computation on the secret shares. Addition can be performed locally, but to multiply we utilize a standard MPC protocol for a dishonest majority. A simple version of our protocol yields security against passive corruptions; to furthermore achieve active security, we leverage constant round non-malleable homomorphic commitments and zero-knowledge proofs based on one-way functions and oblivious-transfer.

The protocol description thus far makes the following two simplifying assumptions: (1) the function f to be computed is deterministic, and (2) all output wire values are learned by all parties. The next two paragraphs discuss how to generalize our protocols, eliminating these assumptions.

We address randomized functions using a standard technique, each party p_i initially chooses a random value ζ_i. We treat (x_i, ζ_i) as the input of party p_i (instead of just x_i as above), and compute the deterministic function f' defined by $f'((x_1, \zeta_1), \ldots, (x_n, \zeta_n)) = f(x_1, \ldots, x_n; \zeta_1 + \cdots + \zeta_n)$. As this is a standard transformation, we omit further details, and for simplicity of exposition, the rest of the paper deals only with deterministic functions.

We now describe an adaptation for the case when each party p_i is to receive its own private output y_i, as follows. This is a slight variation of the standard technique of "masking" output values using a random mask known only to the intended recipient—but we highlight that the standard technique requires a tweak for the proactive setting.[5] Before the reconstruction step, the parties possess a gradual secret sharing of the output values $(y_1, \ldots, y_n)$. At this point, each party chooses a secret random value ρ_i (called a *mask*) and shares it among the n parties using `GradualShare`. Then, the `Add` sub-protocol is run to obtain a gradual secret sharing of $(y_1 + \rho_1, \ldots, y_n + \rho_n)$ instead of $(y_1, \ldots, y_n)$. Next, the `Reconstruct` sub-protocol is run so that every party learns $(y_1 + \rho_1, \ldots, y_n + \rho_n)$. Finally, each party p_i performs an additional local computation at the end of the protocol, subtracting ρ_i from the value on his output wire to obtain his final output y_i.

3.2 Real-World Protocol Operation

We now give the formal definition of protocol operation based on the sub-protocols. Definition 1 is the formalization of the description given in prose in Sect. 3.1.

[5] The standard trick is to consider the masks ρ_i to be part of the parties' inputs. In the proactive setting, it is important that the masks be chosen later on, as we shall see in the security proof.

The description of how each sub-protocol works will be given in Sect. 3.3. Within Definition 1 below, the subprotocols are invoked in black-box manner.

Definition 1 (PMPC Protocol Operation). *Given an arithmetic circuit C (of depth d_C) that is to be computed by an MPC protocol on inputs $x_1, \ldots, x_n$, the proactive MPC protocol is defined as follows. For simplicity, we assume that refresh phases occur between layers of the circuit, and let $\mathfrak{R} \subseteq [d_C]$ be the set of circuit layers after which a refresh phase is to be triggered.*[6]

1. *Each party p_i acts as the dealer in* `GradualShare` *to share its own input x_i among all n parties. (Note that at the conclusion of this step, the parties hold secret sharings of all the values on the input wires of C, i.e., all the inputs to gates at layer 1 of C.)*
2. *Run the* `Refresh` *sub-protocol. The duration of a single* `Refresh` *sub-protocol execution is considered to be a refresh phase.*
3. *For each layer of the circuit, $\ell = 1, \ldots, d_C$:*
   - *For each addition or multiplication gate μ in layer ℓ:*[7]
     Compute a sharing of the value on the output wire of μ by using the `Add` *or* `Mult` *sub-protocol respectively. The parties' inputs to the* `Add` *or* `Mult` *protocol will be the sharings of the values on the input wires of μ, which the parties already possess (the input sharings are computed by step 1 for $\ell = 1$, and subsequently, the input sharings for layer $\ell + 1$ are computed during step ℓ).*
   - *If $\ell \in \mathfrak{R}$, run the* `Refresh` *sub-protocol.*
4. *At the conclusion of step 3, the parties possess a gradual sharing of the value $(y_1, \ldots, y_n)$ on the output wire(s) of the circuit C, where each y_i is the output intended for party p_i. The period from this step until the end of the protocol is a single operation phase. Each party now samples a random value ρ_i and acts as the dealer in* `GradualShare` *to share ρ_i among all n parties. Then, the* `Add` *sub-protocol is run to obtain a gradual sharing of the value $(z_1, \ldots, z_n)$ where $z_i = y_i + \rho_i$.*
5. *The* `Reconstruct` *sub-protocol is then run to reconstruct the shared value $(z_1, \ldots, z_n)$.*
6. *Each party p_i obtains its output y_i by subtraction: $y_i = z_i - \rho_i$.*

Moreover, the adversary may decorrupt *a party at any point, during operation or refresh phases, upon which the decorrupted party is restored to a default state which we shall call $\perp$.*

- *Whenever a party finds itself with internal state $\perp$, it broadcasts a message* Help!.

[6] In general, more complex refresh patterns are possible, e.g., at the level of gates rather than circuit layers.

[7] If the `Add` and `Mult` sub-protocols are secure under parallel composition, the iterations of this for-loop can be executed in parallel for all gates in layer ℓ.

– *Upon receiving message* Help! *from a party* p_i*, all parties immediately execute the* `Recover` *sub-protocol so that* p_i *ends up with the secret shares: of all values on circuit wires that will be used for later computation, or in steps 4–6, of the masks* $\rho_1, \ldots, \rho_n$ *and the shared output* $(z_1, \ldots, z_n)$*. In addition, from step 4 onwards,* p_i *is assisted to recover his own mask* ρ_i*, by the other parties sending to* p_i *their shares thereof. Then, the interrupted* operation phase *or* refresh phase *is resumed, starting with the next round after the last completed operation-phase or refresh-phase round.*

3.3 Sub-protocol Specifications

In the following, field operations occur over a finite field $\mathbb{F}$ (of prime characteristic). The sub-protocols make use of a polynomial-based secret sharing schemes, e.g., [Sha79], and are implicitly parametrized by $(\mathbb{F}, n, d)$ where n is the number of parties and $n-d-1$ is the number of parties that can simultaneously undergo a reboot (thus losing their shares, and requiring recovery). The multiplication sub-protocol is additionally parametrized by Φ (which, in turn, is parametrized by a security parameter κ), which can be any general MPC protocol secure against up to $n-1$ active corruptions (such as [GMW87]). For simplicity, secret values are assumed to be field elements; multi-element secrets can be handled by running the sub-protocols on each element separately.

The proactive MPC protocol resulting from instantiating Definition 1 with the sub-protocols defined in this subsection is denoted by $\texttt{ProactiveMPC}_{\mathbb{F},n,d,\Phi}$.

`GradualShare` is used by parties to share their inputs, i.e., each party acts as a dealer when sharing its own inputs. Parties holding sharings (from `GradualShare`) of secrets s may use subprotocol `Reconstruct` to reconstruct s, or use subprotocol `Refresh` to refresh (re-randomize) their shares. Parties holding sharings of secrets s, s' can compute a sharing of $s + s'$ using `Add`, or a sharing of $s \times s'$ using `Mult`.

Subprotocol 1 (`GradualShare`). *We denote by* p_D *the* dealer *who starts in possession of the secret value* s *to be shared. At the conclusion of this protocol, each party (including the dealer) will possess a share of the secret* s*.*

1. p_D *chooses* d *random summands* $s_1, \ldots, s_d$ *which add up to* s*,* $\Sigma_{\delta=1}^{d} s_\delta = s$*.*
2. *For* $\delta = 1, \ldots, d$*, the dealer* p_D *does the following:*
   (a) p_D *samples a random degree-*δ *polynomial* f_δ *over finite field* $\mathbb{F}$*, subject to* $f_\delta(0) = s_\delta$*.* p_D *stores the evaluations* $f_\delta(1), \ldots, f_\delta(n)$ *and deletes* f_δ *from memory.*
   (b) *For* $i \in [n]$*, the dealer* p_D *sends* $sh_{\delta,i} = f_\delta(i)$ *to* p_i*, then deletes* $f_\delta(i)$ *from memory.*
3. *Each party* p_i *stores its* d *shares* $\boldsymbol{sh}_i = (sh_{1,i}, \ldots, sh_{d,i})$*.*

Subprotocol 2 (`Reconstruct`). *After a sharing of a secret* s *using* `GradualShare`*, the* n *parties can reconstruct* s *as follows.*

1. *For* $\delta = d, \ldots, 1$*:*

(a) *Each party p_i broadcasts its share $sh_{\delta,i}$.*
(b) *Each party locally interpolates to determine the polynomial f_δ, then computes $s_\delta = f_\delta(0)$.*

2. *Each party outputs the secret s computed as $s = s_1 + s_2 + \cdots + s_d$.*

Subprotocol 3 (`Refresh`). *Each party $p_i \in \{p_i | i \in [n]\}$ begins this protocol in possession of shares $\boldsymbol{sh}_i = (sh_{1,i}, \ldots, sh_{d,i})$ and ends this protocol in possession of new "refreshed" shares $\boldsymbol{sh}'_i = (sh'_{1,i}, \ldots, sh'_{d,i})$.*

1. *Each party p_i generates an additive sharing of 0 (i.e., d randomization summands which add up to 0). Let the additive shares of p_i be denoted by $r_{\delta,i}$: note that $\Sigma_{\delta=1}^{d} r_{\delta,i} = 0$.*
2. *For $\delta = 1, \ldots, d$ do:*
   (a) *For $i = 1, \ldots, n$: Party p_i shares $r_{\delta,i}$ by running* `GradualShare` *and acting as the dealer.*
   (b) *Each party p_i adds up the shares it received: $sh''_i = \sum_{j=1}^{n} sh^j_{\delta,i}$ and sets $sh'_{\delta,i} = sh_{\delta,i} + sh''_i$.*
3. *Each honest party p_i deletes the old shares $\boldsymbol{sh}_i$ and stores instead: $\boldsymbol{sh}'_i = (sh'_{1,i}, \ldots, sh'_{d,i})$.*

The following sub-protocol is used by parties to recover shares for a rebooted party.

Subprotocol 4 (`Recover`). *Let parties $\{p_r\}_{r \in R}$ be the ones that need recovery, where $R \subset [n]$. We refer to the other parties, $\{p_i\}_{i \notin R}$, as "non-recovering parties." Below, we describe the procedure to recover the shares of a single party p_r. To recover the shares of all parties, the below procedure should be run $\forall r \in R$.*

1. *For $\delta = 1, \ldots, d$ do:*
   (a) *Each non-recovering party p_i chooses a random degree-δ polynomial $g_{\delta,i}$ subject to the constraint that $g_{\delta,i}(r) = 0$.*
   (b) *Each non-recovering party p_i shares its polynomial with the other $n - |R| - 1$ non-recovering parties as follows: p_i computes and sends to each receiving party p_j the value $sh^i_{\delta,j} = g_{\delta,i}(j)$.*
   (c) *Each non-recovering party p_j adds all the shares it received from the other $n - |R| - 1$ parties for the recovery polynomials $g_{\delta,i}$ to its share of f_δ, i.e., $z^j_\delta = f_\delta(j) + \Sigma_{i=1}^{n} sh^i_{\delta,j} = f_\delta(j) + \Sigma_{i=1}^{n} g_{\delta,i}(j)$.*
   (d) *Each non-recovering party p_j sends z^j_δ to p_r. Using this information, p_r interpolates the recovery polynomial $g_\delta = f_\delta + \Sigma_{i=1}^{n} g_{\delta,i}$ and computes $sh_{\delta,r} = g_\delta(r) = f_\delta(r)$.*

Subprotocol 5 (`Add`). *Each party $p_i \in \{p_i | i \in [n]\}$ begins this protocol in possession of shares $\boldsymbol{sh}_i = (sh_{1,i}, \ldots, sh_{d,i})$ corresponding to a secret s and $\boldsymbol{sh}'_i = (sh'_{1,i}, \ldots, sh'_{d,i})$ corresponding to a secret s', and ends this protocol in possession of shares $\boldsymbol{sh}^+_i = (sh^+_{1,i}, \ldots, sh^+_{d,i})$ corresponding to the secret $s + s'$.*

1. *For each $\delta \in \{1, \ldots, d\}$ and each $i \in [n]$, party p_i sets $sh^+_{\delta,i} = sh_{\delta,i} + sh'_{\delta,i}$.*

Subprotocol 6 (`Mult`). *Each party* $p_i \in \{p_i | i \in [n]\}$ *begins this protocol in possession of shares* $\boldsymbol{sh}_i = (sh_{1,i}, \ldots, sh_{d,i})$ *corresponding to a secret* s *and* $\boldsymbol{sh'_i} = (sh'_{1,i}, \ldots, sh'_{d,i})$ *corresponding to a secret* s'*, and ends this protocol in possession of shares* $\boldsymbol{sh}_i^{\times} = (sh^{\times}_{1,i}, \ldots, sh^{\times}_{d,i})$ *corresponding to the secret* $s \times s'$*.*

1. *Each party* p_i *adds up its local shares of* s *and* s' *respectively:* $\theta_i = \sum_{\delta \in [d]} sh_{\delta,i}$ *and* $\theta'_i = \sum_{\delta \in [d]} sh'_{\delta,i}$*. By construction of the gradual secret sharing scheme, these sums can be expressed as* $\theta_i = \widehat{f}(i)$ *and* $\theta'_i = \widehat{f'}(i)$ *for some degree-*d *polynomials* $\widehat{f}, \widehat{f'}$ *such that* $\widehat{f}(0) = s$ *and* $\widehat{f'}(0) = s'$*.*
2. *Run the MPC protocol of [GMW87] as follows:*
   - *The input of party* p_i *to the MPC is* (θ_i, θ'_i)*.*
   - *The function to be computed by the MPC on the collective input* $\Big((\theta_1, \theta'_1), \ldots, (\theta_n, \theta'_n)\Big)$ *is:*
     (a) *Interpolate* $(\theta_i)_{i \in [n]}$ *and* $(\theta'_i)_{i \in [n]}$ *to recover the secrets* s *and* s' *as the free terms of the respective polynomials* $\widehat{f}$ *and* $\widehat{f'}$*.*
     (b) *Compute the product* $s^{\times} = s \times s'$*.*
     (c) *Compute shares* $(sh^{\times}_{\delta,i})_{\delta \in [d], i \in [n]}$ *as a dealer would when sharing secret* $s^{\times}$ *using* `GradualShare`*.*
     (d) *For each* $i \in [n]$*, output* $(sh^{\times}_{\delta,i})_{\delta \in [d]}$ *to party* p_i*.*

3.4 Security Proofs

Security proofs of the full protocol with respect to the formal definitions in Appendix A are given in Appendix B in the full version [EOPY] due to space constraints.

4 Conclusion and Open Issues

This paper presents the *first proactive secure multiparty computation (PMPC) protocol for a dishonest majority.* Our PMPC protocol is robust and secure against $t < n - 2$ passive only corruptions, and secure but non-robust (but with identifiable aborts) against $t < n/2 - 1$ active corruptions when there are no additional passive corruptions. The protocol is also secure, and non-robust but with identifiable aborts, against mixed adversaries that control a combination of passively and actively corrupted parties such that with k active corruptions there are less than $n - k - 1$ total corruptions.

In this paper we prove the feasibility of constructing PMPC protocols secure against dishonest majorities. Optimizing computation and communication in such protocols (and making them practical) is not the goal of this paper and is an interesting open problem. Specifically, we highlight the following issues of interest which remain open:

- There are currently no practical proactively secure protocols for dishonest majorities for specific classes of computations of interest such as threshold

decryption and signature generation; all existing practical proactively secure threshold encryption and signature schemes such as [FGMY97a, FGMY97b, Rab98, FMY01, Bol03, JS05, JO08, ADN06] require an honest majority.
- There are currently no PMPC protocols (or even only proactive secret sharing schemes) for asynchronous networks and secure against dishonest majorities. Our PMPC protocol assumes a synchronous network.
- It is unclear what the lowest bound for communication required for a PMPC protocol secure against a dishonest majority is. We achieve $O(n^4)$ communication for the refresh and recover sub-protocols which are typically the bottleneck; it remains open if this can be further reduced. PMPC protocols [BELO14, BELO15] for an honest majority have constant (amortized) communication overhead; it is unlikely that this can be matched in the dishonest majority case, but it may be possible to achieve $O(n^3)$ or $O(n^2)$.

Acknowledgements. We thank Antonin Leroux for pointing out typos and issues in the statement of Theorem 2 in the appendix. We also thank the SCN 2018 reviewers for their constructive feedback which helped us improve the readability of the paper. The second author's research is supported in part by NSF grant 1619348, DARPA SafeWare subcontract to Galois Inc., DARPA SPAWAR contract N66001-15-1C-4065, US-Israel BSF grant 2012366, OKAWA Foundation Research Award, IBM Faculty Research Award, Xerox Faculty Research Award, B. John Garrick Foundation Award, Teradata Research Award, and Lockheed-Martin Corporation Research Award. The views expressed are those of the authors and do not reflect position of the Department of Defense or the U.S. Government.

References

[ADN06] Almansa, J.F., Damgård, I., Nielsen, J.B.: Simplified threshold RSA with adaptive and proactive security. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 593–611. Springer, Heidelberg (2006). https://doi.org/10.1007/11761679_35

[BELO14] Baron, J., Eldefrawy, K., Lampkins, J., Ostrovsky, R.: How to withstand mobile virus attacks, revisited. In: Proceedings of the 2014 ACM Symposium on Principles of Distributed Computing, PODC 2014, pp. 293–302. ACM, New York (2014)

[BELO15] Baron, J., Eldefrawy, K., Lampkins, J., Ostrovsky, R.: Communication-optimal proactive secret sharing for dynamic groups. In: Proceedings of the 2015 International Conference on Applied Cryptography and Network Security ACNS 2015 (2015)

[BFO12] Ben-Sasson, E., Fehr, S., Ostrovsky, R.: Near-linear unconditionally-secure multiparty computation with a dishonest minority. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 663–680. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32009-5_39

[BGW88] Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In: STOC, pp. 1–10 (1988)

[Bol03] Boldyreva, A.: Threshold signatures, multisignatures and blind signatures based on the Gap-Diffie-Hellman-Group signature scheme. In: Desmedt, Y.G. (ed.) PKC 2003. LNCS, vol. 2567, pp. 31–46. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36288-6_3

[CCD88] Chaum, D., Crépeau, C., Damgard, I.: Multiparty unconditionally secure protocols. In: Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing, STOC 1988, pp. 11–19. ACM, New York (1988)

[CGJ+99] Canetti, R., Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Adaptive security for threshold cryptosystems. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 98–116. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48405-1_7

[DEL+16] Dolev, S., ElDefrawy, K., Lampkins, J., Ostrovsky, R., Yung, M.: Proactive secret sharing with a dishonest majority. In: Zikas, V., De Prisco, R. (eds.) SCN 2016. LNCS, vol. 9841, pp. 529–548. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-44618-9_28

[DIK+08] Damgård, I., Ishai, Y., Krøigaard, M., Nielsen, J.B., Smith, A.: Scalable multiparty computation with nearly optimal work and resilience. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 241–261. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85174-5_14

[DIK10] Damgård, I., Ishai, Y., Krøigaard, M.: Perfectly secure multiparty computation and the computational overhead of cryptography. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 445–465. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13190-5_23

[EOPY] Eldefrawy, K., Ostrovsky, R., Park, S., Yung, M.: (Full Version) Proactive Secure Multiparty Computation with a Dishonest Majority. https://www.researchgate.net/publication/325722786

[Fel87] Feldman, P.: A practical scheme for non-interactive verifiable secret sharing. In 28th Annual Symposium on Foundations of Computer Science, Los Angeles, California, USA, 27–29 October 1987, pp. 427–437. IEEE Computer Society (1987)

[FGMY97a] Frankel, Y., Gemmell, P., MacKenzie, P.D., Yung, M.: Optimal resilience proactive public-key cryptosystems. In: 38th Annual Symposium on Foundations of Computer Science, FOCS 1997, Miami Beach, Florida, USA, October 19–22, 1997, pp. 384–393. IEEE Computer Society (1997)

[FGMY97b] Frankel, Y., Gemmell, P., MacKenzie, P.D., Yung, M.: Proactive RSA. In: Kaliski, B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 440–454. Springer, Heidelberg (1997). https://doi.org/10.1007/BFb0052254

[FMY01] Frankel, Y., MacKenzie, P.D., Yung, M.: Adaptive security for the additive-sharing based proactive RSA. In: Kim, K. (ed.) PKC 2001. LNCS, vol. 1992, pp. 240–263. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44586-2_18

[GGN16] Gennaro, R., Goldfeder, S., Narayanan, A.: Threshold-optimal DSA/ECDSA signatures and an application to Bitcoin wallet security. In: Manulis, M., Sadeghi, A.-R., Schneider, S. (eds.) ACNS 2016. LNCS, vol. 9696, pp. 156–174. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-39555-5_9

[GMW87] Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or a completeness theorem for protocols with honest majority. In: Aho, A.V. (ed), STOC, pp. 218–229. ACM (1987)

[HLM13] Hirt, M., Maurer, U., Lucas, C.: A dynamic tradeoff between active and passive corruptions in secure multi-party computation. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8043, pp. 203–219. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40084-1_12

[IKOS08] Ishai, Y., Kushilevitz, E., Ostrovsky, R., Sahai, A.: Cryptography with constant computational overhead. In: STOC, pp. 433–442 (2008)

[JO08] Jarecki, S., Olsen, J.: Proactive RSA with non-interactive signing. In: Tsudik, G. (ed.) FC 2008. LNCS, vol. 5143, pp. 215–230. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85230-8_20

[JS05] Jarecki, S., Saxena, N.: Further simplifications in proactive RSA signatures. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 510–528. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-30576-7_28

[OY91] Ostrovsky, R., Yung, M.: How to withstand mobile virus attacks (extended abstract). In: PODC, pp. 51–59 (1991)

[Rab98] Rabin, T.: A simplified approach to threshold and proactive RSA. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 89–104. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0055722

[RB89] Rabin, T., Ben-Or, M.: Verifiable secret sharing and multiparty protocols with honest majority. In: Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing, STOC 1989, pp. 73–85. ACM, New York (1989)

[Sha79] Shamir, A.: How to share a secret. Commun. ACM **22**(11), 612–613 (1979)

[Yao82] Yao, A.C.-C.: Protocols for secure computations (extended abstract). In: 23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA, 3–5 November 1982, pp. 160–164. IEEE Computer Society (1982)