Efficient and Secure Multi-Party Computation with Faulty Majority and Complete Fairness

Juan A. Garay* Philip MacKenzie* Ke Yang†

Jan. 12, 2004

Abstract

We study the problem of constructing secure multi-party computation (MPC) protocols that are *completely fair* — meaning that either all the parties learn the output of the function, or nobody does — even when a majority of the parties are corrupted. We first propose a framework for fair multi-party computation, within which we formulate a definition of secure and fair protocols. The definition follows the standard simulation paradigm, but is modified to allow the protocol to depend on the runing time of the adversary. In this way, we avoid a well-known impossibility result for fair MPC with corrupted majority; in particular, our definition admits constructions that tolerate up to $(n-1)$ corruptions, where n is the total number of parties. Next, we define a "commit-prove-fair-open" functionality and construct an efficient protocol that realizes it, using a new variant of a cryptographic primitive known as "time-lines." With this functionality, we show that some of the existing secure MPC protocols can be easily transformed into fair protocols while preserving their security. Putting these results together, we construct efficient, secure MPC protocols that are completely fair even in the presence of corrupted majorities. Furthermore, these protocols remain secure when arbitrarily composed with any protocols, which means, in particular, that they are concurrently-composable and non-malleable. Finally, as an example of our results, we show a very efficient protocol that fairly and securely solves the socialist millionaires' problem.

1 Introduction

Secure multi-party computation (MPC) has been one of the most fundamental problems in cryptography. At a high level, the problem is concerned with n parties, each holding a private input x_i, that want to compute a function $(y_1, y_2, \ldots, y_n) \leftarrow f(x_1, x_2, ..., x_n)$ so that each party learns its own output y_i, but no other information is revealed, even in the presense of malicious parties that may deviate arbitrarily from the protocol [56, 57, 38, 8, 19, 37].

It is standard to define the security of an MPC protocol using a *simulation paradigm*, where two experiments are presented: one *real world* experiment that models the actual setting in which a protocol takes place, and one *ideal process* where an *ideal functionality* performs the desired computation. The security of a protocol is defined as the existence of an *ideal adversary* in the ideal process that *simulates* the view of any real world adversary. Many simulation-based security definitions in various models have been proposed [37, 15, 47, 16, 40, 42]. Among the models, the *universal composability* (UC) framework of Canetti [16] provides perhaps the strongest security guarantee in the following sense: a protocol π that is secure in this framework is guaranteed to remain secure when arbitrarily composed with other protocols, by means of a "composition theorem." In particular, this means that a secure protocol in the UC framework is both concurrently-composable [27] and non-malleable [25].

*Bell Labs – Lucent Technologies, 600 Mountain Ave., Murray Hill, NJ 07974. {garay,philmac}@research.bell-labs.com.

†Computer Science Department, Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh, PA 15213. yangke@cs.cmu.edu.

Fair multi-party computation. This paper focuses on a particular issue in MPC, namely, *fairness*. Informally speaking, a protocol is fair if either all the parties learn the output of the function, or no party learns anything (about the output). This property is also known as "complete fairness," and can be contrasted with "partial fairness," where fairness is achieved only when some conditions are satisfied [40][1]; see also [30]. In this paper, by "fairness" we will always mean complete fairness.

Clearly, fairness is a very desirable property for secure MPC protocols, and in fact, many of the security definitions cited above imply fairness. (Goldwasser and Lindell [40] give an excellent overview of different types of fairness, along with their corresponding histories.) Here we briefly describe some known results about (complete) fairness. Let n be the total number of participating parties and t be the number of corrupted parties. It is known that if $t < n/3$, then fairness can be achieved without any set-up assumptions, both in the information-theoretic setting [8, 19] and in the computational setting [38, 37] (assuming the existence of trapdoor permutations). If $t < n/2$, one can still achieve fairness if all parties have access to a broadcast channel; this also holds both information theoretically [49] and computationally [38, 37].

Unfortunately, the above fairness results no longer hold when $t \geq n/2$, i.e., when a majority of the parties are corrupted. In fact, it was proved that there do not exist fair MPC protocols in this case, even when parties have access to a broadcast channel [20, 37]. This impossibility result easily extends to the *common reference string* (CRS) model (where there is a common string drawn from a prescribed distribution available to all the parties). Intuitively, this is because the adversary, controlling a majority of the corrupted parties, can abort the protocol prematurely and always gain some unfair advantage. Thus, given this impossibility result, much of the previous research on the case of a corrupted majority focuses on protocols with weakened fairness properties, ranging from being partially fair to completely unfair. For example, the basic UC framework is, by definition, completely unfair.[2]

Nevertheless, fairness is still important (and necessary) in many applications in which at least half the parties may be corrupted. One such application is contract signing (or more generally, the fair exchange of signatures) by two parties [9]. To achieve some form of fairness, various approaches have been explored. One such approach adds to the model a trusted third party, who is essentially a judge that can be called in to resolve disputes between the parties. (There is a large body of work following this approach; see, e.g., [2, 14] and references therein.) This approach requires a trusted external party that is constantly available. This is in contast to, for example, the PKI model, where after the public/secret keys are generated, the trusted party is no longer needed. A different approach that avoids the available trusted party requirement uses a mechanism known as "gradual release," where parties take turns to release their secrets in a "bit by bit" fashion, so that if a corrupted party aborts prematurely, it is only a little "ahead" of the honest party. (Note that this is basically an *ad hoc* notion of fairness.) Early works in this category include [9, 28, 32, 39, 5, 22]. More recent work has focused on making sure — under the assumption that there exist problems, such as modular exponentiation, that are not well suited for parallelization[3] — that this "unfairness" factor is bounded by a small constant [12, 36, 48]. (As we discuss below, our constructions also use a gradual release mechanism secure against parallel attacks.)

In this paper we circumvent the impossibility result discussed above and develop a rigorous

[1]For example, in [40] there exists a specific party P_1 such that the protocol is fair as long as P_1 is uncorrupted; but when P_1 is corrupted, then the protocol may become completely unfair.

[2]In the basic UC framework, the adversary in the ideal process can block the outputs from the ideal functionality to all the parties. Thus, the ideal process itself is already completely unfair, and therefore discussing fair protocols is not possible. We note that Canetti [16] also discusses a modified definition in which the ideal process is "non-blocking," meaning that the ideal adversary *eventually* forwards the outputs of the functionality to the uncorrupted parties. It is not clear how fair this is compared to other definitions which guarantee when uncorrupted parties get their outputs.

[3]Indeed, there have been considerable efforts in finding efficient exponentiation algorithms (e.g., [1, 55]) and still the best methods are sequential.

simulation-based security definition (without a trusted third party) which admits fair MPC protocols for the case of corrupted majorities. In a nutshell, we achieve this by allowing the time of the honest parties to depend on, and be slightly greater than, the time of the adversary. Except for this, our underlying framework and security definition is similar to the well known universal compostion framework and security definitions, and thus achieves similar security properties.

Our definition is designed specifically to allow gradual release to be used to achieve fairness. Typical protocols using gradual release consist of a "computation" phase, where some computation is carried out, followed by a "revealing" phase, where the parties gradually release their private information towards learning a result y.[4] Thus to prove security using a simulation-based definition one needs to be able to simulate both the computation phase and the release phase. Previous (*ad hoc*) security definitions did not require this, and consisted, explicitly or implicitly, of the following three statements:

1. The protocol is completely simulatable up to the revealing phase.
2. The revealing phase is completely simulatable *if the simulator knows* y.
3. If the adversary aborts in the revealing phase and computes y by brute force in time t, then all the honest parties can compute y in time comparable to t.

While carrying some intuition about security and fairness, we note that these definitions are not fully simulation-based. To see this, consider a situation where an adversary $\mathcal{A}$ aborts early in the revealing phase, such that it is still infeasible for $\mathcal{A}$ to find y by brute force. At this time, it is also infeasible for the honest parties to find y by brute force. Now, how does one simulate $\mathcal{A}$'s view in the revealing phase? Notice that the revealing phase is simulatable *only if the ideal adversary* $\mathcal{S}$ *knows* y. However, since nobody learns y in the real world, they should not learn y in the ideal world, and, in particular, $\mathcal{S}$ should not learn y. Thus, the above approach gives no guarantee that $\mathcal{S}$ can successfully simulate $\mathcal{A}$'s view. In other words, by aborting early in the revealing phase, $\mathcal{A}$ might gain some unfair advantage. This can become an even more serious security problem when protocols are composed.

Our results We summarize the main results presented in this paper.

1. **A fair multi-party computation framework** In Section 3, we propose a new framework for fair multi-party computation (FMPC), which is a variation of the UC framework, but with modifications so that the ideal process is (intuitively) fair.[5] We then present definitions for secure and fair MPC protocols in this framework that completely fit into the (standard) simulation paradigm and, at the same time, admit protocols that tolerate an arbitrary (i.e., up to $(n-1)$) number of corruptions.

   We avoid the impossibility result [20, 37] by doing a "quantifier switch" in the security definition. Notice that "standard" definitions calls for a single protocol π against all adversaries. But whenever we fix a protocol, there always exists an adversary that gains unfair advantage by aborting. In contrast, we define a *collection* of protocols $\{\pi[t]\}$, parametrized by a time t, and call the collection a *timed protocol.* Then we say a timed protocol is secure, if for any t and any adversary $\mathcal{A}$ whose running time is bounded by t, the particular protocol instance $\pi[t]$ is secure against $\mathcal{A}$. In this way, we quantify the protocols on the running time of the adversary (switching the quantifiers of the protocols and the adversaries), thereby circumventing the impossibility

[4]For simplicity, we assume that all the parties receive the same result y at the end of the protocol. This can be easily extended to the case where each party receives a (different) private output, since y may contain information for each individual party, encrypted using a one-time pad.

[5]We stress that it is always possible to write unfair functionalities even in a framework that allows for fairness. For example, imagine a functionality which, after computing the output (assume it is y), sends y to $\mathcal{S}$ first, waits for an "acknowledgment" from $\mathcal{S}$ back, and then sends y to all parties. In this way, $\mathcal{S}$ will have an opportunity to block the parties by simply not sending the acknowledgment. Thus, this functionality will be inherently unfair. The FMPC framework only provides the *possibility* to have fair functionalities. See also [3], which uses a similar argument to simulate an asynchronous network (which is inherently unfair) using a synchronous one (which can be fair).

result. Furthermore, we prove a composition theorem similar to the one in the UC framework, which implies that if a protocol π is proved secure in the stand-alone model, then it remains secure when arbitrarily composed with any other protocols. In particular, this means that π is concurrently-composable and non-malleable.

2. **The "commit, prove and fair-open" functionality** We define (in Section 5) a *commit-prove-fair-open* functionality $\mathcal{F}_{\text{CPFO}}$ in the FMPC framework. This functionality allows all parties to each commit to a value, prove relations about the committed value, and more importantly, open all committed values simultaneously (and thus fairly). This functionality lies at the heart of our constructions of fair MPC protocols. Furthermore, we construct an efficient protocol GradRel that securely realizes $\mathcal{F}_{\text{CPFO}}$, assuming static corruption. Our protocol uses a new variant of a cryptographic primitive known as *time-lines* [33], which enjoys a property that we call "strong pseudorandomness." In turn, the construction of time-lines hinges on a further generalization of an assumption presented in [12], which has broader applicability. (Section 4.)

3. **Efficient and fair FMPC protocols** We show that by using the $\mathcal{F}_{\text{CPFO}}$ functionality, many existing secure MPC protocols can be easily adapted into fair protocols, while maintaining their security. In particular, we present two such constructions. The first construction converts the universally-composable MPC protocol by Canetti *et al.* [18] to a fair MPC protocol that is secure against static corruption in the CRS model in the FMPC framework. Essentially, the only thing we need to do here is to replace an invocation of a functionality in the protocol called "commit-and-prove" by our $\mathcal{F}_{\text{CPFO}}$ functionality.

   The second construction turns the efficient MPC protocol by Cramer *et al.* [21] into a fair one in the "public key infrastructure" (PKI) model in a similar fashion. The resulting protocol becomes fair and secure (assuming static corruptions) in the FMPC framework, while preserving the efficiency of the original protocol — an additive overhead of only $O(\kappa^2 n)$ bits of communication and an additional $O(\kappa)$ rounds, for κ the security parameter. These results are presented in Section 6.

   Finally, as an illustration of our results, we construct a very efficient protocol that solves the *socialist millionaires' problem* [29, 41] (aka "private equality testing"). Our protocol, which builds on the MPC construction by Cramer *et al.* [21], is completely fair, and remains secure when arbitrarily composed. Compared with the construction by Boudot *et al.* [13], our construction is as efficient (asymptotically), but enjoys much stronger security and fairness. Furthermore, our construction can be easily extended to the multi-party case. These results are presented in Section 7.

2 Preliminaries

Let κ be the cryptographic security parameter. A function $f : \mathbb{Z} \to [0, 1]$ is negligible if for all $\alpha > 0$ there exists an $\kappa_\alpha > 0$ such that for all $\kappa > \kappa_\alpha$, $f(\kappa) < |\kappa|^{-\alpha}$. All functions we use in this paper will include a security parameter as input, either implicitly or explicitly, and we say that these functions are negligible if they are negligible in the security parameter. (They will be polynomial in all other parameters.) Furthermore, we assume that n, the number of parties, is polynomially bounded by κ as well.

A prime p is *safe* if $p' = (p-1)/2$ is also a prime (in number theory, p' is also known as a Sophie-Germain prime). A Blum integer is a product of two primes, each equivalent to 3 modulo 4. We will be working with a special class of Blum integers $N = p_1 p_2$ where p_1 and p_2 are both safe primes. We call such numbers *safe Blum integers*.[6]

[6]Integers that are the product of two equally-sized safe primes have also been called *rigid integers* [6].

We model all computations as non-uniform polynomial-time interactive PRAM machines (IPRAM). IPRAMs are simply extensions to the PRAM machines with special read-only and write-only memories for interacting with each other. For simplicity, we assume that these IPRAM machines can compute modular squaring operations (i.e., computing $x^2 \bmod M$ on input (x, M)) in constant time. All protocols and adversaries are assumed to be polynomial-time unless otherwise stated.

We use the *composite decisional Diffie-Hellman* (CDDH) assumption [11] and the *decisional composite residuosity assumption* (DCRA) [45] in this paper. We also propose a new assumption that we call the *yet-more-general BBS* (YMG-BBS) assumption, which is a further generalization of the *generalized BBS* assumption by Boneh and Naor [12]. See Section 4 for discussions.

The CDDH assumption. We briefly review the *composite decisional Diffie-Hellman* (CDDH) assumption. (We refer the reader to [11] for more in-depth discussions.) Let $N = p_1 p_2$ where p_1, p_2 are κ-bit safe primes. Let g be a random element from $\mathbb{Z}_N^*$, a, b, c random elements in $\mathbb{Z}_N$, and $\mathcal{A}$ a polynomial-time adversary. There exists a negligible function $\epsilon(\cdot)$, such that

$$\left| \Pr[\mathcal{A}(N, g, g^a, g^b, g^{ab}) = 1] - \Pr[\mathcal{A}(N, g, g^a, g^b, g^c) = 1] \right| \leq \epsilon(\kappa)$$

where the randomness is taken over the random choices of N, g, a, b and c. In this paper, we will use a slight variation of this assumption, where instead of being a random element in $\mathbb{Z}_N^*$, g is a random quadratic residue in $\mathbb{Z}_N^*$. We call the new assumption CDDH-QR. Notice that CDDH-QR easily reduces to CDDH. To see this, given a random tuple (N, g, x, y, z) from the CDDH assumption, (N, g^2, x^2, y^2, z^2) is (statistically close to) a random tuple in the CDDH-QR assumption.

3 The Fair Multi-Party Computation Framework

We define the new framework used in our paper, which we call the *fair multi-party computation* (FMPC) framework. It is similar to the universal composability (UC) framework [16]. In particular, there are n parties, $P_1, P_2, \ldots, P_n$, a real-world adversary $\mathcal{A}$, an ideal adversary $\mathcal{S}$, an ideal functionality $\mathcal{F}$, and an environment $\mathcal{Z}$. However, FMPC contains some modifications so that fairness becomes possible. We stress that the FMPC framework still inherits the strong security of the UC, as we shall prove a composition theorem in the FMPC framework similar to UC.

Instead of describing the FMPC framework from scratch, we only discuss its differences from the UC framework. We present a brief overview of the UC framework in Appendix A; refer to [16] for a detailed presentation. The places the FMPC framework differs from the UC framework are:

1. **Interactive circuits/PRAMs.** Instead of interactive Turing machines, we assume the computation models in the FMPC framework are non-uniform interactive PRAMs (IPRAMs). This is a non-trivial distinction, since we will work with exact time bounds in our security definition, and the "equivalence" between various computation models does not carry over there. For technical reasons, we will use machines that allow for simulation and subroutine access with no overhead. Thus, if we have two protocols, and one calls the other as a black-box, then the total running time of the two protocols together will be simply the sum of their running times. Obviously, Turing machines are not suitable here.

2. **Direct-output functionalities.** This is the most important difference between the two frameworks. Recall that in the UC framework, messages from the ideal functionality $\mathcal{F}$ are forwarded to the uncorrupted parties by the ideal adversary $\mathcal{S}$, who may block these messages and never actually deliver them. The ability of $\mathcal{S}$ to block messages from $\mathcal{F}$ makes the UC framework completely unfair. In the FMPC framework, however, messages from $\mathcal{F}$ are directly sent to the

uncorrupted parties, and $\mathcal{S}$ cannot delay these messages. We call this feature "direct-output functionalities."[7]

3. **Synchronous broadcast communication with rounds.** In the UC framework, the communication is asynchronous, and controlled by the adversary. Again, this makes fair MPC impossible, since the adversary may, e.g., choose not to deliver the final protocol message to an uncorrupted party P_i. To prevent this from happening, communication in the FMPC framework takes the form of synchronous, authenticated broadcast and has *rounds* in both the real world and the ideal process. More precisely, we assume that in the real world, all parties know which round they are in, and in each round, all honest parties are activated (in any order), the adversary is then activated and may read their outgoing communication memories, and write on the outgoing communication tapes of corrupted parties. (This is commonly referred to as a *rushing* adversary.) Then all messages in the outgoing communication memories of all parties are transferred to the incoming communication memories of all parties, each one tagged with the identity of the sending party. In the ideal process, the only active parties are the ideal functionality $\mathcal{F}$ and the ideal adversary $\mathcal{S}$. So, by having rounds in the ideal process, we simply mean that $\mathcal{F}$ is defined in terms of rounds (in general, these rounds would match the rounds in the protocol π realizing it), and that $\mathcal{S}$ knows which round it is in. With this round structure, the honest parties will be able to know when the adversary aborts the protocol and take appropriate actions.[8]

With these modifications, it is now possible to discuss fair functionalities. As an important example, we present the *secure function evaluation* functionality $\mathcal{F}_f$ below. Note that it is similar to the homonymous functionality in the UC framework [16], except for (1) the added round structure, (2) the fact that there is no reference to the number of corrupted parties, as in our case it may be arbitrary, and (3) the output is a single public value, instead of private outputs to each party.[9] However, since we are in the FMPC framework, the functionality $\mathcal{F}_f$ is fair in the ideal process.

Functionality $\mathcal{F}_f$

$\mathcal{F}_f$ proceeds as follows, running with security parameter k, parties $P_1, \ldots, P_n$, an adversary $\mathcal{S}$ and a "round" parameter s:

- In the first $(s-1)$ rounds, upon receiving a value (input, sid, v) from P_i, set $x_i \leftarrow v$. As soon as inputs have been received from all parties, compute $y \leftarrow f(x_1, \ldots, x_n)$ and immediately send message $(\mathsf{output}, sid, y)$ to $\mathcal{S}$.
- In the sth round, broadcast message $(\mathsf{output}, sid, y)$.

We note that $\mathcal{S}$ receives the output y if and only if the honest parties receive y, although $\mathcal{S}$ may receive y earlier (recall that $\mathcal{S}$ cannot block the messages from $\mathcal{F}_f$ to the parties). This models the situation in the real world — the corrupted parties may gain some advantage of getting the result earlier by premature abort, but the honest parties will get the output eventually, even if later.

[7] We note that this is different from the "non-blocking ideal adversary" in [16]. We require that a message from the ideal functionality is immediately delivered to the parties, while the non-blocking ideal adversary is only required to "eventually deliver" messages. It is not clear if the two ideal processes are equivalent, but the direct-output functionality appears to be more convenient to use and more intuitive for constructing fair protocols.

[8] We comment that the requirement for the round structure can be somewhat weakened, so long as the honest parties can discover premature abort. For example, it is possible to have a framework where the "global" rounds do not necessarily match the rounds in a protocol, and different protocols can be interleaved. As a particular example, the environment may start a protocol π, and in the middle of it, start a new protocol ρ, finish ρ, and come back to π. This would not affect the fairness of the protocol so long as there is an upper bound for the number of "global" rounds between any two consecutive rounds in a protocol. See [4] for more related discussions on the round structure and synchronous communication that ensure fairness.

[9] We note that an analogous "private-output" functionality can be easily realized by $\mathcal{F}_f$, as discussed in Section 1.

3.1 Timed protocols and security definitions

As discussed in the introduction, we formulate the security in the FMPC framework using the simulation paradigm. To avoid the impossibility result, we add an explicit bound t to the running time of both the real-world adversary $\mathcal{A}$ and the environment $\mathcal{Z}$. We always assume that t is a polynomial of the security parameter κ. We do not explicitly write t as $t(\kappa)$ for simplicity.

Definition 3.1 *We say an IPRAM is t-bounded if it runs for a total of at most t steps.*

We can view a t-bounded IPRAM as a "normal" IPRAM with an explicit "clock" attached to it that terminates the execution after a total number of t cumulative steps (notice that an IPRAM is reactive).

Definition 3.2 (Timed Protocols) *We say a protocol π is* timed *if it has an additional parameter T. We write this as $\pi[T]$ and call T the* timing parameter.

One can think of a timed protocol as a collection of "normal" protocols $\{\pi[t]\}$, each parametrized by a particular time function t (T is simply a placeholder). In the sequel, with slight abuse of notations, we will use $\pi[t]$ to denote a specific instance of the timed protocol $\pi[T]$ where the parameter T is fixed to be t. We can view a normal protocol π as a specific type of timed protocol $\pi[T]$ where $\pi[t]$ is the same protocol π for all t.
We are now ready to define security in the FMPC framework. As discussed in Section 1, the "full security definition" in standard MPC framework cannot be satisfied with a faulty majority. Therefore, we resort to a weaker definition of security, which we call *bounded-adversary security.*

Definition 3.3 (Bounded-Adversary Security) *Let $\pi[T]$ be timed protocol and let $\mathcal{F}$ be an ideal functionality. We say $\pi[T]$* securely realizes *$\mathcal{F}$ if there exists a polynomial-time black-box ideal adversary $\mathcal{S}$, such that for all t, for any t-bounded real-world adversary $\mathcal{A}$ and t-bounded environment $\mathcal{Z}$,*

$$\mathrm{REAL}_{\pi[t],\mathcal{A},\mathcal{Z}} \approx \mathrm{IDEAL}_{\mathcal{F},\mathcal{S}^{\mathcal{A}}(t),\mathcal{Z}}. \tag{1}$$

Remarks We insist on a "universal" black box simulator (ideal adversary) that works against all adversaries and environments of any time bound. This only makes our security definition stronger. We comment, however, that all known constructions of simulators in the UC framework are universal black-box. Notice that the running time of $\mathcal{S}$ is a fixed polynomial, independent of the timing parameter t.[10] This is possible since we only count the "net" time $\mathcal{S}$ takes on its own, and we count an invocation of the real-world adversary $\mathcal{A}$ as one step for $\mathcal{S}$. Naturally, the *total* time $\mathcal{S}$ takes to do the simulation (which includes both $\mathcal{S}$ and $\mathcal{A}$) can be much larger than the net time. Also notice that $\mathcal{S}$ takes t as its input, since $\mathcal{S}$ will need to know the running time of $\mathcal{A}$ and $\mathcal{Z}$ in order to perform the simulation. In the rest of this paper, we often omit this input t for simplicity.

We now define the "fairness" of a protocol, as the ratio of the worst-case running time of uncorrupted parties to the running time of the adversaries.

Definition 3.4 (Fair Protocols) *We say a timed protocol $\pi[T]$ is* λ-fair *if there exists a polynomial $p(\cdot)$ such that for all t, each uncorrupted party's running time in protocol $\pi[t]$ is at most $\lambda \cdot t + p$. If $\lambda = O(n)$, where n is the number of parties, we say $\pi[T]$ is* fair. *If $\lambda = 0$, we say $\pi[T]$ is* strongly fair.

[10] Here, by running time, we mean the running time of $\mathcal{S}$ *per activation*, similar to the UC framework [16].

Remarks

1. In the definition, honest parties in a fair protocols may run longer than a corrupted one, which might appear a little unnatural. However, since it is the corrupted parties that abort, they clearly might have some advantage, and it is only reasonable to give the honest party more time so that they can "catch up." In the worst case, there can be $(n-1)$ corrupted parties against one honest party. Since the honest party may need to invest a certain amount of work against every corrupted party, we expect that the honest party runs about $(n-1)$ times as long as the adversary. This is the intuition behind our fairness definition.

2. On the other hand, if a normal protocol π (i.e., $\pi[t] = \pi$ for any t) is secure in the FMPC framework, then it is strongly fair, since the honest parties' running time does not depend on the timing parameter. One such example is the one-to-many UCZK protocols in [18, 24, 35]. It is easy to verify that these protocols remain secure in the FMPC framework. (Basically this is because as soon as the prover broadcasts its response, every party can verify on its own the validity of the proof.)

A further justification to our fairness definition comes from the fact that we are measuring the *worst case* running time of the honest parties. Typically, it is the case that the "normal" running time of an honest party (i.e., the time to finish the protocol when nobody deviates from the protocol) is much smaller than the worst-case time. To illustrate this point further, we define "reasonable" protocols as follows.

First, we define a *benign* environment $\widetilde{\mathcal{Z}}$ to be one that initiates a single instance of the protocol, activates all parties in each round and does not corrupt any party during the entire execution. Obviously, $\widetilde{\mathcal{Z}}$ (in conjunction with a protocol π) models a "normal," failure-free execution of π. Therefore, we call the time it takes to finish the execution of π with a benign environment $\widetilde{\mathcal{Z}}$ the *normal running time* of π.

Definition 3.5 (Reasonable Protocols) *We say a timed protocol $\pi[T]$ is* reasonable *if the normal running time of $\pi[t]$ is a fixed polynomial independent from t.*

All timed protocols constructed in this paper are reasonable protocols.

Before stating the composition theorem in the FMPC framework, we need some additional notation. Suppose we want to compose timed protocols $\pi[T]$ and $\rho[T]$, where π calls ρ as a subroutine. Since both protocols are timed, we need to specify precisely which instances of $\pi[T]$ and $\rho[T]$ are used. Let $f(\cdot)$ and $g(\cdot)$ be two polynomials. We use $\pi[f(T)]^{\rho[g(T)]}$ to denote a timed-protocol, say $\sigma(T)$, where $\sigma(t)$ is made up of protocol $\pi[f(t)]$ composed with protocol $\rho[g(t)]$. We call functions $f(\cdot)$ and $g(\cdot)$ the *security amplification functions.*

The following composition theorem and its corollary are similar to the respective counterparts in the UC framework. However, since we use timed protocols and keep track of the exact running time, the statements are more complicated.

Theorem 3.6 (Bounded Adversary Composition) *Let $\mathcal{F}$ be an ideal functionality. Let $\pi[T]$ be an n-party timed protocol in the $\mathcal{F}$-hybrid model. Let $\ell(t)$ be an upper bound on the number of times $\pi[t]$ calls $\mathcal{F}$. Assume the running time of the honest parties in protocol $\pi[t]$ is bounded by $\lambda \cdot t + p$. Let $\rho[T]$ be an n-party timed protocol that securely realizes $\mathcal{F}$. Let q be a polynomial that dominates the running times of $\mathcal{F}$ and the ideal adversary for ρ. Then there exists a black-box hybrid-mode adversary $\mathcal{H}$, such that for any t-bounded real-world adversary $\mathcal{A}$ and t-bounded environment $\mathcal{Z}$, we have*

$$\mathrm{REAL}_{\pi[f(t)]^{\rho[g(t)]},\mathcal{A},\mathcal{Z}} \approx \mathrm{HYB}^{\mathcal{F}}_{\pi[f(t)],\mathcal{H}^{\mathcal{A}}(t),\mathcal{Z}}, \tag{2}$$

with security amplification functions $f(t) = t + r$, where r upper-bounds the running time of $\mathcal{H}$, and $g(t) = (2 + \lambda n)t + (np + \ell(t) \cdot q + (1 + \lambda n)r)$.

Proof: The idea of the proof is very similar to the proof of the composition theorem in the UC framework, except that we need to keep track of the explicit running time of the adversaries and the environment.

Our construction of the hybrid-mode adversary $\mathcal{H}$ is similar to the construction in [16], but we have an arbitrary real adversary instead of a dummy adversary.[11] This does not add much complexity: $\mathcal{H}$ runs a copy of $\mathcal{A}$ as a black box, and simulates the messages in the protocol $\rho[g(t)]$ using (as black boxes) the simulator $\mathcal{S}$ for $\rho[g(t)]$ and the ideal functionality $\mathcal{F}$. If an environment $\mathcal{Z}$ can distinguish the real world experiment from the hybrid experiment with an adversary $\mathcal{A}$, then we can construct a new environment $\mathcal{Z}'$ that can distinguish the real-world experiment for protocol $\rho[g(t)]$ with a dummy adversary $\overline{\mathcal{A}}$ from the ideal process with $\mathcal{S}$ and $\mathcal{F}$.

We analyze the running time of $\mathcal{Z}'$ and $\overline{\mathcal{A}}$ so that we are sure that when $\mathcal{Z}$ breaks protocol $\pi[f(t)]^{\rho[g(t)]}$, $\mathcal{Z}'$ breaks the security of protocol $\rho[g(t)]$. Basically, the new environment $\mathcal{Z}'$ is a combination of $\mathcal{Z}$, $\mathcal{H}$, and the protocol $\pi[f(t)]$; the new adversary $\overline{\mathcal{A}}$ simply copies data and thus we do not consider its time. Notice that $\mathcal{H}$ calls $\mathcal{A}$, and also calls the ideal functionality $\mathcal{F}$ and the ideal adversary $\mathcal{S}$ for $\rho[g(t)]$ at most $\ell(f(t))$ times. Thus, the total time of $\mathcal{H}$ (not including the time of $\mathcal{A}$) is dominated by $\ell(f(t)) \cdot q + r$, since the time of $\mathcal{F}$ plus the time of $\mathcal{S}$ is dominated by q. The time to run protocol $\pi[f(t)]$ is at most $\lambda n \cdot (t + r) + n \cdot p$. So if the running time of $\mathcal{Z}$ and $\mathcal{A}$ are bounded by t, then the running time of $\mathcal{Z}'$ is bounded by $g(t) = (2 + \lambda n)t + (np + \ell(f(t)) \cdot q + (1 + \lambda n)r)$. ∎

We now define the security of a timed protocol in the "hybrid" model of computation. The definition is almost the same as Definition 3.3.

Definition 3.7 (Bounded-Adversary Hybrid Security) *Let $\pi[T]$ be a timed protocol and let $\mathcal{F}$ and $\mathcal{G}$ be functionalities. We say $\pi[T]$* securely realizes $\mathcal{G}$ in the $\mathcal{F}$-hybrid model *if there exists a polynomial-time black-box ideal adversary $\mathcal{S}$, such that for all t, for any t-bounded hybrid adversary $\mathcal{H}$ and t-bounded environment $\mathcal{Z}$,*

$$\mathrm{HYB}^{\mathcal{F}}_{\pi[t],\mathcal{H},\mathcal{Z}} \approx \mathrm{IDEAL}_{\mathcal{G},\mathcal{S}^{\mathcal{H}}(t),\mathcal{Z}}. \tag{3}$$

Corollary 3.8 *Let $\mathcal{F}$, $\mathcal{G}$ be ideal functionalities. Let $\pi[T]$ be an n-party timed protocol that securely realizes $\mathcal{G}$ in the $\mathcal{F}$-hybrid model. Let $\ell(t)$ be an upper bound on the number of times $\pi[t]$ calls $\mathcal{F}$. Assume that the running time of the honest parties in protocol $\pi[t]$ is bounded by $\lambda \cdot t + p$. Let $\rho[T]$ be an n-party timed protocol that securely realizes $\mathcal{F}$ and is λ'-fair. Let q be a polynomial that dominates the running times of $\mathcal{F}$ and the ideal adversary for ρ. Let $f(t) = t + r$, where r is an upper bound on the running time of the hybrid adversary $\mathcal{H}$ in Theorem 3.6, and $g(t) = (2 + \lambda n)t + (np + \ell(f(t)) \cdot q + (1 + \lambda n)r)$. Then timed protocol $\pi[f(T)]^{\rho[g(T)]}$ securely realizes $\mathcal{G}$. Furthermore, if $\ell(t)$ is constant (i.e., it does not depend on t), then $\pi[f(T)]^{\rho[g(T)]}$ is $(\lambda\lambda'\ell n + \lambda + 2\lambda'\ell)$-fair.*

Proof: Theorem 3.6 guarantees that there exists an adversary $\mathcal{H}$ in the $\mathcal{F}$-hybrid model such that

$$\mathrm{REAL}_{\pi[f(t)]^{\rho[g(t)]},\mathcal{A},\mathcal{Z}} \approx \mathrm{HYB}^{\mathcal{F}}_{\pi[f(t)],\mathcal{H}^{\mathcal{A}},\mathcal{Z}}$$

for any t-bounded $\mathcal{Z}$ and t-bounded $\mathcal{A}$. That $\pi[f(T)]$ securely realizes $\mathcal{G}$ in the $\mathcal{F}$-hybrid model guarantees that there exists an ideal process adversary $\mathcal{S}$ such that $\mathrm{HYB}^{\mathcal{F}}_{\pi[f(t)],\mathcal{H}^{\mathcal{A}},\mathcal{Z}} \approx \mathrm{IDEAL}_{\mathcal{G},\mathcal{S}^{\mathcal{H}^{\mathcal{A}}},\mathcal{Z}}$, since both $\mathcal{H}^{\mathcal{A}}$ and $\mathcal{Z}$ are $f(t)$-bounded (and actually, $\mathcal{Z}$ is t-bounded). Hence, we have $\mathrm{REAL}_{\pi[f(t)]^{\rho[g(t)]},\mathcal{A},\mathcal{Z}}$ $\approx \mathrm{IDEAL}_{\mathcal{G},\mathcal{S}^{\mathcal{H}^{\mathcal{A}}},\mathcal{Z}}$, for any t-bounded adversary $\mathcal{A}$ and environment $\mathcal{Z}$. The security of $\pi[f(T)]^{\rho[g(T)]}$ thus follows.

[11] A dummy adversary follows instructions from the environment to either send a message, retrieve all messages, or corrupt a party. Thus it simply acts like a vessel for the environment to act on the real system.

For the fairness, assuming $\ell(t)$ does not depend on t, the running time for each uncorrupted party in protocol $\pi[f(t)]$ is bounded by $\lambda \cdot t + (\lambda r + p)$ for some polynomial p, while the running time in protocol $\rho[g(t)]$ is bounded by $\lambda' \cdot g(t) + s$ for some polynomial s. Thus, the running time for protocol $\pi[f(t)]^{\rho[g(t)]}$ is bounded by

$$\lambda \cdot t + (\lambda r + p) + \ell \cdot (\lambda' \cdot g(t) + s) = (\lambda\lambda'\ell n + \lambda + 2\lambda'\ell)t + (\lambda r + p + \ell\lambda'(np + \ell q + (1 + \lambda n)r) + \ell s).$$

Thus, protocol $\pi[f(T)]^{\rho[g(T)]}$ is $(\lambda\lambda'\ell n + \lambda + 2\lambda'\ell)$-fair. ∎

In the sequel, when we perform composition of timed protocols, we refrain from giving the explicit security amplification functions $f(\cdot)$ and $g(\cdot)$ for the sake of clarity.

Remarks

1. Notice that the composition becomes much simpler when one of the two protocols is strongly fair. If protocol ρ is strongly fair (meaning that $\lambda' = 0$) then $\pi[f(T)]^{\rho}$ is a timed protocol that is λ-fair. On the other hand, if π is strongly fair (meaning that $\lambda = 0$), then $\pi^{\rho[2T+np+\ell q+r]}$ is a timed protocol that is $2\lambda'\ell$-fair.
2. The number of times $\pi[f(t)]$ calls $\mathcal{F}$ (or $\rho[g(t)]$) can be naturally bounded by $f(t)$, i.e., $\ell(f(t)) \leq t + r$. Thus, we can state both the composition theorem and its corollary without the knowledge of ℓ. This in particular means that if we use the "right" security amplification functions, the protocol can be made secure with respect to unbounded composition. However, the resulting composed protocol would not be very fair since the honest parties need to run for much longer.

The CRS model and the PKI model Later in this paper we work in the $\mathcal{F}_{\text{CRS}}$-hybrid model, implying that there is a common reference string (CRS) generated from a prescribed distribution accessible to all parties at the beginning of the protocol. The $\mathcal{F}_{\text{CRS}}$ functionality simply returns the CRS. The public key infrastructure (PKI) model is stronger. Upon initial activation, a PKI functionality $\mathcal{F}_{\text{PKI}}$ generates a public string as well as a private string for each party.

4 "Yet More General" BBS Assumptions and Time-Lines

Let N be a Blum integer. Consider a sequence G of elements in $\mathbb{Z}_N$, $(g, g^2, g^{2^2}, g^{2^3}, ..., g^{2^T})$, where each element is the square of the previous one [10]. Assuming a unit cost to perform a modular squaring, one can move "forward" in the sequence in one step; in ℓ steps, one can move ℓ positions forward. If one knows the factorization of N, however, one can compute any element in the sequence efficiently, and therefore "jump" both forward and backward. On the other hand, without the knowledge the factorization of N, it appears (with our current understanding) one can only move forward one step at a time. Building on this intuition, Boneh and Naor in [12] postulate the assumption that points that are "far away" are not only unknown, but also pseudorandom. They call this assumption the *generalized BBS assumption* (G-BBS) Appendix B).

In this section, we introduce a further generalization to the G-BBS assumption. Then we introduce a variant of "time-lines" [33], with a proof of security that relies on the new assumption. First, we introduce some additional notation that will be used in the remainder of the paper.

We use QR_N to denote the quadratic residues modulo N. In other words, $\mathsf{QR}_N = \{x^2 \mid x \in \mathbb{Z}_N^*\}$. For a vector $\vec{a}$, we use $a[i]$ to denote the ith element in A. A vector $\vec{a}$ of dimension d consists of $a[1]$, $a[2]$, ..., $a[d]$. The *distance* between a number x and a vector $\vec{a}$ is the minimal absolute difference between x and elements in $\vec{a}$; we denote this as $\mathsf{Dist}(x, \vec{a})$. More formally, assuming that d is the dimension of $\vec{a}$, we have $\mathsf{Dist}(x, \vec{a}) = \min_{i=1}^{d}\{|x - a[i]|\}$. Finally, we define the "repeated squaring" function as $\mathsf{RepSq}_{N,g}(x) = g^{2^x} \bmod N$.

4.1 The YMG-BBS assumption

In G-BBS the collection of points on the "time-line" (definition below) that the adversary $\mathcal{A}$ sees (Eq. (6)) is fixed to a specific pattern. In our generalization, which we call the **yet-more-general BBS assumption** (YMG-BBS), we relax this condition. Additionally, the assumption allows the point assumed to be pseudorandom to lie at a "far-away" distance in any direction from the given points, rather that only "to the right" of all of them.

More formally, let κ be a security parameter. Let $N = p_1p_2$ be a safe Blum integer with $|p_1| = |p_2| = \kappa$, and let k be an integer bounded from below by κ^c for some positive c. Let $\vec{a}$ be an arbitrary ℓ-dimensional vector where $0 = a[1] < a[2] < \cdots < a[\ell] < 2^k$, and x be an integer between 0 and 2^k such that $\mathsf{Dist}(x, \vec{a}) = S$.[12] Let g be a random element in $\mathbb{Z}_N^*$, and $\vec{u}$ be an ℓ-dimensional vector such that $u[i] = \mathsf{RepSq}_{N,g}(a[i])$, for $i = 1, ..., \ell$. Let $\mathcal{A}$ be a PRAM algorithm whose running time is bounded by $\delta \cdot S$ for some constant δ. Let R be a random element in $\mathbb{Z}_N^*$. The YMG-BBS assumption states that there exists a negligible function $\epsilon(\kappa)$ such that for any $\mathcal{A}$ (with running time bounded by $\delta \cdot D$),

$$\left|\Pr[\mathcal{A}(N, g, \vec{a}, \vec{u}, x, \mathsf{RepSq}_{N,g}(x)) = 1] - \Pr[\mathcal{A}(N, g, \vec{a}, \vec{u}, x, R^2) = 1]\right| \leq \epsilon(\kappa). \tag{4}$$

Intuitively, the assumption states that for any adversary $\mathcal{A}$ whose running time is bounded by $\delta \cdot S$, and who sees a collection of ℓ points on the time-line — with an arbitrary distribution, a point at distance S away from this collection is still not only unknown, but appears pseudorandom.

Note that in YMG-BBS, the running time of the adversary is no longer necessarily (proportional) to a power of two, as in G-BBS. Additionally, the split of parameters in the YMG-BBS — κ and k — is needed since, as stated, the G-BBS would allow an adversary to run in time $2^{\Omega(\kappa)}$. Using this much time, however, the adversary can factor N and easily distinguish points on the time-line from a random point — there exist factoring algorithms of running time $2^{O(\kappa^{1/3}(\log\kappa)^{2/3})}$ (see, e.g., [54] for details). The YMG-BBS fixes this problem by setting an upper bound function of k so that the adversary's running time would not be sufficient to factor N.

The proof of security of the time-lines presented in the next subsection relies on the YMG-BBS assumption. We remark that this assumption is also needed by some of the existing two-party fair exchange constructions [36, 48]. These constructions only assume G-BBS, but proceed by gradually revealing points in a time-line whose distance from the previous point keeps decreasing, thereby deviating from the fixed distribution of points given by Eq. (6).

4.2 Decreasing time-lines

We now present a variant of time-lines [33] that will play an essential role in constructing fair multi-party computation protocols. We first present a definition suitable for our purposes, followed by an efficient way to generate a time-line (according to this definition), the security of which relies on YMG-BBS and CDDH-QR.

Definition 4.1 *Let κ be a security parameter. A* decreasing time-line *is a tuple $L = \langle N, g, \vec{u}\rangle$, where $N = p_1p_2$ is a safe Blum integer where both p_1 and p_2 are κ-bit safe primes, g is an element in $\mathbb{Z}_N^*$, and $\vec{u}$ is a κ-dimensional vector defined as $u[i] = \mathsf{RepSq}_{N,g}(2^\kappa - 2^{\kappa-i})$ for $i = 1, 2, ..., \kappa$. We call N the time-line* modulus*, g the* seed*, the elements of $\vec{u}$ the* points *in L, and $u[\kappa]$ the* end point *in L.*

In the rest of the paper, we will sometimes simply call a decreasing time-line a "time-line."

To randomly generate a time-line, one picks a random safe Blum integer N along with $g \xleftarrow{R} \mathbb{Z}_N^*$ as the seed, and then produces the points. Naturally, one can compute the points by repeated squaring: By squaring the seed g $2^{\kappa-1}$ times, we get $u[1]$, and from then on, we can compute $u[i]$ by squaring $u[i-1]$; it is not hard to verify that $u[i] = \mathsf{RepSq}_{N,u[i-1]}(2^{\kappa-i})$, for $i = 2, ..., \kappa$. Obviously, using this

[12] Notice that in particular, we have $x \geq S$, since $a[1] = 0$.

method to compute all the points would take exponential time. However, if one knows the factorization of N, the time-line can be efficiently computed [12].

Alternatively, and assuming one time-line is already known, Garay and Jakobsson [33] suggested the following way to efficiently generate additional time-lines. Given a time-line L, one can easily *derive* a new time-line from L, by raising the seed and every point in L to a fixed power α. Clearly, the result is a time-line with the same modulus.

Definition 4.2 *Let $L = \langle N, g, \vec{u} \rangle$ and $L' = \langle N, h, \vec{v} \rangle$ be two lines of identical modulus. We say that time-line L' is* derived *from L* with shifting factor α *if there exists an $\alpha \in \mathbb{Z}_{[1, \frac{N-1}{2}]}$ such that $h = g^{\alpha} \bmod N$. We call L the* master *time-line.*

Note that the cost of derivation is just one exponentiation per point, and there is no need to know the factorization of N.

Given a master time-line $L = \langle N, g, \vec{u} \rangle$, there are two methods for computing the "next point" in the derived time-line L'. First, if one knows the shifting factor α, one can simply raise the correponding point in L to the α-th power — we call this the "deriving method." If α is not known, since L' is a time-line, one can still compute $v[i]$ by repeated squaring $v[i-1]$ for $2^{\kappa - i}$ times — we call this the "squaring method." We illustrate this in Figure 1. Clearly, the deriving method is more efficient than the squaring method, especially at the beginning of the time-line, where the squaring method would take exponential time.

$$
\begin{array}{lllllll}
\text{Master time-line } L: & \text{seed } g; & \text{points: } u[1] & \Longrightarrow & u[2] & \Longrightarrow \cdots \Longrightarrow & u[\kappa] \\
 & \downarrow & \quad\ \ \downarrow & & \downarrow & & \downarrow \\
\text{Derived time-line } L': & \text{seed } g^{\alpha}; & \text{points: } (u[1])^{\alpha} & \Longrightarrow & (u[2])^{\alpha} & \Longrightarrow \cdots \Longrightarrow & (u[\kappa])^{\alpha}
\end{array}
\qquad
\boxed{\begin{array}{l} \longrightarrow \text{derivation} \\ \Longrightarrow \text{squaring} \end{array}}
$$

Figure 1: *Computing the next point on a derived time-line*

In fact, without knowing the master time-line L, if an adversary $\mathcal{A}$ of running time $\delta \cdot 2^{\ell}$ sees only the seed and the *last* $(\ell + 1)$ points of a derived time-line L', the previous point (which is distance 2^{ℓ} away) appears pseudorandom to $\mathcal{A}$, assuming that the YMG-BBS assumption holds. Obviously, this pseudorandomness is no longer true if $\mathcal{A}$ also knows the entire master time-line L and the shifting factor α, since it can then use the deriving method to find the previous point (in fact, any point) on L' efficiently. Nevertheless, as we show in the following lemma, assuming CDDH and YMG-BBS, this pseudorandomness remains true if $\mathcal{A}$ knows L, but not the shifting factor α. We call this property the *strong pseudorandomness* of time-lines.[13]

Lemma 4.3 (Strong Pseudorandomness) *Let $L = \langle N, g, \vec{u} \rangle$ be a randomly-generated decreasing time-line and $L' = \langle N, h, \vec{v} \rangle$ be a time-line derived from L with random shifting factor α. Let κ and δ be as in the YMG-BBS assumption. Let $\vec{w}$ be the vector containing the last $(\ell+1)$ elements in $\vec{v}$, i.e., $\vec{w} = (v[\kappa - \ell], v[\kappa - \ell + 1], ..., v[\kappa])$. Let $\mathcal{A}$ be a PRAM algorithm whose running time is bounded by $\delta \cdot 2^{\ell}$ for some constant δ. Let R be a random element in $\mathbb{Z}_N^*$. Then, assuming CDDH and YMG-BBS hold, there exists a negligible function $\epsilon(\cdot)$ such that, for any $\mathcal{A}$,*

$$\left| \Pr[\mathcal{A}(N, g, \vec{u}, h, \vec{w}, v[\kappa - \ell - 1]) = 1] - \Pr[\mathcal{A}(N, g, \vec{u}, h, \vec{w}, R^2) = 1] \right| \leq \epsilon(\kappa). \tag{5}$$

Proof: We consider four different distributions and prove that they are all computationally indistinguishable to each other, which shall imply our lemma.

[13]For convenience, we state the lemma for the specific case of decreasing time-lines. But it is easy to see that it holds for other distributions of points.

At a high level, all four distributions consists of a master time-line L and part of a derived time-line L'. However, in two of the distributions, the master time-line is real, while in the other two the master time-line is "faked." Similarly, in two of the distributions, the derived time-line is "faithful," while in the other two the derived time-line is "unfaithful." We now describe these four distributions in more detail.

$\mathsf{Dist}^{\mathsf{R},\mathsf{F}}$: **Real master time-line, faithful derived time-line.** Let $L = \langle N, g, \vec{u} \rangle$ be a master time-line, $\alpha \xleftarrow{R} \mathbb{Z}_{[1,\frac{N-1}{2}]}$, and $h = g^\alpha \bmod N$. Let $\vec{w}$ be the last $(\ell+1)$ elements of the time-line derived from L with shifting factor α; i.e., $\vec{w} = \langle (u[\kappa-\ell])^\alpha, (u[\kappa-\ell+1])^\alpha, ...(u[\kappa])^\alpha \rangle$. Let $X = (u[\kappa-\ell-1])^\alpha$. $\mathsf{Dist}^{\mathsf{R},\mathsf{F}} = \langle N, g, \vec{u}, h, \vec{w}, X \rangle$.

$\mathsf{Dist}^{\mathsf{R},\mathsf{U}}$: **Real master time-line, unfaithful derived time-line.** Same as in $\mathsf{Dist}^{\mathsf{R},\mathsf{F}}$, except that $X = R^2$ is a random quadratic residue in $\mathbb{Z}_N$, instead of $(u[\kappa-\ell-1])^\alpha$.

$\mathsf{Dist}^{\mathsf{F},\mathsf{F}}$: **Fake master time-line, faithful derived time-line.** Same as in $\mathsf{Dist}^{\mathsf{R},\mathsf{F}}$, except that the first $(\kappa-\ell-1)$ points in the master time-line are fake. More precisely, let $L^* = \langle N, g, \vec{u}^* \rangle$, where $u^*[i] = \mathsf{RepSq}_{N,g}(2^\kappa - 2^{\kappa-i})$ for $i = \kappa-\ell, ..., \kappa$, and $u^*[\kappa-i] = S_i^2$ is a random quadratic residue in $\mathbb{Z}_N$, for $i = 1, ..., \kappa-\ell-1$. The derived time-line is constructed in the same way as in $\mathsf{Dist}^{\mathsf{R},\mathsf{F}}$. $\mathsf{Dist}^{\mathsf{F},\mathsf{F}} = \langle N, g, \vec{u}^*, h, \vec{w}, X \rangle$.

$\mathsf{Dist}^{\mathsf{F},\mathsf{U}}$: **Fake master time-line, unfaithful derived time-line.** Same as in $\mathsf{Dist}^{\mathsf{F},\mathsf{F}}$, except that $X = Q^2$ is a random quadratic residue in $\mathbb{Z}_N$, instead of $(u[\kappa-\ell-1])^\alpha$.

Note that $\mathsf{Dist}^{\mathsf{R},\mathsf{F}}$ and $\mathsf{Dist}^{\mathsf{R},\mathsf{U}}$ are the two distributions that $\mathcal{A}$ tries to distinguish in the lemma. So if we can prove that all the four distributions are indistinguishable, the lemma is proved. We do this in three steps, as follows:

$$\mathsf{Dist}^{\mathsf{R},\mathsf{F}} \overset{\text{(YMG-BBS)}}{\approx} \mathsf{Dist}^{\mathsf{F},\mathsf{F}} \overset{\text{(CDDH)}}{\approx} \mathsf{Dist}^{\mathsf{F},\mathsf{U}} \overset{\text{(YMG-BBS)}}{\approx} \mathsf{Dist}^{\mathsf{R},\mathsf{U}}$$

$\mathsf{Dist}^{\mathsf{R},\mathsf{F}}$ and $\mathsf{Dist}^{\mathsf{F},\mathsf{F}}$ are indistinguishable to an $\mathcal{A}$ of running time $\delta \cdot 2^\ell$: The difference between $\mathsf{Dist}^{\mathsf{R},\mathsf{F}}$ and $\mathsf{Dist}^{\mathsf{F},\mathsf{F}}$ is in the "early" points of the master time-line, namely, points $u[1]$, ..., $u[\kappa-\ell-1]$. Notice that each of these points are at distance at least 2^ℓ away from each other, and from the rest of the points in the time-line. A standard hybrid-argument reduces the indistinguishability between $\mathsf{Dist}^{\mathsf{R},\mathsf{F}}$ and $\mathsf{Dist}^{\mathsf{F},\mathsf{F}}$ to the YMG-BBS assumption.

$\mathsf{Dist}^{\mathsf{F},\mathsf{F}}$ and $\mathsf{Dist}^{\mathsf{F},\mathsf{U}}$ are indistinguishable to any polynomial-time $\mathcal{A}$: The only difference between $\mathsf{Dist}^{\mathsf{F},\mathsf{F}}$ and $\mathsf{Dist}^{\mathsf{F},\mathsf{U}}$ is the element X. In $\mathsf{Dist}^{\mathsf{F},\mathsf{F}}$, $X = (u^*[\kappa-\ell-1])^\alpha = S^{2\alpha}_{\kappa-\ell-1}$, while in $\mathsf{Dist}^{\mathsf{F},\mathsf{U}}$ $X = Q^2$ is a random quadratic residue. Notice that $S_{\kappa-t}$ and Q are both independent from the rest of the elements in their distributions, thus, one can reduce the indistinguishability between $\mathsf{Dist}^{\mathsf{F},\mathsf{F}}$ and $\mathsf{Dist}^{\mathsf{F},\mathsf{U}}$ to the indistinguishability between the tuples $(g, h, S^2_{\kappa-\ell-1}, S^{2\alpha}_{\kappa-\ell-1})$ from $\mathsf{Dist}^{\mathsf{F},\mathsf{F}}$ and $(g, h, S^2_{\kappa-\ell-1}, Q)$ from $\mathsf{Dist}^{\mathsf{F},\mathsf{U}}$, which in turn reduces to the CDDH-QR assumption.[14]

$\mathsf{Dist}^{\mathsf{F},\mathsf{U}}$ and $\mathsf{Dist}^{\mathsf{R},\mathsf{U}}$ are indistinguishable to an $\mathcal{A}$ of running time $\delta \cdot 2^\ell$: Same as in the first step, the indistinguishability is reduced to YMG-BBS via a hybrid argument. ∎

5 The Commit-Prove-Fair-Open Functionality

In this section we present the "commit-prove-fair-open" functionality $\mathcal{F}_{\text{CPFO}}$. Then, we show how to construct a timed protocol, $\mathsf{GradRel}$, that securely realizes the $\mathcal{F}_{\text{CPFO}}$ functionality using the time-lines from the previous section.

Functionality $\mathcal{F}_{\text{CPFO}}$ is described below. Notice that the ideal adversary $\mathcal{S}$ receives the output (the openings) if and only if all parties receive it, although $\mathcal{S}$ may receive it earlier.

[14]We assume that Q and $S^2_{\kappa-\ell-1}$ are powers of g, which is the case except with negligible probability.

Functionality $\mathcal{F}_{\text{CPFO}}^{R,s}$

$\mathcal{F}_{\text{CPFO}}^{R,s}$ is parameterized by a polynomial-time computable binary relation R and a round parameter s. It proceeds as follows, running with parties P_1, P_2, ..., P_n and an adversary $\mathcal{S}$.

Round 1 **(commit phase):** Receive message $(\mathsf{commit}, sid, x_i)$ from every party P_i and broadcast $(\mathsf{RECEIPT}, sid, P_i)$ to all parties and $\mathcal{S}$.

Round 2 **(prove phase):** Receive message $(\mathsf{prove}, sid, y_i)$ from every party P_i, and if $R(y_i, x_i) = 1$, broadcast $(\mathsf{PROOF}, sid, P_i, y_i)$ to all parties and $\mathcal{S}$.

Rounds $3, ..., s$ **(open phase):** Wait to receive message (open, sid) from party P_i, $1 \leq i \leq n$. As soon as all n open messages are received, send $(\mathsf{DATA}, sid, x_1, x_2, ..., x_n)$ to $\mathcal{S}$ immediately, and broadcast $(\mathsf{DATA}, sid, x_1, x_2, ..., x_n)$ to all parties in round s.

Functionality $\mathcal{F}_{\text{CPFO}}$ is similar to the "commit-and-prove" functionality $\mathcal{F}_{\text{CP}}$ in [18], in that both functionalities allow parties to commit to a value v and prove relations about v. Notice that although $\mathcal{F}_{\text{CP}}$ does not provide an explicit "opening" phase, the opening of v can be achieved by proving an "equality" relation. However, $\mathcal{F}_{\text{CPFO}}$ is designed to enforce fairness in the opening, while $\mathcal{F}_{\text{CP}}$ is not concerned with fairness at all. Later in the paper, we shall see that, by replacing some invocations to the $\mathcal{F}_{\text{CP}}$ functionality by invocations to $\mathcal{F}_{\text{CPFO}}$, we can convert the MPC protocol by Canetti *et al.* (which is completely unfair) into a completely fair protocol.

Now we construct a timed protocol, GradRel, that securely realizes the $\mathcal{F}_{\text{CPFO}}$ functionality in the $(\mathcal{F}_{\text{CRS}}, \hat{\mathcal{F}}_{\text{ZK}})$-hybrid model.[15] We use the multi-session version of the "one-to-many" $\hat{\mathcal{F}}_{\text{ZK}}$ functionality from [18], presented in Figure 2. In particular, we need the $\hat{\mathcal{F}}_{\text{ZK}}$ functionality for the following relations.

Functionality $\hat{\mathcal{F}}_{\text{ZK}}^{R}$

$\hat{\mathcal{F}}_{\text{ZK}}^{R}$ proceeds as follows, running parties $P_1, \ldots, P_n$, and an adversary $\mathcal{S}$:

- Upon receiving $(\mathsf{zk\text{-}prove}, sid, ssid, x, w)$ from P_i: If $R(x, w)$ then broadcast $(\mathsf{ZK\text{-}PROOF}, sid, ssid, P_i, x)$. Otherwise, ignore.

Figure 2: *The (multi-session) zero-knowledge functionality for relation R*

Discrete log: $\mathsf{DL} = \{((M, g, h), \alpha) \mid h = g^\alpha \bmod M\}$

Diffie-Hellman quadruple: $\mathsf{DH} = \{((M, g, h, x, y), \alpha) \mid h = g^\alpha \bmod M \ \wedge\ y = x^\alpha \bmod M\}$

Blinded relation: Given a binary relation $R(y, x)$, we define a "blinded" relation $\hat{R}$ as follows

$$\hat{R}((M, g, h, w, z, y), \alpha) = (h = g^\alpha \bmod M) \ \wedge\ R(y, z/w^\alpha \bmod M)$$

Intuitively, $\hat{R}$ "blinds" the witness x using the Diffie-Hellman tuple $(g, h, w, z/x)$. Obviously $\hat{R}$ is an NP relation if R is.

First we describe GradRel informally. The CRS in GradRel consists of a master time-line $L = \langle N, g, \vec{u} \rangle$. To commit to a value x_i, party P_i derives a new time-line $L_i = \langle N, g_i, \vec{v_i} \rangle$, and uses the end point in L_i to "blind" x_i. More precisely, P_i sends $z_i = v_i[\kappa] \cdot x_i$ as a "timeline-commitment" to x_i together with a zero-knowledge proof of knowledge (through $\hat{\mathcal{F}}_{\text{ZK}}^{\mathsf{DL}}$) that it knows the L_i's shifting

[15] Pedantically, we should write the protocol as $\mathsf{GradRel}[T]^R$ since it is a timed protocol and is parameterized by the relation R. However, for simplicity, we write it as $\mathsf{GradRel}^R$ or even GradRel, and we use $\mathsf{GradRel}[t]$ to denote the particular instance of GradRel with the timing parameter being fixed to t.

factor, and thus, x_i [12, 33]. Note that any party can *force-open* the commitment by performing repeated squaring from points in the time-line. However, forced opening can take a long time, and in particular, since $v_i[\kappa]$ is $(2^\kappa - 1)$ steps away from the seed g_i, it appears pseudorandom to the adversary.

The prove phase is directly handled by the $\hat{\mathcal{F}}_{\mathrm{ZK}}^{\hat{R}}$ functionality. The opening phase consists of of κ rounds. In the i-th round, all parties reveal the ith point in their derived time-lines, followed by a zero-knowledge proof that this point is valid (through $\hat{\mathcal{F}}_{\mathrm{ZK}}^{\mathsf{DH}}$), for $i = 1, 2, ...\kappa$. If at any time in the gradual opening stage, a corrupted party aborts (this can be detected since the protocol proceeds in rounds) or submits an invalid proof, all the uncorrupted parties enter a *panic mode*. When in this mode, the uncorrupted parties check how "far away" they are from the end of the time-lines. If the distance is too large so that no party can force-open the timeline-commitments (in fact, the adversary would not even be able to distinguish the end points from random values), the parties simply abort; if, on the other hand, the distance is not too large so that it is possible for the corrupted parties to force-open the timeline-commitments, the uncorrupted parties perform the force-opening as well. In this way, the protocol makes sure that either all parties receive all the openings, or nobody does.

The detailed description of the protocol is given in Figure 3. The δ in the protocol is the constant δ from the YMG-BBS assumption. As a technical note, GradRel assumes that all the committed values are quadratic residues in $\mathbb{Z}_N^*$. Later in the paper we discuss how this assumption can be removed.

Protocol $\mathsf{GradRel}^R$

Set-up: The protocol has a timing parameter t. The CRS consists of a master time-line $L = \langle N, g, \vec{u} \rangle$.

Round 1 **(commit phase)** For each party P_i, $1 \le i \le n$, upon receiving input $(\mathsf{commit}, sid, x_i)$, do:

1. Pick $\alpha_i \overset{R}{\leftarrow} [1, \frac{N-1}{2}]$, set $g_i \leftarrow g^{\alpha_i} \bmod N$, and compute from L a derived time-line $L_i = \langle N, g_i, \vec{v_i} \rangle$.
2. Set $z_i \leftarrow v_i[\kappa] \cdot x_i = (u[\kappa])^{\alpha_i} \cdot x_i \bmod N$ and broadcast message $(\textsc{commit}, sid, P_i, g_i, z_i)$.
3. Send message $(\mathsf{zk\text{-}prove}, sid, 0, (N, g, g_i), \alpha_i)$ to the $\hat{\mathcal{F}}_{\mathrm{ZK}}^{\mathsf{DL}}$ functionality.

All parties output $(\mathsf{RECEIPT}, sid, P_i)$ after receiving $(\mathsf{ZK\text{-}PROOF}, sid, 0, P_i, (N, g, g_i))$ from $\hat{\mathcal{F}}_{\mathrm{ZK}}^{\mathsf{DL}}$.

Round 2 **(prove phase)** For each party P_i, $1 \le i \le n$, upon receiving input $(\mathsf{prove}, sid, y_i)$, do:

1. Send message $(\mathsf{zk\text{-}prove}, sid, 0, (N, g, g_i, u[\kappa], z_i, y_i), \alpha)$ to the $\hat{\mathcal{F}}_{\mathrm{ZK}}^{\hat{R}}$ functionality.
2. After receiving messages $(\mathsf{ZK\text{-}PROOF}, sid, 0, P_i, (N, g, g_i, u[\kappa], z_i, y_i))$ from $\hat{\mathcal{F}}_{\mathrm{ZK}}^{\hat{R}}$, all parties output $(\mathsf{PROOF}, sid, P_i, y_i)$.

Round $r = 3, ..., (\kappa + 2)$ **(open phase)** Let $\ell = r - 2$. For each party P_i, $1 \le i \le n$, do:

1. Broadcast $(\textsc{release}, sid, v_i[\ell])$ and send message $(\mathsf{zk\text{-}prove}, sid, r, (N, g, g_i, u[\ell], v_i[\ell]), \alpha_i)$ to ideal functionality $\hat{\mathcal{F}}_{\mathrm{ZK}}^{\mathsf{DH}}$.
2. After receiving all n RELEASE and ZK-PROOF messages, proceed to the next round. Otherwise, if any of the broadcast messages is missing, go to panic mode.

At the end of round $(\kappa+2)$, compute $x_j = z_j \cdot (v_j[\kappa])^{-1} \bmod N$, for $1 \le j \le n$, output $(\mathsf{DATA}, sid, x_1, x_2, ..., x_n)$ and terminate.

Panic mode: For each party P_i, $1 \le i \le n$, do:

- If $t < \delta \cdot 2^{\kappa - \ell - 1}$, then output $\perp$ in round $(\kappa + 2)$ and terminate.
- Otherwise, for $j = 1, 2, ..., n$, and use $v_j[\ell - 1]$ from the previous round to directly compute x_j committed by P_j as $x_j = z_j \cdot \left(\mathsf{RepSq}_{N, v_j[\ell-1]}(2^{\kappa - \ell + 1} - 1)\right)^{-1} \bmod N$. Then output $(\mathsf{DATA}, sid, x_1, x_2, ..., x_n)$ in round $(\kappa + 2)$ and terminate.

Figure 3: *Protocol* GradRel, *running in the CRS model in* $(\kappa + 2)$ *rounds*

Clearly, protocol GradRel uses $O(\kappa^2 n)$ bits of communication. In the panic mode, each party P_i will perform at most $2^{\kappa-\ell+1} - 1 \leq 4t/\delta$ squarings for each P_j, $j \neq i$. So the total time spent in the panic mode is at most $4nt/\delta$. The time in other phases does not depend on t. Therefore, GradRel is $(4n/\delta)$-fair. Note that GradRel does have the property that if nobody enters the panic mode (i.e., the protocol proceeds normally), then the actual running time is much smaller.

Theorem 5.1 *Assume that YMG-BBS and CDDH hold. Then timed protocol* GradRel *securely realizes the ideal functionality* $\mathcal{F}_{\text{CPFO}}^{R,(\kappa+2)}$ *in the* $(\mathcal{F}_{\text{CRS}}, \hat{\mathcal{F}}_{\text{ZK}}^{\mathsf{DL}}, \hat{\mathcal{F}}_{\text{ZK}}^{\mathsf{DH}}, \hat{\mathcal{F}}_{\text{ZK}}^{\hat{R}})$*-hybrid model, assuming static corruptions. Furthermore,* GradRel *is fair.*

Proof: Let $\mathcal{A}$ be a t-bounded adversary that operates against protocol GradRel. We construct an ideal adversary $\mathcal{S}$ so that no t-bounded environment can distinguish the hybrid model with $\mathcal{A}$ and protocol GradRel$[t]$ from the ideal process with $\mathcal{S}$ and $\mathcal{F}_{\text{CPFO}}$.

At the beginning of the protocol, $\mathcal{S}$ simulates the $\mathcal{F}_{\text{CRS}}$ functionality by generating a master time-line $L = \langle N, g, \vec{u} \rangle$ just as in the real protocol. Note that since $\mathcal{S}$ generates N, it knows the factorization of N. Assume that $N = p_1 p_2$; $\mathcal{S}$ sets $\Lambda = (p_1 - 1)(p_2 - 1)/4$. Then, during the ideal process, $\mathcal{S}$ runs a simulated copy of $\mathcal{A}$. Messages received from $\mathcal{Z}$ are forwarded to the simulated $\mathcal{A}$, and messages sent by the simulated $\mathcal{A}$ to its environment are forwarded to $\mathcal{Z}$. Furthermore, $\mathcal{S}$ also plays the roles of the various ideal ZK functionalities.

We describe the behavior of $\mathcal{S}$ as responses to other parties' actions.

Commitment by an uncorrupted party: When $\mathcal{S}$ sees a broadcast message $(\mathsf{RECEIPT}, sid, P_i)$ from $\mathcal{F}_{\text{CPFO}}$, it means that an uncorrupted party P_i has committed to a value. $\mathcal{S}$ then generates random elements $g_i \stackrel{R}{\leftarrow} \mathbb{Z}_N^*$, $z_i \stackrel{R}{\leftarrow} \mathsf{QR}_N$ and simulates the two broadcast messages in the real world: $(\text{COMMIT}, sid, P_i, g_i, z_i)$ from P_i, and $(\mathsf{ZK\text{-}PROOF}, sid, 0, P_i, (N, g, g_i))$ from $\hat{\mathcal{F}}_{\text{ZK}}^{\mathsf{DL}}$. Effectively, $\mathcal{S}$ "fakes" a derived time-line with g_i being the seed and z_i being the fake timeline-committed value for P_i.

Commitment by a corrupted party: When $\mathcal{S}$ sees a broadcast message $(\text{COMMIT}, sid, P_i, g_i, z_i)$ from a corrupted party P_i (controlled by $\mathcal{A}$), it means that P_i is committing to a value. Then $\mathcal{S}$ acts as the $\hat{\mathcal{F}}_{\text{ZK}}^{\mathsf{DL}}$ functionality and expects the message $(\mathsf{zk\text{-}prove}, sid, 0, (N, g, g_i), \alpha_i)$ from P_i. Assuming that $g_i = g^{\alpha_i} \bmod N$ (otherwise $\mathcal{S}$ ignores this message), $\mathcal{S}$ can then find out the value P_i commits to by $x_i \leftarrow z_i \cdot (u[\kappa])^{-\alpha_i} \bmod N$. $\mathcal{S}$ then sends message $(\mathsf{commit}, sid, x_i)$ to $\mathcal{F}_{\text{CPFO}}$ on behalf of P_i.

Proof by an uncorrupted party: When $\mathcal{S}$ sees a broadcast message $(\mathsf{PROOF}, sid, P_i, y_i)$ from $\mathcal{F}_{\text{CPFO}}$, it means that an uncorrupted party P_i has succeeded in a proof to $\mathcal{F}_{\text{CPFO}}$. $\mathcal{S}$ then simulates this by faking a broadcast message $(\mathsf{ZK\text{-}PROOF}, sid, 0, P_i, (N, g, g_i, u[\kappa], z_i, y_i))$ from the $\hat{\mathcal{F}}_{\text{ZK}}^{\hat{R}}$ functionality.

Proof by a corrupted party: When $\mathcal{S}$ sees a message $(\mathsf{zk\text{-}prove}, sid, 0, (N, g, g_i, u[\kappa], z_i, y_i), \alpha)$ from a corrupted party P_i (controlled by $\mathcal{A}$) to $\hat{\mathcal{F}}_{\text{ZK}}^{\hat{R}}$, it means that P_i is attempting a proof. $\mathcal{S}$ then verifies the witness, and if the verification succeeds, it sends message $(\mathsf{prove}, sid, y_i)$ to $\mathcal{F}_{\text{CPFO}}$ on behalf of P_i.

Simulating the open phase: In the open phase, $\mathcal{S}$ simulates the gradual opening of the uncorrupted parties. Naturally, the simulation proceeds in rounds. Let $m = \kappa - \lfloor \log_2 \left(\frac{t}{\delta}\right) \rfloor - 1$. $\mathcal{S}$ behaves differently in the first $m - 1$ rounds of the Open phase from the last $\kappa - m + 1$ rounds; the difference lies in the release value used in simulating the uncorrupted parties (i.e., the value x in the message $(\text{RELEASE}, sid, P_i, x)$ sent by the uncorrupted parties).

- In the first $m-1$ rounds of the open phase, $\mathcal{S}$ simply uses a random value each time for the value being released. That is, in round ℓ, $1 \le \ell \le (m-1)$, for each uncorrupted party P_i, $\mathcal{S}$ randomly generates $v_{i,\ell} \xleftarrow{R} \mathsf{QR}_N$ and fakes two broadcast messages: $(\text{RELEASE}, sid, P_i, v_{i,\ell})$ from P_i and $(\mathsf{ZK\text{-}PROOF}, sid, \ell, P_i, (N, g, g_i, u[\ell], v_{i,\ell}))$ from $\hat{\mathcal{F}}_{\text{ZK}}^{\mathsf{DH}}$.

  Then, $\mathcal{S}$ waits to receive the RELEASE messages from all the corrupted parties, as well as their $\mathsf{zk\text{-}prove}$ messages to the $\hat{\mathcal{F}}_{\text{ZK}}^{\mathsf{DH}}$ functionality. $\mathcal{S}$ proceeds to the next round if all the anticipated messages are received and verified. If any of the messages is missing, or any of the proofs are incorrect, $\mathcal{S}$ aborts.
- At round m of the open phase, $\mathcal{S}$ switches its strategy. Notice that from this round on, the uncorrupted parties will force-open the commitments even if $\mathcal{A}$ aborts. So $\mathcal{S}$ will find the openings in this round. First, $\mathcal{S}$ sends the messages (open, sid) to $\mathcal{F}_{\text{CPFO}}$ on behalf of every corrupted party P_j, and then immediately receives the opening of all the committed values in the message $(\mathsf{DATA}, sid, x_1, x_2, ..., x_n)$ from $\mathcal{F}_{\text{CPFO}}$.

  Once $\mathcal{S}$ knows the committed value x_i from uncorrupted P_i, $\mathcal{S}$ can now generated a "real" derived time-line for P_i that is consistent with x_i. This is done by producing the time-line *backward:* We know that the end point of the time-line must be z_i/x_i, and thus the other points should be the roots of z_i/x_i. More precisely, for each uncorrupted party P_i, $\mathcal{S}$ computes $w_i = (z_i/x_i)^{(2^{1-2^{\kappa-m}}) \bmod \Lambda} \bmod N$, which is the $2^{2^{\kappa-m}-1}$th root of (z_i/x_i). Then $\mathcal{S}$ fakes broadcast messages $(\text{RELEASE}, sid, P_i, w_i)$ from P_i and $(\mathsf{ZK\text{-}PROOF}, sid, m, P_i, (N, g, g_i, u[i], w_i))$ from $\hat{\mathcal{F}}_{\text{ZK}}^{\mathsf{DH}}$.

  Then, $\mathcal{S}$ waits to receive the open messages from all the corrupted parties, as well as their $\mathsf{zk\text{-}prove}$ messages. As in the previous rounds, it proceeds to the next round if all the messages are received and verified. Otherwise $\mathcal{S}$ aborts the gradual opening.
- From round m on, $\mathcal{S}$ simulates the gradual opening using the time-line generated in round m. More precisely, in the ℓth round, $\mathcal{S}$ sends the message $(\text{RELEASE}, sid, P_i, \mathsf{RepSq}_{N,w_i}(2^{\kappa-m} - 2^{\kappa-\ell}))$ from P_i and fakes the corresponding messages from $\hat{\mathcal{F}}_{\text{ZK}}^{\mathsf{DH}}$.

  Then, as in the previous case, $\mathcal{S}$ waits to receive the broadcast messages and the messages to the $\hat{\mathcal{F}}_{\text{ZK}}^{\mathsf{DH}}$ functionality from all the corrupted parties, and aborts if these messages are not received or incorrect.

Finally, $\mathcal{S}$ outputs what the simulated $\mathcal{A}$ outputs.

This finishes the description of the ideal adversary $\mathcal{S}$. Now we prove that no t-bounded environment $\mathcal{Z}$ can distinguish the hybrid experiment with a t-bounded $\mathcal{A}$ and $\mathsf{GradRel}$ from the ideal process with $\mathcal{S}$ and $\mathcal{F}_{\text{CPFO}}$, or equivalently,

$$\text{HYB}^{\mathcal{F}_{\text{CRS}}, \hat{\mathcal{F}}_{\text{ZK}}}_{\mathsf{GradRel}[t], \mathcal{A}, \mathcal{Z}}(\kappa, z) \approx \text{IDEAL}_{\mathcal{F}_{\text{CPFO}}, \mathcal{S}^{\mathcal{A}}, \mathcal{Z}}(\kappa, z)$$

To do so, we construct a new experiment that we call Mix, which, intuitively, is a "mixture" of the hybrid experiment with the ideal process.

We describe the Mix experiment more precisely. It is like the ideal experiment in that there is an environment, a simulator $\mathcal{S}'$ and a set of dummy parties. $\mathcal{S}'$ behaves like $\mathcal{S}$ in that it runs $\mathcal{A}$ (forwarding messages between $\mathcal{Z}$ and $\mathcal{A}$) and simulates uncorrupted parties and the ideal ZK functionalities for $\mathcal{A}$. However, it does not simulate the CRS functionality (i.e., it does not generate the master time-line, and thus does not know the factorization of N), but instead has access to a CRS functionality along with the $\mathcal{F}_{\text{CPFO}}$ functionality. ($\mathcal{S}'$ forwards any messages between the CRS functionality and $\mathcal{A}$.) Also, its behavior is different from $\mathcal{S}$ in the commit and open phases as follows.

When a corrupted party P_j commits to a value x_j, $\mathcal{S}'$ is used to extract x_j (by acting as $\hat{\mathcal{F}}_{\mathrm{ZK}}^{\mathsf{DL}}$) and send the message $(\mathsf{commit}, sid, x_j)$ to $\mathcal{F}_{\mathrm{CPFO}}$. When an uncorrupted party P_i sends a message $(\mathsf{commit}, sid, x_i)$ on to $\mathcal{F}_{\mathrm{CPFO}}$, $\mathcal{S}'$ simulates the $\mathsf{GradRel}$ protocol for P_i, i.e., it derives a new time-line L_i with a derivation factor α_i, broadcasts the blinded x_i, and sends a $\mathsf{zk\text{-}prove}$ message to $\hat{\mathcal{F}}_{\mathrm{ZK}}^{\mathsf{DL}}$. In addition to this, the commit message is forwarded to $\mathcal{F}_{\mathrm{CPFO}}$.

When a corrupted P_j attempts a proof, again $\mathcal{S}'$ is used to extract the witness by acting as $\hat{\mathcal{F}}_{\mathrm{ZK}}^{\hat{R}}$ and send the appropriate prove message from P_j to $\mathcal{F}_{\mathrm{CPFO}}$. Also, when an uncorrupted party P_i sends a prove message to $\mathcal{F}_{\mathrm{CPFO}}$, $\mathcal{S}'$ simulates the prove phase of the protocol for P_i, and also forward the message to $\mathcal{F}_{\mathrm{CPFO}}$.

$\mathcal{S}'$ simulates the open phase for uncorrupted parties as in the real-world experiment. It can do this since it knows the shifting factors of their time-lines. However, it interacts with corrupted party in the open phase as $\mathcal{S}$ does in the ideal process, namely, it acts as $\hat{\mathcal{F}}_{\mathrm{ZK}}^{\mathsf{DH}}$, receives $\mathsf{zk\text{-}prove}$ messages from these corrupted parties, and verifies them.

The distribution of the output of $\mathcal{Z}$ at the end of the experiment is denoted as $\mathrm{MIX}_{\mathcal{A},\mathcal{S}',\mathcal{Z}}(\kappa, z)$.

Next, we show that Mix is indistinguishable from both the hybrid experiment and the ideal process. This will finish the proof.

First, we show that $\mathrm{HYB}^{\mathcal{F}_{\mathrm{CRS}},\hat{\mathcal{F}}_{\mathrm{ZK}}}_{\mathsf{GradRel}[t],\mathcal{A},\mathcal{Z}}(\kappa, z) \approx \mathrm{MIX}_{\mathcal{A},\mathcal{S}',\mathcal{Z}}(\kappa, z)$. Note that the only difference between the two experiments is that in the real world, $\mathcal{A}$ is interacting with the true $\hat{\mathcal{F}}_{\mathrm{ZK}}^{\mathsf{DL}}$ and $\hat{\mathcal{F}}_{\mathrm{ZK}}^{\hat{R}}$ functionalities, while in Mix, $\mathcal{A}$ is interacting with $\mathcal{S}'$, which simulates the two functionalities. It is easy to see that the simulation is perfect. In the Open phase, $\mathcal{S}'$ behaves exactly as in the real world. Therefore, the two distributions are in fact identical.

Next, we show that $\mathrm{MIX}_{\mathcal{A},\mathcal{S}',\mathcal{Z}}(\kappa, z) \approx \mathrm{IDEAL}_{\mathcal{F}_{\mathrm{CPFO}},\mathcal{S}^{\mathcal{A}},\mathcal{Z}}(\kappa, z)$. The difference between the Mix experiment and the ideal process is that the commitment and the opening by the uncorrupted parties are real in Mix, but faked in the ideal process. More precisely, in the Mix experiment, for each uncorrupted party P_i, the adversary $\mathcal{A}$ (and therefore $\mathcal{Z}$) sees a consistent time-line L_i — or a prefix of it (in the case of premature abort). In the ideal process, however, the first $(m-1)$ points of each uncorrupted party's time-line are replaced by random quadratic residues. Nevertheless, the last $(\kappa - m + 1)$ points of each time-line are real and consistent with the committed value of each uncorrupted party. So the difference lies in the prefixes of the time-lines of the uncorrupted parties. The indistinguishability between the two experiments reduces to the strong pseudorandomness of time-lines (Lemma 4.3) via a standard hybrid argument. ∎

For the following corollary, we simply plug the UCZK protocol from [18] (which relies on the existence of enhanced trapdoor permutations) into protocol $\mathsf{GradRel}$, and observe the the UCZK protocol is strongly fair.

Corollary 5.2 *Assume YMG-BBS and CDDH hold, and that enhanced trapdoor permutations exist. Then there exists a fair timed protocol that securely realizes the ideal functionality $\mathcal{F}_{\mathrm{CPFO}}^{R,(\kappa+2)}$ in the $\mathcal{F}_{\mathrm{CRS}}$-hybrid model, assuming static corruptions.*

5.1 Extending GradRel to the general case

The $\mathsf{GradRel}$ protocol only works if all committed values are quadratic residues. To fake the commitment to a value x, $\mathcal{S}$ needs to fake a timeline-commitment z before knowing x. Then, after seeing x, $\mathcal{S}$ needs to generate a time-line in the "reverse" direction. In particular, $\mathcal{S}$ needs to find the 2^{2^t-1}th roots of z/x for various values of t. So z/x needs to be a quadratic residue. In the simulation, z is chosen to be a quadratic residue, and thus x needs to be a quadratic residue as well.

We can modify our protocol in the following way to allow a timeline-commitment to any x. Observe that -1 has Jacobi symbol $+1$, but is not a quadratic residue modulo N. Let V be an arbitrary element in $\mathbb{Z}_N^*$ with Jacobi symbol -1. Then for any $x \in \mathbb{Z}_N^*$, exactly one of the four elements $\{x, -x, xV, -xV\}$ is a quadratic residue. Consider a modified version of protocol GradRel that additionally contains V in the common reference string. When a party commits to a value x, it makes five timeline-commitments to x_1, x_2, x_3, x_4, y, where $\{x_1, x_2, x_3, x_4\}$ is a random permutation of $\{x, -x, xV, -xV\}$ and $y \in \{1, 4, 9, 16\}$ indicates which x_i is the x ($y = i^2$ means that $x_i = x$). Naturally, the party also needs to provide zero-knowledge proofs that these commitments are consistent. In the open phase, all five values are opened. Obviously, in the case of premature abort, the uncorrupted parties can still force-open all five commitments and recover x. Furthermore, $\mathcal{S}$ can simulate the commitments and the opening for any x. For the commitments to $\{x_1, x_2, x_3, x_4\}$, $\mathcal{S}$ generates random $\{z_1, z_2, z_3, z_4\}$ whose quadratic residuosity modulo p_1 and p_2 are a random permutation of $\{(+1, +1), (+1, -1), (-1, +1), (-1, -1)\}$. Then $\mathcal{S}$ generates a random quadratic residue w as the timeline-commitment for y, since y is always a quadratic residue. When receiving the actual value x, $\mathcal{S}$ can find out its quadratic residuosity modulo p_1 and p_2, and thus can find the correct permutation and fake the openings of $\{z_1, z_2, z_3, z_4\}$ to $\{x, -x, xV, -xV\}$, as well as the fake opening of w to one of the values in $\{1, 4, 9, 16\}$.

The modified GradRel protocol works for any inputs in $\mathbb{Z}_N^*$, and its communication complexity is only a constant times that of GradRel. The proof of security is straightforward.

6 Completely Fair Multi-Party Computation

In this section we show how to construct fair protocols that securely realize the SFE functionality in the FMPC framework. At a high level, our strategy is very simple. Typical secure multi-party protocols (e.g., [21, 18, 24]) contain an "output" phase, in which every party reveals a secret value, and once all secret values are revealed, every party computes the output of the function. We modify the output phase to have the parties invoke the $\mathcal{F}_{\text{CPFO}}$ functionality. A bit more concretely, assuming each party P_i holds a secret value v_i to reveal, each P_i first commits to v_i and then proves its correctness. Finally $\mathcal{F}_{\text{CPFO}}$ opens all the commitments simultaneously.

In particular, we present two constructions that convert the MPC protocols of Canetti *et al.* [18] and Cramer *et al.* [21] into fair MPC protocols.

6.1 Fair MPC in the CRS model

Theorem 6.1 *Assuming the existence of enhanced trapdoor permutations, for any polynomial-time computable function f, there exists a polynomial-time (strongly fair) protocol that securely realizes $\mathcal{F}_f$ in the $\mathcal{F}_{\text{CRS}}, \mathcal{F}_{\text{CPFO}}$-hybrid model in the FMPC framework, assuming static corruptions.*

Proof: Consider the MPC protocol secure against malicious adversaries by Canetti *et al.* [18]. We denote it by π_f. Recall that π_f is "compiled" from another protocol $\hat{\pi}_f$ that is only secure against "honest-but-curious" adversaries. In the compilation process, P_i commits to its initial values using a commit-and-prove functionality called $\mathcal{F}_{\text{CP}}$, and then for every message m that P_i sends in protocol $\hat{\pi}_f$, the compiler makes P_i send a zk-prove message to the $\mathcal{F}_{\text{CP}}$ ideal functionality in protocol π_f to prove that message m was computed correctly. The protocol $\hat{\pi}_f$ itself consists of three *stages* — the input preparation stage, the circuit evaluation stage, and the output stage. In particular, the output stage of $\hat{\pi}_f$ consists of each party P_i broadcasting its share m_i of the output. After the compilation, the output stage in π_f consists of each party P_i broadcasting a m_i along with a proof that m_i is valid.

We modify protocol π_f to make it secure in the FMPC framework. Notice that π_f assumes a broadcast channel, which is built into the FMPC framework, and it is rather straightforward to fit π_f into the round structure of FMPC — we omit these technical details. The non-trivial modification comes at the output stage, where instead of broadcasting m_i, each party P_i commits to m_i by sending message $(\mathsf{commit}, sid, m_i)$ to $\mathcal{F}_{\text{CPFO}}$. In the next round, each party P_i then sends message $(\mathsf{prove}, sid, y_i)$ to $\mathcal{F}_{\text{CPFO}}$ to prove the correctness of m_i. Here y_i is the appropriate string so that the proof to $\mathcal{F}_{\text{CPFO}}$ is equivalent to the proof to $\mathcal{F}_{\text{CP}}$. Finally, all parties send (open, sid) to $\mathcal{F}_{\text{CPFO}}$, which opens all messages m_i simultaneously to all parties. We denote this modified protocol by $\tilde{\pi}_f$.

Next, we describe an ideal adversary $\tilde{\mathcal{S}}$ for the adversary in protocol $\tilde{\pi}_f$. Naturally, $\tilde{\mathcal{S}}$ is adapted from the ideal adversary $\mathcal{S}$ for protocol π_f. In fact, $\tilde{\mathcal{S}}$ behaves exactly as $\mathcal{S}$ until the output stage. In this stage, $\tilde{\mathcal{S}}$, acting as the $\mathcal{F}_{\text{CPFO}}$ functionality, waits to receiving the commit and prove messages from all the corrupted parties and simulates all the RECEIPT and PROOF messages from all parties, both corrupted and uncorrupted. Next, $\tilde{\mathcal{S}}$ waits to receive the open message from all corrupted parties. After all corrupted parties have sent their open messages, $\tilde{\mathcal{S}}$ then sends message $(\mathsf{input}, sid, x_i)$ to $\mathcal{F}_f$ on behalf of all corrupted P_i, where x_i is the private input to P_i (extracted by $\tilde{\mathcal{S}}$ from the initial commitment). After receiving the output from $\mathcal{F}_f$, $\tilde{\mathcal{S}}$ then simulates the messages m_j from uncorrupted parties P_j in the output stage, just as $\mathcal{S}$ does, and uses these messages to simulate the opening of $\mathcal{F}_{\text{CPFO}}$.

To prove that $\tilde{\mathcal{S}}$ is a valid ideal adversary, it suffices to notice that with $\tilde{\mathcal{S}}$, parties in the ideal process receive the output from $\mathcal{F}_f$ (triggered by $\tilde{\mathcal{S}}$), if and only if parties receive the opening of the commitments from $\mathcal{F}_{\text{CPFO}}$ (simulated by $\tilde{\mathcal{S}}$). ▮

Corollary 6.2 *Assuming YMG-BBS, CDDH, and the existence of enhanced trapdoor permutations, for any polynomial-time computable function f, there exists a fair timed protocol that securely realizes $\mathcal{F}_f$ in the $\mathcal{F}_{\text{CRS}}$-hybrid model in the FMPC framework, assuming static corruptions.*

Proof: It directly follows from Theorem 3.6, Corollary 5.2, and Theorem 6.1. ▮

6.2 Efficient fair MPC in the PKI model

Using a protocol by Cramer *et al.* [21] as the starting point, we can construct more efficient MPC protocols in the PKI model, which is a slightly stronger model than the CRS model. Cramer *et al.* prove that for any polynomial-time computable function f (represented as an arithmetic circuit C), there exists a protocol CDN_f that securely realizes $\mathcal{F}_f$ in the PKI model, assuming DCRA and DDH. Furthermore, their protocol uses $O(\kappa n|C|)$ bits of communication and proceeds in $O(d)$ rounds, where κ is the security parameter, $|C|$ is the number of gates in C, and d the depth of C.[16] We note that the protocol is only secure against fewer than $n/2$ corruptions. A crucial ingredient in their construction is a threshold homomorphic cryptosystem (e.g., the threshold version of the Paillier cryptosystem). Values on the wires of C are encrypted, and the parties share the decryption key using an (n, t)-threshold system, so that any $(t+1)$ parties can jointly decrypt, but any t parties cannot. By avoiding the sharing of values (but only sharing the decryption key instead) their construction is very efficient in terms of communication complexity.

Protocol CDN_f in [21] is proven secure in a "modular composition" framework [15], which is somewhat weaker than the UC framework and our FMPC framework. In particular, CDN_f does not remain secure when composed. The problem is that they use a *standard* trapdoor commitment (TC) to construct zero-knowledge protocols (or more precisely, to convert Σ-protocols into "normal" zero-knowledge protocols), and this commitment scheme may be malleable. However, this problem can be solved by replacing the TC scheme by a simulation-sound trapdoor commitment (SSTC) scheme [34, 44]. MacKenzie and Yang [44] proved that this change will make a zero-knowledge protocol universally composable if the underlying protocol has a non-rewinding knowledge extractor. The

[16] The theorem in [21] is more general. We just state a special case of their result, using a threshold version of the Paillier encryption scheme [23, 31] and Pedersen commitment [46] as the trapdoor commitment scheme.

zero-knowledge protocols from [21] can be easily modified to accommodate a non-rewinding extractor, using techniques from, for example, [34]. Notice that there also exist very efficient constructions of SSTC schemes, assuming strong RSA [34, 44]. After the zero-knowledge protocols are strengthened to be universally composable, it is not hard to verify that protocol CDN_f becomes secure in the UC framework.[17]

Next, we modify the joint decryption phase by having all parties invoke the $\mathcal{F}_{\rm CPFO}$ to release their secret information simultaneously. In [21], two homomorphic cryptosystems are proposed: a threshold version of the Paillier cryptosystem and a system based on the quadratic residuosity assumption and DDH. Both systems admit efficient zero-knowledge proofs and joint decryption protocols. Furthermore, in both systems, the joint decryption phase consists of each party P_i broadcasting a single value v_i along with a zero-knowledge proof that v_i is "correct." After all parties broadcast the correct values, every party can then perform the decryption on its own. To make the protocol fair, we fix the cryptosystem to be $(n, n-1)$-threshold, so that only when all parties participate can they decrypt an encrypted message. Then we only need to change the joint decryption phase so that each party P_i commits to its value v_i, proves the correctness of v_i, and then has the $\mathcal{F}_{\rm CPFO}$ functionality open all the values simultaneously. After all these modifications, the resulting protocol is secure in the $\mathcal{F}_{\rm PKI}, \mathcal{F}_{\rm CPFO}$-hybrid model in the FMPC framework.

Finally, plugging in protocol $\mathsf{GradRel}$, which realizes the $\mathcal{F}_{\rm CPFO}$ functionality in the $(\mathcal{F}_{\rm CRS}, \hat{\mathcal{F}}_{\rm ZK})$-hybrid model and efficient UCZK protocols (e.g. constructions based on SSTC [34, 35], or constructions based on UCC [24]), we obtain the following result (assuming that the homomorphic threshold encryption is Paillier):

Theorem 6.3 *Assuming YMG-BBS, CDDH, DCRA, and strong RSA, for any polynomial-time computable function f, there exists a fair timed protocol that securely realizes $\mathcal{F}_f$ in the $\mathcal{F}_{\rm PKI}$-hybrid model in the FMPC framework, assuming static corruptions. Furthermore, this protocol has communication complexity $O(\kappa n|C| + \kappa^2 n)$ bits and consists of $O(d+\kappa)$ rounds.*

The proof is very similar to that of Theorem 6.1.

As an illustration of our results, we now give a fair and efficient solution for a specific multi-party problem.

7 Efficient and Fair Solutions to the Socialist Millionaires' Problem

In the *socialist millionaires' problem* (SMP) [29, 41], two parties each holding a private values, want to know if these two values are equal. The problem is a natural variant to Yao's millionaires' problem [56, 57], where the parties want to find out whose value is greater, without revealing anything else.

SMP can be naturally extended to the multi-party case, where now each party holds a private value, and they want to know if all the values are equal. We call this extended problem MSMP.

Several solutions to SMP have been proposed. Salomaa [52] and Schneier [53] describe solutions for the original millionaires' problem, which can be easily adapted to solve SMP. However, the complexity of these constructions is exponential in the size of the private values, and therefore the protocols are only efficient for very small inputs. Fagin, Naor and Winkler [29] pose the problem, and give a variety of specialized protocols. Jakobsson and Yung [41] give an efficient cryptographic solution for SMP

[17] An alternative approach is to use universally composable commitment (UCC) schemes, instead of SSTC schemes to replace the standard TC scheme. This is the approach Damgård and Nielsen [24] use. In fact, they manage to prove that their protocol is secure against adaptive corruptions in the *non-erasing* model, while the constructions in [34] and subsequently in [44] using SSTCs only achieve security against adaptive corruptions in the *erasing* model. On the other hand, SSTC admits simpler, more efficient constructions than UCC, and thus allows more efficient constructions. Since this paper is only concerned with static corruption, the difference between erasing and non-erasing models does not matter.

for the case of actively malicious players, but their solution is not fair. Boudot *et al.* [13] construct a protocol for SMP that is fair and also very efficient, requiring $O(\kappa)$ exponentiations and $O(\kappa)$ rounds. We note, however, that the fairness and the security definition that is implicit in [13] does not seem to fit into the standard simulation paradigm, nor does it guarantee a bounded advantage to the players. Furthermore, none of the above protocols remain secure when concurrently composed.

We now present a very simple protocol to solve SMP that is completely fair and secure in the FMPC framework, and thus remains secure when arbitrarily composed with any protocol. Our protocol only uses $O(\kappa)$ exponentiations and $O(\kappa)$ rounds, and therefore asymptotically matches the most efficient protocols known.[18]

The idea behind our construction is very simple. Consider MSMP (which has SMP as a special case) cast as a secure function evaluation functionality with function

$$f(x_1, x_2, ..., x_n) = \begin{cases} 0 & \text{if } x_1 = x_2 = \cdots = x_n; \\ 1 & \text{otherwise.} \end{cases}$$

We could directly apply Theorem 6.1 or Theorem 6.3 to compute f. However, this function f does not appear to admit very efficient circuits, and the resulting protocol would not be very efficient. Instead, we use a well-known technique to construct an efficient function that "approximates" f.

We first look at the case of $n = 2$, where MSMP is simply SMP. Assuming that all the players' values are in $\mathbb{Z}_N$, where N is a safe Blum integer, we consider the function $g(x_1, x_2) = (x_1 - x_2) \cdot r$, where r is a random element in $\mathbb{Z}_N^*$ and all the operations are in $\mathbb{Z}_N$ (the field modulo N). Notice that if $x_1 = x_2$, then $g(x_1, x_2) = 0$; and if $x_1 \neq x_2$ and $GCD(x_1 - x_2, N) = 1$, then $g(x_1, x_2)$ is a random element in $\mathbb{Z}_N^*$. Therefore, assuming that $(x_1 - x_2)$ does not contain a non-trivial factor of N, the value of $g(x_1, x_2)$ carries enough information to compute $f(x_1, x_2)$ — in fact, this is the *only* information $g(x_1, x_2)$ carries, as one can easily simulate $g(x_1, x_2)$ given $f(x_1, x_2)$. Thus, we can use function $g(x_1, x_2)$ to "approximate" $f(x_1, x_2)$.

Of course, the above argument is true only when $GCD(x_1 - x_2, N) = 1$ or $x_1 = x_2$. However, notice that N is part of the public parameters and is chosen by a trusted party. Assuming that factoring N is hard, we know that the probability that $x_1 - x_2$ contains a factor of N is negligibly small.[19]

To compute function g using the protocol CDN_g in [21], one only needs three gates. Since g is randomized, the two parties need to compute a joint coin-tossing as well. Effectively, they compute g via a "modified" function $g' = ((x_1, r_1), (x_2, r_2)) = (x_1 - x_2) \cdot (r_1 + r_2)$, where both r_1 and r_2 are private random elements.

Therefore, by instantiating Theorem 6.3 with function $g(x_1, x_2)$, we have the following corollary.

Corollary 7.1 *Assuming YMG-BBS, CDDH, DCRA and strong RSA, there exists a fair timed two-party protocol that solves SMP in the $\mathcal{F}_{\text{PKI}}$-hybrid model in the FMPC framework, assuming static corruptions. Furthermore, the protocol involves $O(\kappa)$ exponentiations and consists of $O(\kappa)$ rounds.*

For MSMP, simply observe that the function $g(x_1, x_2, ..., x_n) = (x_1 - x_n) \cdot r_1 + (x_2 - x_n) \cdot r_2 + \cdots + (x_{n-1} - x_n) \cdot r_{n-1}$, where $r_1, r_2, ..., r_{n-1}$ are random elements in $\mathbb{Z}_N^*$, approximates $f(x_1, x_2, ..., x_n)$ by the same reasoning as above. The function requires $O(n^2)$ gate operations (since the parties need to perform $(n-1)$ joint coin-tossings) and can be computed in constant rounds (since in the protocol CDN_g only multiplications need interaction).

[18] If the fairness requirement is dropped, then the protocol in [13] only needs $O(1)$ exponentiations and $O(1)$ rounds. This is true for our construction as well.

[19] Note that the private inputs x_1 and x_2 are adversarially chosen by the environment $\mathcal{Z}$, which does not know the factorization of N. So any $\mathcal{Z}$ that produces x_1 and x_2 such that $GCD(x_1 - x_2, N) \neq 1$ with non-negligible probability can be converted into an algorithm that factors N.

Corollary 7.2 *Assuming YMG-BBS, CDDH, DCRA and strong RSA, there exists a fair timed n-party protocol that solves MSMP in the $\mathcal{F}_{\mathrm{PKI}}$-hybrid model in the FMPC framework, assuming static corruptions. Furthermore, the protocol involves $O(n^2+\kappa n)$ exponentiations and consists of $O(\kappa)$ rounds.*

We can in fact do even better by using threshold ElGamal encryption modulo a safe Blum integer (assuming Composite DDH [11]) instead of threshold Paillier. A similar technique (although not over a composite) was used in MacKenzie *et al.* [43].

References

[1] L. Adleman and K. Kompella. Using smoothness to achieve parallelism. In *20th STOC*, pp. 528–538, 1988.

[2] N. Asokan, V. Shoup, and M. Waidner. Optimistic Fair Exchange of Digital Signatures (Extended Abstract). In *EUROCRYPT 1998*, pp. 591–606, 1998.

[3] M. Backes. Unifying Simulatability Definitions in Cryptographic Systems under Different Timing Assumptions (Extended Abstract). In *Proceedings of 14th International Conference on Concurrency Theory (CONCUR)*, Marseille, France, pages 350-365, LNCS 2761, September 2003. Full version available as Cryptology ePrint Archive, Report 2003/114, `http://eprint.iacr.org/`.

[4] M. Backes, B. Pfitzmann, M. Steiner, and M. Waidner. Polynomial Fairness and Liveness. In *Proceedings of 15th IEEE Computer Security Foundations Workshop, CSFW '02*, Cape Breton, Nova Scotia, Canada, pages 160 - 174, June 2002.

[5] D. Beaver and S. Goldwasser. Multiparty Computation with Faulty Majority. In *30th FOCS*, pages 503–513, 1990.

[6] J. Benaloh and M. de Mare. One-Way Accumulators: A Decentralized Alternative to Digital Signatures. In *EUROCRYPT 1993* (LNCS 765), pp. 274–285, 1994.

[7] M. Ben-Or, O. Goldreich, S. Micali and R. Rivest. A Fair Protocol for Signing Contracts. *IEEE Transactions on Information Theory* 36(1):40–46, 1990.

[8] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *20th STOC*, pp. 1–10, 1988.

[9] M. Blum. How to exchange (secret) keys. In *ACM Transactions on Computer Systems*, 1(2):175–193, May 1983.

[10] L. Blum, M. Blum, and M. Shub. A simple unpredictable pseudo-random number generator. *SIAM Journal on Computing*, 15(2):364–383, May 1986.

[11] D. Boneh. The decision Diffie-Hellman problem. In *Proceedings of the Third Algorithmic Number Theory Symposium* (LNCS 1423), pp. 48–63, 1998.

[12] D. Boneh and M. Naor. Timed commitments (extended abstract). In *Advances in Cryptology—CRYPTO '00*, volume 1880 of *Lecture Notes in Computer Science*, pages 236–254, Springer-Verlag, 2000.

[13] F. Boudot, B. Schoenmakers and J. Traoré. A fair and efficient solution to the socialist millionaires' problem. In *Discrete Applied Mathematics*, 111(1-2): 23-36 2001.

[14] C. Cachin and J. Camenisch. Optimistic Fair Secure Computation. In *Advances in Cryptology—CRYPTO '00*, volume 1880 of *Lecture Notes in Computer Science*, pages 93–111, Springer-Verlag, 2000.

[15] R. Canetti. Security and Composition of Multiparty Cryptographic Protocols. *Journal of Cryptology*, 13(1):143-202, Winter 2000.

[16] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Symposium on Foundations of Computer Science*, pp. 136–145, 2001.

[17] R. Canetti and M. Fischlin. Universally composable commitments. In *CRYPTO 2001* (LNCS 2139), pp. 19–40, 2001.

[18] R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai. Universally Composable Two-party Computation. In *34th ACM Symposium on the Theory of Computing*, 2002.

[19] D. Chaum, C. Crépeau, and I. Damgård. Multiparty unconditionally secure protocols. In *20th STOC*, pp. 11–19, 1988.

[20] R. Cleve. Limits on the security of coin flips when half the processors are faulty. In *Proceedings of the 18th Annual ACM Symposium on Theory of Computing (STOC 1986)*, pages 364-369 (1986)

[21] R. Cramer, I. Damgård, and J. Nielsen. Multiparty Computation from Threshold Homomorphic Encryption In *Advances in Cryptology - EuroCrypt 2001 Proceedings*, volume 2045 of *Lecture Notes in Computer Science*, pp. 280–300, Springer-Verlag, 2001.

[22] I. Damgård. Practical and Provably Secure Release of a Secret and Exchange of Signatures. In *Journal of Cryptology* 8(4), pp. 201–222, 1995.

[23] I. Damgård and M .Jurik. Efficient protocols based probabilistic encryptions using composite degree residue classes. In *Research Series RS-00-5*, BRICS, Department of Computer Science, University of Aarhus, 2000.

[24] I. Damgård, and J. Nielsen. Universally Composable Efficient Multiparty Computation from Threshold Homomorphic Encryption. In *CRYPTO 2003*, 2003.

[25] D. Dolev, C. Dwork and M. Naor. Non-malleable cryptography. *SIAM J. on Comput.*, 30(2):391–437, 2000. An earlier version appeared in *23rd ACM Symp. on Theory of Computing*, pp. 542–552, 1991.

[26] C. Dwork and M. Naor. Pricing via processing or combatting junk mail. In *Advances in Cryptology—CRYPTO '92*, volume 740 of *Lecture Notes in Computer Science*. Springer-Verlag, pp. 139–147, 1993.

[27] C. Dwork, M. Naor and A. Sahai. Concurrent zero-knowledge. In *30th ACM Symposium on the Theory of Computing*, pp. 409–418, 1998.

[28] S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *Commun. ACM*, 28(6):637–647, June 1985.

[29] R. Fagin, M. Naor and P. Winkler. Comparing information without leaking it. *Communications of the ACM*, 38(5):77-85, 1996.

[30] M. Fitzi, D. Gottesman, M. Hirt, T. Holenstein and A. Smith. Detectable Byzantine Agreement Tolerating Faulty Majorities (from scratch). In *21st PODC*, pages 118–126, 2002.

[31] P. Fouque, G .Poupard, and J. Stern. Sharing decryption in the context of voting or lotteries. In *Proceedings of Financial Crypto 2000*, 2000.

[32] Z. Galil, S. Haber, and M. Yung. Cryptographic Computation: Secure Fault-tolerant Protocols and the Public-Key Model. In *CRYPTO'87*, pages 135–155, 1988.

[33] J. Garay and M. Jakobsson. Timed Release of Standard Digital Signatures. In *Financial Cryptography '02*, volume 2357 of *Lecture Notes in Computer Science*, pages 168–182, Springer-Verlag, pp. 168–182, 2002.

[34] J. Garay, P. MacKenzie and K. Yang. Strengthening Zero-Knowledge Protocols using Signatures. In *Advances in Cryptology – Eurocrypt 2003*, Warsaw, Poland, LNCS 2656, pp.177-194, 2003. Full version available at *ePrint Archive*, report 2003/037, `http://eprint.iacr.org/2003/037`, 2003.

[35] J. Garay, P. MacKenzie and K. Yang. Efficient and Universally Composable Committed Oblivious Transfer and Applications. To appear in *1st Theory of Cryptography Conference*, 2004.

[36] J. Garay and C. Pomerance. Timed Fair Exchange of Standard Signatures. In *Financial Cryptography 2003*, volume 2742 of *Lecture Notes in Computer Science*, pp. 190–207, Springer-Verlag, 2003.

[37] O. Goldreich. Secure Multi-Party Computation (Working Draft, Version 1.2), March 2000. Available from `http://www.wisdom.weizmann.ac.il/ oded/pp.html`.

[38] O. Goldreich, S. Micali, and A. Wigderson. How to Play any Mental Game – A Completeness Theorem for Protocols with Honest Majority. In *19th ACM Symposium on the Theory of Computing*, pp. 218–229, 1987.

[39] S. Goldwasser and L. Levin. Fair computation of general functions in presence of immoral majority, In *CRYPTO '90*, pp. 77-93, Springer-Verlag, 1991.

[40] S. Goldwasser and Y. Lindell. Secure Computation Without Agreement. In *16th DISC*, Springer-Verlag (LNCS 2508), pages 17-32, 2002.

[41] M. Jakobsson and M. Yung. Proving without knowning: on oblivious, agnostic and blindfolded provers. In *CRYPTO '96*, pp. 186–200, LNCS 1109, 1996.

[42] Y. Lindell. General Composition and Universal Composability in Secure Multi-Party Computation.In *FOCS 2003*.

[43] P. MacKenzie, T. Shrimpton, and M. Jakobsson. Threshold password-authenticated key exchange. In *CRYPTO 2002* (LNCS 2442), pp. 385–400, 2002.

[44] P. MacKenzie and K. Yang. On Simulation Sound Trapdoor Commitments. Manuscript.

[45] P. Paillier. Public-key cryptosystems based on composite degree residue classes. In *Advances in Cryptology–Eurocrypt '99*, pp.223–238, 1999.

[46] T. P. Pedersen. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In *Advances in Cryptology – CRYPTO '91* (LNCS 576), 129–140, 1991.

[47] B. Pfitzmann and M. Waidner. Composition and Integrity Preservation of Secure Reactive Systems. In *ACM Conference on Computer and Communications Security (CSS)*, pp. 245–254, 2000.

[48] B. Pinkas. Fair Secure Two-Party Computation. In *Eurocrypt 2003*, pages 87–105, 2003.

[49] T. Rabin and M. Ben-Or. Verifiable Secret Sharing and Multiparty Protocols with Honest Majority. In *21st STOC*, pp. 73–85, 1989.

[50] R. Rivest, A. Shamir, and D. Wagner. Time-lock puzzles and timed-release crypto. In MIT/LCS/TR-684, 1996.

[51] Phillip Rogaway. The Round Complexity of Secure Protocols. *MIT Ph.D. Thesis*, June 1991.

[52] A. Solomaa. Public-key cryptography. *Springer-Verlag*, 1990.

[53] B. Schneier. Applied cryptography. *John Wiley & Sons*, 1996.

[54] V. Shoup. A Computational Introduction to Number Theory and Algebra. *Preliminary book, available at* `http://shoup.net/ntb/`.

[55] J. Sorenson. A Sublinear-Time Parallel Algorithm for Integer Modular Exponentiation. Available from `http://citeseer.nj.nec.com/sorenson99sublineartime.html`.

[56] A. Yao. Protocols for Secure Computation. In *FOCS 1982*, pages 160–164, 1982.

[57] A. Yao. How to generate and exchange secrets. In *FOCS 1986*, pages 162–167, 1986.

A The Universal Composability Framework

We describe the universal composability (UC) framework briefly.

The execution in the real-life model and the ideal process proceeds basically as follows. The environment $\mathcal{Z}$ drives the execution. It can provide input to a party P_i or to the adversary, $\mathcal{A}$ or $\mathcal{S}$. If P_i is given an input, P_i is activated. In the ideal process P_i simply forwards the input directly to $\mathcal{F}$, which is then activated, possibly writing messages on its outgoing communication tape, and then handing activation back to P_i.[20] In the real-life model, P_i follows its protocol, either writing messages on its outgoing communication tape or giving an output to $\mathcal{Z}$. Once P_i is finished, $\mathcal{Z}$ is activated again. If the adversary is activated, it follows its protocol, possibly giving output to $\mathcal{Z}$, and also either corrupting a party, or performing one of the following activities. If the adversary is $\mathcal{A}$ in the real-life model, it may deliver a message from the output communication tape of one honest party to another, or send a message on behalf of a corrupted party. If the adversary is $\mathcal{S}$ in the ideal process, it may deliver a message from $\mathcal{F}$ to a party, or send a message to $\mathcal{F}$. If a party or $\mathcal{F}$ receives a message, it is activated, and once it finishes, $\mathcal{Z}$ is activated

At the beginning of the execution, all participating entities are given the security parameter $k \in \mathbb{N}$ and random bits. The environment is also given an auxiliary input $z \in \{0,1\}^*$. At the end of the execution, the environment outputs a single bit. Let $\mathrm{REAL}_{\pi,\mathcal{A},\mathcal{Z}}$ denote the distribution ensemble of random variables describing $\mathcal{Z}$'s output when interacting in the real-life model with adversary $\mathcal{A}$ and players running protocol π, with input z, security parameter k, and uniformly-chosen random tapes for all participating entities. Let $\mathrm{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ denote the distribution ensemble of random variables describing $\mathcal{Z}$'s output after interacting with adversary $\mathcal{S}$ and ideal functionality $\mathcal{F}$, with input z, security parameter k, and uniformly-chosen random tapes for all participating entities.

A protocol π *securely realizes* an ideal functionality $\mathcal{F}$ if for any real-life adversary $\mathcal{A}$ there exists an ideal-process adversary $\mathcal{S}$ such that no environment $\mathcal{Z}$, on any auxiliary input, can tell with non-negligible advantage whether it is interacting with $\mathcal{A}$ and players running π in the real-life model, or with $\mathcal{S}$ and $\mathcal{F}$ in the ideal-process. More precisely, $\mathrm{REAL}_{\pi,\mathcal{A},\mathcal{Z}}$ $\mathrm{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$, where denotes *computational indistinguishability*. (In particular, this means that for any $d \in \mathbb{N}$ there exists $k_0 \in \mathbb{N}$ such that for all $k > k_0$ and for all inputs z, $|\Pr[\mathrm{REAL}_{\pi,\mathcal{A},\mathcal{Z}}(k,z)] - \Pr[\mathrm{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k,z)]| < k^{-d}$).

To formulate the composition theorem, one must introduce a hybrid model, a real-life model with access to an ideal functionality $\mathcal{F}$. In particular, this $\mathcal{F}$-hybrid model functions like the real-life model, but where the parties may also exchange messages with an unbounded number of copies of $\mathcal{F}$, each copy identified via a unique *session identifier* (sid). The communication between the parties and each one of these copies mimics the ideal process, and in particular the hybrid adversary does not have access to the contents of the messages. Let $\mathrm{HYB}^{\mathcal{F}}_{\pi,\mathcal{A},\mathcal{Z}}$ denote the distribution ensemble of random variables describing the output of $\mathcal{Z}$, after interacting in the $\mathcal{F}$-hybrid model with protocol π. Let π be a protocol in the $\mathcal{F}$-hybrid model, and ρ a protocol that secures realizes $\mathcal{F}$. The composed protocol π^ρ is now constructed by replacing the first message to $\mathcal{F}$ in π by an invocation of a new copy of ρ, with fresh random input, the same sid, and with the contents of that message as input; each subsequent message to that copy of $\mathcal{F}$ is replaced with an activation of the corresponding copy of ρ, with the contents of that message as new input to ρ.

Canetti [16] proves the following composition theorem.

Theorem A.1 [[16]] Let $\mathcal{F}$, $\mathcal{G}$ be ideal functionalities. Let π be an n-party protocol that securely realizes $\mathcal{G}$ in the $\mathcal{F}$-hybrid model, and let ρ be an n-party protocol that securely realizes $\mathcal{F}$. Then protocol π^ρ securely realizes $\mathcal{G}$.

[20]In [18], the behavior is modified. The inputs are forwarded to the functionality by the ideal adversary, which sees the *public header* of the inputs and may choose not to forward them. See [18] for more detailed discussions. We choose to use the old formulation since it is slightly simpler and the distinction does not make a difference when one only deals with static corruption.

B The Generalized BBS Assumption

We present the generalized BBS (GBBS) assumption from [12].

Let n be a positive integer representing a certain security parameter. Let N be a Blum integer, i.e., $N = p_1 p_2$, with p_1 and p_2 as above, $|p_1| = |p_2| = n$. Recall the notion of a Blum-Blum-Shub (BBS) sequence $x_0, x_1, \cdots, x_m$, with $x_0 = g^2 \pmod N$ for a random $g \in \mathbb{Z}_N$, and $x_i = {x_{i-1}}^2 \pmod N$, $1 \leq i \leq m$. It was shown in [10] that the sequence defined by taking the least significant bit of the elements above is polynomial-time unpredictable, provided the quadratic residuosity assumption (QRA) holds. Recall also that these sequences are periodic (although not always purely periodic).

In [12], Boneh and Naor postulate the following generalization of unpredictability of BBS-type sequences. For g and N as above, and a positive integer $k > n'$, let

$$\vec{w}_{g,k} \;=\; < g^2, g^4, g^{16}, \ldots, g^{2^{2^i}}, \ldots, g^{2^{2^{k-1}}}, g^{2^{2^k}} > (\mathrm{mod} N). \tag{6}$$

The $(n', n, \delta, \epsilon)$ **generalized BBS assumption** (G-BBS) states that for any integer $n' < k < n$ and any PRAM algorithm $\mathcal{A}$ whose running time is less than $\delta \cdot 2^k$,

$$\left| \Pr[\mathcal{A}(N, g, k, \vec{w}_{g,k}, g^{2^{2^{k+1}}}) = 1] - \Pr[\mathcal{A}(N, g, k, \vec{w}_{g,k}, R^2) = 1] \right| < \epsilon \tag{7}$$

where the probability is taken over the random choice of an n-bit Blum integer as above, an element $g \stackrel{R}{\leftarrow} \mathbb{Z}_N$ and $R \stackrel{R}{\leftarrow} \mathbb{Z}_N$.