Fairness in an Unfair World: Fair Multiparty Computation from Public Bulletin Boards

Arka Rai Choudhuri
achoud@cs.jhu.edu
Johns Hopkins University

Matthew Green
mgreen@cs.jhu.edu
Johns Hopkins University

Abhishek Jain
abhishek@cs.jhu.edu
Johns Hopkins University

Gabriel Kaptchuk
gkaptchuk@cs.jhu.edu
Johns Hopkins University

Ian Miers
imiers@cs.jhu.edu
Johns Hopkins University

ABSTRACT

Secure multiparty computation allows mutually distrusting parties to compute a function on their private inputs such that nothing but the function output is revealed. Achieving *fairness* — that all parties learn the output or no one does – is a long studied problem with known impossibility results in the standard model if a majority of parties are dishonest.

We present a new model for achieving fairness in MPC against dishonest majority by using public bulletin boards implemented via existing infrastructure such as blockchains or Google's certificate transparency logs. We present both theoretical and practical constructions using either witness encryption or trusted hardware (such as Intel SGX).

Unlike previous works that either penalize an aborting party or achieve weaker notions such as Δ-fairness, we achieve complete fairness using existing infrastructure.

KEYWORDS

Secure Multiparty Computation, Fairness

1 INTRODUCTION

Secure multiparty computation (MPC) allows a collection of mutually distrusting parties to jointly compute a function on their private inputs while revealing nothing beyond the function output. Since its conception three decades ago [41, 62], MPC has found wide applicability to important tasks such as electronic auctions, voting, valuation of assets, and privacy-preserving data mining.

Fairness. Over the years, several security definitions for MPC have been studied. One natural and desirable definition for MPC stipulates that either all parties receive the protocol output or no party does. This is referred to as *fair* MPC.

The notion of fairness is very important (and necessary) in applications such as auctions and contract signing. For example, if Alice is the first to learn she did not win an auction, she may abort, claim a network failure, and try again with a new bid that just exceeds the previous winning bid. More generally when the "value" of the function output may be enhanced by an information asymmetry, e.g., if Alice is better off exclusively knowing the true value of a financial asset than all parties knowing it, fairness is an issue.

In a seminal work, Cleve [25] proved that fair MPC is impossible to realize for general functions when a majority of the parties are dishonest. This result even holds when the parties have access to a trusted setup such as a common reference string.

The pursuit of fairness. In light of Cleve's impossibility result, a vast amount of research effort has been dedicated towards the study of mitigations to the fairness problem. In particular, two prominent lines of research have emerged over the years. The first research direction considers the problem of achieving fairness in the standard model for a *restricted* classes of functions [7–9, 45, 47].

The second research direction studies fairness for *general* functions by augmenting the computation model and/or by relaxing the definition of fairness. The prominent examples in this direction range from using a trusted party to restore fairness [20], to weaker models where the honest parties can recover the output at computational cost or time at most Δ-times that of the adversary [13, 31, 34, 42, 59, 60] (where Δ is a constant), to penalizing aborting parties monetarily [6, 17, 52, 53]. (See Section 2 for a more elaborate discussion.)

While these mitigations are helpful, they fall short of solving the problem in many circumstances. In particular, they either require appointing trusted parties for very specific tasks (related to the protocol) that can be hard to find, or require that the parties' possess precise estimates of the adversary's resources and incentives. If the adversary values exclusive knowledge of the output very highly, it may not be practical to have a large enough computational differential or penalty to deter aborts. Indeed, in many cases it may be impossible value the MPC output at all.

Our Model: Public Bulletin Boards. In this work, we take a new approach to achieving complete fairness in MPC for general functions. We consider a setting where the parties have access to a public ledger, or a *bulletin board* that allows anyone to publish arbitrary strings. Upon publishing its data D on the bulletin board, a party receives a proof (or a signature) to establish that D was published. The bulletin board is *public*, in that anyone can see all of its contents. The main security requirements from the bulletin

CCS'17, Oct. 30–Nov. 3, 2017, Dallas, TX, USA.

ACM ISBN 978-1-4503-4946-8/17/10…$15.00
https://doi.org/10.1145/3133956.3134092

board are that its contents cannot be erased, nor can a proof of publish be forged.

Our choice of the bulletin board model as a viable model for fair MPC is motivated by the fact that implementations of public bulletin boards *already exist in practice*. We can realize a bulletin board from an existing centralized system: Google's certificate transparency project which logs issued certificates.[1]

A decentralized implementation of a bulletin board can be realized from blockchain-based ledgers such as Etherium and Bitcoin implemented with proofs of stake or proofs of work [16]. Proof of work based blockchains rely on the assumption that the majority of the network's computational power is honest. In contrast, proof of stake systems assume that some quorum of users are honest. For each block, proof of stake systems select the quorum users, typically at random but proportional to the amount of currency or stake they have in the system, and that quorum must sign the next block and (randomly) select the next quorum. The signature on a block by the quorum constitutes an unforgeable proof that data is on the bulletin board. In contrast to e.g. byzantine agreement protocols, however, the user group is ad-hoc and tolerant to churn. Proof of Stake and hybrid Proof of Stake/Proof of Work systems are an area of active research.

1.1 Our Results

In this work, we construct theoretical and practical fair MPC protocols for general functions in the bulletin board model. We, in fact, provide general transformations from any (possibly unfair) n-party MPC protocol that supports $t < n$ corruptions to a fair MPC protocol secure against the same number of corruptions. Crucially, the assumptions used in our transformations affect fairness only: the correctness and privacy properties of the underlying MPC scheme are completely preserved even if the assumptions were not to hold.

I. Fair MPC from Witness Encryption. Our first contribution is a fair MPC protocol in the bulletin board model assuming the existence of witness encryption (WE) [37] and injective one-way functions. In order to rely on the standard security of WE, we require the bulletin board's proof of publish to be implemented via unique signatures [43, 56]. If the bulletin board is implemented via standard signatures (e.g., in Google Transparency Certificates) or proofs of stake (e.g., in Etherium), then we require the stronger assumption of extractable witness encryption [19].

Candidate constructions of WE for NP [37, 39] are known from multilinear maps [36]. Since present constructions [26, 27, 36, 38] of multilinear maps are quite inefficient, we view our first construction as a feasibility result. We note, however, that our construction requires WE for a specific NP language for which constructing efficient schemes from simpler assumptions might be easier. Indeed, a fascinating open question for future work is whether WE for the specific language used in our constructions can be implemented from existing constructions for the related notion of hash proof systems [28].

II. Fair MPC from Secure Processors. Our second contribution is a fair MPC protocol in the bulletin board where all the parties have access to secure processors. In fact, Cleve's impossibility result holds even in the presence of secure processors, and was proved recently in [58]. For concreteness, we work with Intel SGX as a secure processor, following the formalization of [59]. For this result, we only need standard cryptographic assumptions such as secret-key authenticated encryption and signatures. We provide an implementation of this protocol in Section 8.

Comparison with recent works. Recently, [6, 17] showed how block-chain based decentralized cryptocurrencies such as Bitcoin can be used to achieve a notion of *fairness with penalties* where aborting parties are forced to pay a pre-agreed financial penalty. We note that while we also use blockchain based bulletin boards in our work, our end result is quite different in that we achieve the standard notion of fairness – either all parties get the output or none do.

Very recently, [59] studied fairness in the model where each party has access to a secure hardware equipped with secure clock. They achieve a notion of Δ-fairness which guarantees that if an aborting adversary can learn the output in time T, then the honest party can also learn the output in time $\Delta \cdot T$ for $\Delta = 2$. A disadvantage of this model is that T is controlled by the adversary, who can set it arbitrarily to create large delay (e.g., in the order of several minutes or hours) between the times when it gets the output and when the honest party does.

We note that while we also use secure hardware for our second result, we do not require them to implement secure clocks.[2] More importantly, we achieve the standard notion of fairness.

1.2 Technical Overview

We now describe the main ideas used in our constructions. For simplicity of exposition, we restrict this discussion to the two-party case. It is easy to generalize the ideas presented below to the multiparty case.

Starting Ideas. Our starting idea is to run an unfair MPC protocol to compute an encryption of the function output as opposed to computing it in the clear. We then design a special decryption procedure such that either no party is able to perform the decryption or both parties can. In other words, we reduce the fairness problem in MPC to the problem of fair decryption.

At first, it may seem that we haven't made any progress because it is unclear why fair decryption would be any easier than achieving fairness for general functions. Indeed, fair decryption was shown to be a *complete* functionality for fair MPC in [46].

Our *key insight is that a public bulletin board can be used to implement a fair decryption protocol for a witness encryption scheme.* We elaborate on this idea below.

Fairness from Witness Encryption. A witness encryption scheme for a language L can be used to encrypt a message m with a statement x in such a manner that the resulting ciphertext can only be decrypted using a witness w for x. We now explain how we use witness encryption to implement our fair MPC protocol.

[1] Looking forward, our protocol only needs to post a constant sized token to the blockchain and this can readily be embedded in a URL or certificate.

[2] In the specific case where the bulletin board is implemented using a proof of work blockchain, we can use secure clocks to achieve stronger security guarantees. This is unnecessary when the bulletin board uses signatures. We discuss this further in Section 8.

In order to securely compute a function f with complete fairness, the parties first run a standard (possibly unfair) MPC protocol to compute a randomized function that takes the private inputs say (y_1, y_2) of the parties and returns a witness encryption ciphertext CT of the desired output $\mathcal{F}(y_1, y_2)$. The statement x associated with CT is set to be such that a valid witness for x corresponds to the proof of posting a "release token" α (to be determined later) on the bulletin board.

The only way for any party to obtain such a witness is to post α on the bulletin board and obtain the corresponding proof of posting σ. However, in doing so, the pair (α, σ) is made public, and therefore, anyone can obtain it. Thus, if a malicious adversary learns the witness for decrypting CT, then so can the honest party since it can simply read the public bulletin board. This mechanism puts the honest party and the adversary on equal footing and resolves the fairness problem.

While the above constitutes the core idea behind our work, we run into several technical issues in implementing this idea. We discuss these next, together with the solutions.

Issue #1: Setting the release token. An immediate issue with implementing the above idea is that we cannot set the release token α to be an a priori fixed value that is known to the adversary. Indeed, if this is the case, then the adversary can simply abort during the execution of the unfair MPC protocol so that it learns the ciphertext CT, but the honest party does not. Now, even if the honest party can obtain (α, σ) once the adversary has posted it on the bulletin board, it cannot learn the output $\mathcal{F}(y_1, y_2)$ since it does not have CT to decrypt.

To address this issue, we set α to be a pair of random values (α_1, α_2) where α_i is chosen by the i-th party. During the initial MPC phase, each party uses α_i as an additional input such that the output of the MPC is (β, CT) where $\beta_i = f(\alpha_i)$ for some one-way function f and $\beta = (\beta_1, \beta_2)$. Now, even given (β, CT), the value α is not completely known to the adversary. Therefore, if it aborts prematurely, then the honest party aborts as well, knowing that the adversary would not be able to recover the output.

On the other hand, if the first phase is successfully completed, then the parties execute a second phase where each party i simply sends over α_i to the other party. Of course, the adversary may abort in this phase after learning α. However, in order to decrypt CT, it will have to post α on the bulletin board which means the honest party would learn it as well. This restores the balance between the honest party and the adversary.

Issue #2: Security of WE. The standard definition of witness encryption only guarantees semantic security for a ciphertext CT if the statement x associated with it is *false*. In our case, the statement is always true. The only way to argue security in this case is to use a stronger notion of extractable witness encryption [19] which guarantees that for any statement x, if an adversary can distinguish between witness encryption of m from an encryption of $m' \neq m$, then one can efficiently recover from that adversary a witness w for x. Now, if the witness w is computationally hard to find, then we can get a contradiction.

It was shown in [19] that for languages with statements that have only polynomially many witnesses, the standard definition of WE implies the stronger definition of extractable WE. We note that if we set f to be an *injective* one-way function and implement the proof of posting on the bulletin board via unique signatures [43, 56], then we can bound the number of valid witnesses. In this case, we can rely on the standard definition of WE.

Issue #3: Rewinding. We run into yet another issue while arguing security of the above construction. Recall that in order to prove security of a fair MPC protocol, we must construct a simulator who can "force" the correct output on the real adversary, provided that the adversary did not abort prematurely. In our protocol, the only opportunity for the simulator to "program" the output is inside the ciphertext CT computed during the initial MPC phase. However, this point in our overall protocol is "too early" for the simulator to determine with enough confidence whether the real adversary is going to later abort or not. If the simulator's decision to program the output turns out to be wrong, then it would immediately lead to a distinguisher between the outputs of the real and ideal experiments.

To deal with this issue, we use a rewinding strategy previously used in [40, 44, 46] to determine the aborting probability of the adversary with enough accuracy, while still ensuring (expected) polynomial running time for the simulator. In order to ensure indistinguishability of the adversary's view in the real and ideal experiments, we allow the simulator to also rewind the bulletin board to a previous state, as and when necessary. Indeed, without this capability, the simulator cannot prevent an adversary from "detecting rewinding" by continuously posting on the bulletin board. A consequence of this is that we must model the bulletin board as a "local" functionality as opposed to a "global" functionality [21, 22]. Furthermore, since our simulator performs rewinding, we only achieve stand-alone security.

Fairness from Secure Hardware. Roughly, the main idea in our second protocol is to replace the witness encryption in the plain model with a secure hardware that implements (essentially) the same functionality as witness encryption. We require that each party is equipped with such a secure hardware (e.g., Intel SGX). While much of the details in this protocol are similar to the previous one, there are some key differences. We explain them below.

Once the parties have "installed" an appropriate program P (discussed below) in their own local secure hardware and attestation of the same is successfully performed by everyone, they run (as in the previous protocol) an execution of a standard MPC protocol to compute an encryption CT of the desired output. Unlike the previous scheme where CT was computed using witness encryption, here we use a regular secret-key encryption scheme. The secret key K used for encryption is secret-shared amongst the parties who use their respective shares as additional inputs to the MPC. The key K is also loaded in each party's secure hardware, and is in fact computed by the secure hardware devices during an initial key-exchange phase.

As in the previous protocol, we require that the ciphertext CT can only be decrypted if the release token α has been posted on the bulletin board. The program P loaded in each party's secure hardware implements such a conditional decryption mechanism. Specifically, upon receiving a ciphertext CT, a release token α and a corresponding proof of posting σ, the program P verifies the validity of α and σ. If the verification succeeds, then it decrypts CT and returns the output; otherwise it returns $\perp$.

We remark upon two security issues: first, in order to prevent malleability attacks, we require that an authenticated encryption scheme is used in order to compute CT. Further, to prevent an adversary from performing a related key attack (by changing its input key share in the MPC), we require that the secure hardware also provide commitments C_i of each key share K_i to all the parties upon generation of K. A party i is required to input the decommitment to C_i in the MPC protocol, and the MPC functionality checks that all the input key shares are valid by verifying the decommitment information.

Second, for this protocol, we can completely dispense with rewinding and instead construct a black-box, non-rewinding simulator. This is because the use of secure hardware allows the simulator to "program" the output at the very end, when the adversary makes a decryption query to its secure hardware.[3] Indeed, in the secure hardware model, the simulator has the ability to observe (and modify) the queries made by the adversary to its secure hardware. This means that when the adversary makes a final decryption query, the simulator can check if it is valid. If this is the case, then it queries the trusted party to obtain the function output. At this point, the simulator sends a "fake" decryption query to the secure hardware that already contains the desired output. Upon receiving this query, the secure hardware returns the programmed output to the adversary. We note that this programming technique for secure hardware was recently used in [59].

Because of the above modifications, in this protocol, we can model the bulletin board as a global functionality. In this manuscript, however, we do not prove UC security of our protocol and leave it for future work.

Realizing the Bulletin Board. Our constructions assume a public bulletin board that is capable of producing an unforgeable proof that a string has been published to the bulletin board. Such bulletin boards can easily be constructed practice if one is willing to instantiate the board using a single trusted party. While this seems a strong assumption, the advantage of this approach is that such systems already exist and have been widely deployed in practice for applications such as Certificate Transparency [1]. Re-using them to achieve fairness in *arbitrary* MPC protocols requires no specific to the existing systems.

Alternatively, a bulletin board can be realized using a decentralized systems such as proof of stake blockchains (*e.g.,* [49]). These systems allow a quorum of honest users – who together possess a majority ownership "stake" in a cryptocurrency – to securely authenticate an append-only log using signatures. Finally, a weaker notion of security can be achieved using a proof of work blockchain. In the latter case, the "proof" of publication is not a cryptographically unforgeable signature, but rather the solution to a sequence of one or more computational puzzles which may be, in practice, prohibitively expensive for an attacker to forge.[4] We explore this approach in our experimental implementation, although we stress that this is merely an implementation detail. Our bulletin board could easily be replaced with one of the alternatives above.

Optimizations. We mention a few optimizations to the above protocols to improve efficiency. First, we can add an *optimistic decryption phase* in the above protocols that allows the parties to learn the output using a simple decryption process, without using the bulletin board, provided that all the parties are honest. Roughly, the MPC protocol executed in the first phase now additionally computes another encryption CT' of the function output, where CT' is implemented using a regular encryption scheme. The decryption key K' corresponding to CT' is secret-shared between the parties. Now, if the release-token exchange performed in the second phase is successful, then the parties execute a third phase (that we refer to as the optimistic decryption phase) where they exchange the key shares corresponding to K'. If all the parties are honest, then they all learn K' and use it to decrypt CT', without using the bulletin board. However, if one or more parties are adversarial and abort in this phase, then the honest parties can still post the release token α (that they learned in the second phase) on the bulletin board and then use the proof of posting to decrypt CT as before.

We remark that in order to avoid related key attacks by an adversary, we would need a slight modification to the above protocol where the MPC in the first phase outputs commitments to each key share K'_i to both the parties. During the optimistic decryption phase, each party must reveal the decommitment value together with K'_i. A party only accepts the key share as valid if the associated decommitment information is correct.

Finally, we note that the size of the release token $\alpha = (\alpha_1, \alpha_2)$ used in the above described protocols grows with the number of parties N. However, it is easy to make it independent of N by setting $\alpha = \oplus_i \alpha_i$ and using $\beta = f(\alpha)$ to verify the correctness of release token. An advantage of this modification is that the witness length for the witness encryption used in our construction, as well as the length of the string that is posted on the bulletin board becomes independent of the number of parties.

2 RELATED WORK

A large body of research work has addressed the problem of fairness in secure protocols over the years. Below, we provide a non-exhaustive summary of prior works. A more elaborate summary can be found, e.g., in [17].

Fairness in Standard Model. Assuming an honest majority of parties, fair MPC can be achieved in both computational [41] and information-theoretic setting [61]. Cleve [25] proved the impossibility of MPC for general functions n the dishonest majority setting. Subsequently, an exciting sequence of works [7–9, 45, 47] have shown that complete fairness can still be achieved for a restricted class of functions. The works of [5, 14, 48] study the problem of partial fairness.

Optimistic Models. Starting from the early work of [15], optimistic models for fair exchange have been studied in a long sequence of works [11, 12, 30, 33, 54, 57]. An optimistic model for fair two-party computation using a semi-trusted third party was studied in [20, 51].

[3] We also use an MPC in the common random string (CRS) model (e.g., [23]) to implement the first phase of the protocol. By using the CRS trapdoor, the simulator for this phase can avoid any rewinding of the adversary.

[4] In practice, such proof of work blockchains provide a slightly weaker security that is related to Δ-fairness. An attacker, given enough time, may be able to forge the proof of work necessary to prove publication. However, in the trusted hardware setting we are able to mitigate this concern to some extent by requiring the attacker to provide a proof in a limited period of time, as judged by the hardware.

Gradual Release Mechanisms. A different approach to fairness that avoids trusted third parties was considered in a long sequence of works [18, 32, 35, 60], following the early works of [13, 31, 42]. The protocols in these works employ a "gradual release" mechanism where the parties take turns to release their secrets in a bit-by-bit fashion. The intuitive security guarantee (formalized in [34]) is that even if an adversary aborts prematurely, the honest party can recover the output in time comparable to that of the adversary by investing equal (or more) computational effort.

Δ-Fairness. Very recently, [59] considered a notion of Δ-fairness with the guarantee that if an adversary aborts, then the honest party can learn the output in time $\Delta \cdot T$, where T is the time in which the adversary would learn the output. They propose a fair two-party computation protocol with ($\Delta = 2$)-fairness assuming that all the parties have secure hardware equipped with secure clocks.

Fairness with Penalties. Recently, with the popularity of decentralized cryptocurrencies such as Bitcoin, a sequence of works [6, 17, 52, 53] have shown how to implement a fairness-with-penalties model for MPC where adversarial parties who prematurely abort are forced to pay financial fines. Prior works in similar spirit considered fairness with reputation systems [10] and legally enforced fairness [24, 55].

3 DEFINITIONS

3.1 Fair Multi Party Computation

A secure fair multi-party computation protocol is a protocol executed by n number of parties $P_1, \cdots, P_n$ for a n-party functionality F. We allow for parties to exchange messages simultaneously. In every round, every party is allowed to broadcast messages to all parties. We require that at the end of the protocol, all the parties receive the output $F(x_1, \ldots, x_n)$, where x_i is the i^{th} party's input.[5] We formalize the security notion below.

Ideal World. We start by describing the ideal world experiment where n parties $P_1, \cdots, P_n$ interact with an ideal functionality for computing a function F. An adversary may corrupt any subset $\mathcal{P}^{\mathcal{A}} \subset \mathcal{P}$ of the parties. We denote the honest parties by $\mathcal{H}$.

Inputs: Each party P_i obtains an initial input x_i. The adversary Sim is given auxiliary input z. Sim selects a subset of the parties $\mathcal{P}^{\mathcal{A}} \subset \mathcal{P}$ to corrupt, and is given the inputs x_k of each party $P_k \in \mathcal{P}^{\mathcal{A}}$.

Sending inputs to trusted party: Each honest party P_i sends its input x_i to the trusted party. For each corrupted party $P_i \in \mathcal{P}^{\mathcal{A}}$, the adversary may select any value x_i^* and send it to the ideal functionality.

Trusted party computes output: Let $x_1^*, \ldots, x_n^*$ be the inputs that were sent to the trusted party. If any of the received inputs were $\perp$, then the trusted party sends $\perp$ to all the parties. Else, the trusted party sends $F(x_1^*, \ldots, x_n^*)$ to all the parties.

Outputs: Honest parties output the function output they obtained from the ideal functionality. Malicious parties may output an arbitrary PPT function of the adversary's view.

[5] One can also consider asymmetric functionalities where every party receives a different output. Since there are generic transformations from the symmetric case to the asymmetric case, we only consider symmetric functionalities for simplicity of exposition.

The overall output of the ideal-world experiment consists of the outputs of all parties. For any ideal-world adversary Sim with auxiliary input $z \in \{0,1\}^*$, input vector $\vec{x}$, and security parameter λ, we denote the output of the corresponding ideal-world experiment by $\mathsf{IDEAL}_{\mathsf{Sim},F}\left(1^\lambda, \vec{x}, z\right)$.

Real World. The real world execution begins by an adversary $\mathcal{A}$ selecting any arbitrary subset of parties $\mathcal{P}^{\mathcal{A}} \subset \mathcal{P}$ to corrupt. The parties then engage in an execution of a real n-party protocol Π. Throughout the execution of Π, the adversary $\mathcal{A}$ sends all messages on behalf of the corrupted parties, and may follow an arbitrary polynomial-time strategy. In contrast, the honest parties follow the instructions of Π.

At the conclusion of all the update phases, each honest party P_i outputs whatever output it received from the computation. Malicious parties may output an arbitrary PPT function of the view of $\mathcal{A}$.

For any adversary $\mathcal{A}$ with auxiliary input $z \in \{0,1\}^*$, input vector $\vec{x}$, and security parameter λ, we denote the output of the MPC protocol Π by $\mathsf{REAL}_{\mathcal{A},\Pi}\left(1^\lambda, \vec{x}, z\right)$.

MPC with Complete Fairness. We say that a protocol Π is a secure protocol if any adversary, who corrupts a subset of parties and runs the protocol with honest parties, gains *no information* about the inputs of the honest parties beyond the protocol output.

Definition 3.1. A protocol Π is a secure n-party protocol computing F with complete fairness if for every PPT adversary $\mathcal{A}$ in the real world, there exists a PPT adversary Sim corrupting the same parties in the ideal world such that for every initial input vector $\vec{x}$, every auxiliary input z, it holds that

$$\mathsf{IDEAL}_{\mathsf{Sim},F}\left(1^\lambda, \vec{x}, z\right) \approx_c \mathsf{REAL}_{\mathcal{A},\Pi}\left(1^\lambda, \vec{x}, z\right).$$

Security with Abort. For our constructions, we shall require a weaker security notion of MPC referred to as security with abort. This definition differs from the above only in the ideal world, where the adversary receives the output prior to the honest parties and then decides if the trusted party should give the output to the honest parties or not.

3.2 Authentication Scheme with Public Verification

An authentication scheme with public verification consists of three polynomial algorithms (Gen, Tag, Verify).

- Gen is PPT algorithm that takes as input λ and generates a key for signing. $\mathsf{sk} \leftarrow \mathsf{Gen}(\lambda)$.
- Tag is a deterministic algorithm that computes a tag on a message x. $\sigma = \mathsf{Tag}_{\mathsf{sk}}(x)$.
- Verify is a deterministic algorithm that allows for public verification of the tag. $\mathsf{Verify}(x, \sigma)$ returns 1 if the tag σ verifies.

Definition 3.2. A scheme $\Sigma = (\mathsf{Gen}, \mathsf{Tag}, \mathsf{Verify})$ is an authentication scheme with public verification if for any sequence of messages $m_1, \ldots, m_q$ and any PPT adversary $\mathcal{A}$, the following is negligible

in the security parameter:

$$\Pr\left[\begin{array}{l} \mathsf{sk} \leftarrow \mathsf{Gen}(\lambda); \\ \forall i\ \sigma_i = \mathsf{Tag}_{\mathsf{sk}}(m_i); \\ (m', \sigma') \leftarrow \mathcal{A}\left(\{m_i, \sigma_i\}_{i=1}^{q}\right) \end{array} : \begin{array}{l} \mathsf{Verify}(m', \sigma') = 1 \\ \wedge\, m' \notin \{m_1, \ldots, m_q\} \end{array}\right]$$

3.3 Witness Encryption

In this section, we define witness encryption [39] and state its relation with extractable witness encryption [19] for polynomial witness languages.

Definition 3.3 (Extractable Witness Encryption). An extractable witness encryption $\mathsf{ExtWE} = (\mathsf{Enc}, \mathsf{Dec})$ for a NP language $\mathcal{L}$ associated with relation R consists of the following algorithms:

- **Encryption,** $\mathsf{Enc}(1^\lambda, x, m)$: On input instance x and message $m \in \{0,1\}$, it outputs a ciphertext CT.
- **Decryption,** $\mathsf{Dec}(\mathsf{CT}, w)$: On input ciphertext CT and witness w, it outputs m'.

We require that the above primitive satisfies the following properties:

- **Correctness**: For every $x \in \mathcal{L}$, let w be such that $(x, w) \in R$, for every $m \in \{0,1\}$,

$$\Pr[m \leftarrow \mathsf{Dec}(\mathsf{Enc}(x, m), w)] = 1$$

- **Security**: Let $\mathcal{A}$ be a PPT adversary such that the following holds: for every x, m_0, m_1, every auxiliary information $z \in \{0,1\}^{\mathsf{poly}(\lambda)}$:

$$\left|\Pr[1 \leftarrow \mathcal{A}(1^\lambda, \mathsf{Enc}(x, m_0))] - \Pr[1 \leftarrow \mathcal{A}(1^\lambda, \mathsf{Enc}(x, m_1))]\right| \leq \varepsilon$$

Then there exists a PPT extractor Ext such that:

$$\Pr[w \leftarrow \mathsf{Ext}_{\mathcal{A}}(1^\lambda, x)\ :\ (x, w) \in R] \geq \varepsilon - \mathsf{negl}$$

We now define the notion of polynomial witness languages.

Definition 3.4 (Witness Languages). Consider an NP language $\mathcal{L}$ and let R be its associated relation. We say that $\mathcal{L}$ is a polynomial witness language if there exists a fixed polynomial p such that for every $x \in \mathcal{L}$ it holds that there exists a size $p(|x|)$ set of witnesses w such that $w \in \{0,1\}^{\mathsf{poly}(|x|)}$ and $(x, w) \in R$.

Definition 3.5 (Witness Encryption). A witness encryption $\mathsf{WE} = (\mathsf{Enc}, \mathsf{Dec})$ for a NP language $\mathcal{L}$ consists of the following algorithms:

- **Encryption,** $\mathsf{Enc}(1^\lambda, x, m)$: On input instance x, message m and it outputs a ciphertext CT.
- **Decryption,** $\mathsf{Dec}(\mathsf{CT}, w)$: On input ciphertext CT and witness w, it outputs m'.

We require that the following properties hold:

- **Correctness**: For every $x \in \mathcal{L}$, let w be such that $(x, w) \in R$, for every $m \in \{0,1\}$,

$$\Pr[m \leftarrow \mathsf{Dec}(\mathsf{Enc}(x, m), w)] = 1$$

- **Message Indistinguishability**: For every PPT adversary $\mathcal{A}$, there is a negligible function ε, such that for every $x \notin \mathcal{L}$ the following holds:

$$\left|\Pr[1 \leftarrow \mathcal{A}(1^\lambda, \mathsf{Enc}(x, m_0))] - \Pr[1 \leftarrow \mathcal{A}(1^\lambda, \mathsf{Enc}(x, m_1))]\right| \leq \varepsilon.$$

The following theorem was shown in [19].

Theorem 3.6. *Suppose $\mathcal{L}$ is a polynomial witness language. Then, witness encryption for $\mathcal{L}$ implies extractable witness encryption for $\mathcal{L}$.*

4 MODELING THE BULLETIN BOARD

We describe briefly our modeling of the bulletin board. The bulletin board models a public ledger that lets parties publish arbitrary strings. On publishing the string on the bulletin board, the party receives a proof to establish the string was indeed published. In our setting, we model these proofs via authentication tags that can be publicly verified and the string subsequently publicly accessible. For security, we require that the authentication tags follow the standard notion of unforgeability described earlier (see definition 3.2).

In addition, the bulletin board implements a counter. Each time a string is published on the bulletin board, the counter is incremented and the authentication tag is produced on the string and counter pair. While the counter value of the bulletin board is assumed to be publicly accessible, we shall model it as an explicit query. The counter also serves as an index to the string on the bulletin board.

Hence, we model the bulletin board BB through the following queries:

- getCurrentCounter: the bulletin board returns the current value of the counter.

$$t \leftarrow \mathsf{BB}(\mathsf{getCurrentCounter}).$$

- post: on receiving value x, the bulletin board increments the counter value by 1 to t, computes the authentication tag on $(t||x)$ and responds with the tag and t to the posting party. The value and the corresponding tag can be retrieved by querying the bulletin board on t.

$$(\sigma, t) \leftarrow \mathsf{BB}(\mathsf{post}, x)$$

such that $\mathsf{Verify}_{\mathsf{BB}}(\sigma, (t||x)) = 1$.

- getContent: on receiving input t, it returns the value and the corresponding tag stored at counter value t. If t is greater than the current counter value, it returns $\perp$. Else,

$$(\sigma, x) \leftarrow \mathsf{BB}(\mathsf{getContent}, t)$$

We note that bulletin boards have previously been considered in works such as [50], but their model differs significantly from ours.

5 FAIR MPC FROM WITNESS ENCRYPTION

Overview. We start by giving an overview of our protocol. Our protocol builds on an MPC protocol that achieves the weaker notion of security with abort, where the fairness condition is not required to hold. The initial phase constitutes of the parties using this unfair MPC protocol to compute a witness encryption ciphertext of the function value they wish to compute. To decrypt, a party must post messages of a specific form (referred to as "release tokens") on to the bulletin board which the bulletin board validates with an authentication tag. The idea then is that any party can use this posted information and authentication tag to decrypt the witness encryption ciphertext. The release token must include shares of all parties that are secret prior to the completion of this initial phase. These shares must also be easily verifiable. Our construction uses

injective one-way functions, where the images of these shares are sent out during the initial phase.

The next phase, on completion of the initial phase, constitutes of parties sending these secrets to every other party. Once a party releases its share, it must not abort until it is sure that the other parties cannot post to the bulletin board, and hence decrypt the message thereafter. Otherwise, the adversary on receiving the secret shares will wait for the honest parties to abort before posting to the bulletin board. This is resolved by parameterizing the protocol by a cut-off period which once elapsed, effectively ends the protocol. If there isn't a valid post to the bulletin board at this time, no party gets the output.

To argue security, we require each statement in the language corresponding to the witness encryption to have only a polynomial size witness set. To do so, we use an injective one-way function and a unique signature scheme. The witness for the statement are the pre-images of the values sent during the initial phase, and the corresponding tag from the bulletin board. This pair is unique for a given statement. But we need to incorporate the cut-off period into the witness. This is enforced by the counter in the bulletin board as described in section 4. In the protocol, this translates to a window (set) of counter values which qualify as the additional variable in the witness. To ensure that the number of witnesses are still polynomial, the window size has to be polynomial. We parameterize the protocol with the size of this window, and the parties choose the start point of the window.

As discussed in the introduction, for the proof in this model, it is essential that the simulator is able to reset the bulletin board to a prior point (in essence, rewinding).

- rewind: This functionality is reserved for the simulator in the ideal world. On receiving additional input t, the bulletin board internally resets its counter to t and clears all data stored beyond the counter value t. The simulator gets no output on this query.

$$\perp \leftarrow \mathsf{BB}(\mathsf{rewind}, t)$$

We want to stress that this additional capability is only limited to the construction in this section and the construction in the next section (using trusted hardware) we will not require this.

We additionally discuss an extension to an optimistic phase where the parties can share some additional secrets (different from before) that enable them to decrypt a (different) ciphertext containing the output, without having to post to the bulletin board. Of course, the adversary can prematurely abort in this phase and obtain the output for itself. To protect against this, the optimistic phase is reached only once it has been established that the parties have enough information that would enable them to use the bulletin board, to decrypt to the output, in case the adversary aborts in this phase.

Construction. We now proceed to describe our protocol Π_{fair}. It uses the cryptographic primitives and a bulletin board as described below. The formal protocol description is given in Figure 1.

(1) A injective one-way functions f.
(2) An authentication scheme with public verification (Gen, Tag, $\mathsf{Verify}_{\mathsf{BB}}$) such that the authentication tags are unique for a given message.
(3) A witness encryption WE for the language

$$L_{\mathsf{WE},\Delta t} = \Big\{ \Big(\{y_i\}_{i\in[n]}, T \Big) \;\Big|\; \exists \Big(t, \sigma, \{\rho_i\}_{i\in[n]} \Big) \text{ s.t. } (\forall i \in [n],\ y_i = f(\rho_i)) \text{ AND } t \in \{T, T+1, \cdots, T+\Delta t\} \text{ AND } \mathsf{Verify}_{\mathsf{BB}}((t||\rho_1||\cdots||\rho_n), \sigma) = 1 \Big\}$$

For a given $x \in L_{\mathsf{WE},\Delta t}$, if f is an injective one-way function and (Gen, Tag, $\mathsf{Verify}_{\mathsf{BB}}$) is a scheme that generates unique authentication tags, it is easy to see that there are only $\Delta t + 1$ witnesses for x. If Δt is set to be polynomial in the size of x, there are only polynomially many witnesses for any given statement, and thus $L_{\mathsf{WE},\Delta t}$ is a polynomial witness language (see Definition 3.4). From Theorem 3.6, given $L_{\mathsf{WE},\Delta t}$ is a polynomial witness language, we know that a witness encryption for $L_{\mathsf{WE},\Delta t}$ is also an extractable witness encryption for $L_{\mathsf{WE},\Delta t}$.
(4) An MPC protocol that computes:

$$\mathcal{F}'_{\Delta t}((x_1,\rho_1,t_1),\cdots,(x_n,\rho_n,t_n)) = \Big(c, \{f(\rho_i)\}_{i\in[n]}, T\Big)$$

where $T = \max(t_1, \cdots, t_n)$ and $c = \mathsf{WE.Enc}(x_{\mathsf{WE},\Delta t}, \mathcal{F}(x_1, \cdots, x_n))$ for $x_{\mathsf{WE},\Delta t} = (\{f(\rho_i)\}_{i\in[n]}, T)$. We do not require this protocol to be fair. Importantly, we use the MPC protocol in the common random string (CRS) model. This allows for black-box simulation of the adversary without the necessity of rewinding. For this section, we shall drop the CRS notation, but it will be implicit.

Remark 1. *In the construction described above, the size of the witness encryption circuit is dependent on the number of parties in the protocol. This can be remedied by using the XOR of the ρ_i values as the release token, and applying the injective one-way function on this. The rest of the protocol remains the same.*

5.1 Proof of Security

We prove the security of our construction in the $\mathcal{F}'_{\Delta t}$-hybrid model.

Simulator $\mathcal{S}$. We start by constructing a simulator $\mathcal{S}$. Our simulator uses rewinding strategy similar to the one described in [44] (which in turn builds on [40]). The simulator has access to an ideal functionality for computing $\mathcal{F}$, and simulates $\mathcal{F}'_{\Delta t}$ for the real world adversary. In addition, for the proof in this model, the simulator reserves the right to reset the bulletin board to prior point (in essence, rewinding). (We will not require this property in the protocol based on secure hardware.) Further, $\mathcal{S}$ forwards any queries the adversary makes to the bulletin board, and returns the corresponding response from the bulletin board.

(1) $\mathcal{S}$ receives inputs $\{(x_a, \rho_a, t_a)\}_{a\in\mathcal{A}}$ sent by the adversary that are intended for $\mathcal{F}'_{\Delta t}$.
(2) Mark the current value of the counter so that $\mathcal{S}$ can rewind the bulletin board to this point.

$$t_{\mathsf{mark}} \leftarrow \mathsf{BB}(\mathsf{getCurrentCounter})$$

(3) $\mathcal{S}$ simulates the output of ideal functionality computing $\mathcal{F}'_{\Delta t}$ as follows:
   (a) Set $T = \max\{\{t_a\}_{a\in\mathcal{A}}, t_{\mathsf{mark}}\}$.
   (b) Randomly pick $\{\rho_h\}_{h\in\mathcal{H}}$ for the honest parties.

Protocol $\Pi_{\mathbf{fair}}$ in the $\mathcal{F}'_{\Delta t}$-Hybrid model

Inputs: Each party P_i has an input x_i.
Common input: The verification key for the bulletin board $\mathsf{vk}_{\mathsf{BB}}$.
The protocol:

(1) **Computation of $\mathcal{F}'_{\Delta t}$.**
- P_i samples token $\rho_i \xleftarrow{\$} \{0,1\}^{\mathsf{poly}(\lambda)}$.
- P_i queries the bulletin board to get the current counter value, i.e. $t_i \leftarrow \mathsf{BB}(\mathsf{getCurrentCounter})$.
- P_i sends (x_i, ρ_i, t_i) to the ideal functionality $\mathcal{F}'_{\Delta t}$ and receives $\left(c, \{y_i\}_{i\in[n]}, T\right)$. It aborts if it receives $\perp$ from the ideal functionality.

(2) **Exchange of tokens.** P_i broadcasts ρ_i to all other parties, and receives $\{\widehat{\rho_l}\}_{l\in[n]\setminus\{i\}}$.

(3) **Obtaining the output.** We split this into three cases, where either (i) P_i can post on the bulletin board to receive a valid witness; or (ii) P_i waits for another party to post to the bulletin board; or (iii) no party posts to the bulletin board.
- (i) P_i received ρ_j from all the other parties, such that $\forall j \in [n] \setminus \{i\} : f(\rho_j) = y_j$. In this case, P_i waits for the counter to get to T before posting to the bulletin board. Prior to posting, it check to see if another party has already posted the same. This could be done either by observing the broadcasts sent (to the bulletin board), or querying the bulletin board at most Δt times. On obtaining the appropriate authentication tag, the witness encryption can be decrypted to get the output.
- (ii) P_i received a ρ_j such that $f(\rho_j) \neq y_j$, or $\rho_j = \perp$ (i.e. a party didn't send its token). In this case, P_i checks if the right message is posted to the bulletin board for counter values between T and $T + \Delta T$. If it finds the right value, it obtains the authentication tag and decrypts the witness encryption to get the output.
- (iii) If there are no posts on the bulletin board satisfying the given requirements, and the counter has progressed beyond $T + \Delta t$, P_i aborts.

Figure 1: $\Pi_{\mathbf{fair}}$ in the $\mathcal{F}'_{\Delta t}$-Hybrid model. The protocol relies on the security of witness encryption for a polynomial witness language, injective one-way functions and authentication scheme with public verification and unique tags.

(c) $\forall i \in [n],\ y_i \coloneqq f(\rho_i)$.
(d) Compute $\widehat{\mathsf{out}} \leftarrow \mathcal{F}(\widehat{x}_1, \cdots, \widehat{x}_n)$ where $\widehat{x}_h = 0$ for all $h \in \mathcal{H}$.
(e) Set $x_{\mathsf{WE}} \coloneqq \left(\{y_i\}_{i\in[n]}, T\right)$ and compute

$$c \leftarrow \mathsf{WE.Enc}\left(x_{\mathsf{WE}}, \widehat{\mathsf{out}}\right).$$

(f) Send $(c, \{y_i\}_{i\in[n]}, T)$ to the adversarial parties.
(g) If the adversary responds with an abort, $\mathcal{S}$ sends abort to the ideal functionality computing $\mathcal{F}$, and exits. For our analysis, we denote this by abort_1.

(4) $\mathcal{S}$ sends values $\{\rho_h\}_{h\in\mathcal{H}}$ to the adversary. If the adversary sends values $\{\rho_a\}_{a\in\mathcal{A}}$ such that $\forall a, y_a = f(\rho_a)$; or sends a post query to the bulletin board with value $(\rho_1||\cdots||\rho_n)$ when counter value is between T and $T+\Delta t$ such that $\forall i, y_i = f(\rho_i)$, the adversary has not aborted.

(5) If the adversary aborted in the previous step, $\mathcal{S}$ sends abort to the ideal functionality computing $\mathcal{F}$, and exits. For our analysis, we denote this by abort_2.

(6) If the adversary didn't abort prior to this, we need to estimate the probability of the adversary not aborting. Let q represents the true of probability of this event, where the randomness is over random coins used in step 3(b) and 3(e). The estimated probability will be denoted by $\widetilde{q}$.
(a) $\mathcal{S}$ fixes some number $t = \mathsf{poly}(\lambda)$.
(b) $\mathcal{S}$ rewinds the adversary to step 3, rewinds the bulletin board $\mathsf{BB}(\mathsf{rewind}, t_{\mathsf{mark}})$ and repeats steps 3 and 4 (other than 3(g)) with fresh randomness each time. Repeat till the adversary has not aborted t times.
(c) $\mathcal{S}$ estimates q as $\widetilde{q} = \frac{t}{\#\text{ of repetitions}}$. The polynomial defining t is chosen to be large enough that

$$\Pr\left[\frac{1}{2} \leq \frac{q}{\widetilde{q}} \leq 2\right] > 1 - 2^{\lambda}.$$

(7) The simulator sends $\{x_a\}_{a\in\mathcal{A}}$ to the ideal functionality for $\mathcal{F}$ and receives out. $\mathcal{S}$ repeats the following at most $\frac{t}{\widetilde{q}}$ times.
(a) With fresh randomness each time, $\mathcal{S}$ rewinds the adversary to step 3, rewinds the bulletin board $\mathsf{BB}(\mathsf{rewind}, t_{\mathsf{mark}})$ and repeats steps 3 and 4 (other than 3(g)) replacing $\widehat{\mathsf{out}}$ with out.
(b) If the adversary does not abort, we output its view and the simulator terminates.

(8) If $\mathcal{S}$ has not terminated yet, output fail and terminate the simulation.

Claim 1. *If simulator $\mathcal{S}$ does not outputs* fail*, the hybrid world and the ideal world are indistinguishable.*

Proof. We split the analysis into two cases:

- *Case 1: The adversary does not abort.* Since the simulator does not output fail, it has successfully got the adversary to accept the transcript for the right output. In this case, the main thread of the adversary is statistically indistinguishable from the real execution. Additionally, since the simulator is able to rewind the bulletin board, the adversary's view of the bulletin board is that of a straight line execution. Thus the joint distribution consisting of the view of the adversary and the honest party outputs is indistinguishable.

– *Case 2: The adversary aborts.* As noted in the simulator, the adversary can abort in two phases of the protocol. We deal with the two case separately:

  abort_1: The adversary aborts immediately after running the MPC for $\mathcal{F}'_{\Delta t}$. In both the real and ideal world, honest parties do not get any output. Thus, we need to argue that the adversary's view is indistinguishable when he receives a witness encryption of the actual output as opposed to when he received the witness encryption of a random string. To the contrary, assume that the adversary can distinguish between these two cases. Since there only polynomially many witnesses, we use the extractor for the adversary to recover the witness. Since the honest parties aborts without revealing its share of the token, we can use the extractor to construct an adversary that breaks the security of the injective one-way function.

  abort_2: The adversary aborts on receiving the honest party's tokens without posting to the bulletin board. We use the same technique as above, leveraging the extractor for witness encryption, to construct an adversary that breaks the unforgability of the authentication tags issued by the bulletin board.

□

Claim 2. *The simulator $\mathcal{S}$ outputs* fail *with only negligible probability.*

The proof can be found in Appendix A.;

We assume that the value of T chosen in the protocol is such that the real execution of the protocol ends in time bounded above by a polynomial $g(\lambda)$. Otherwise $\mathcal{F}'_{\Delta t}$ implements an additional check to ensure this.

Claim 3. *The simulator $\mathcal{S}$ runs in expected polynomial time.*

Proof. With probability $1-q$, the simulator aborts prior to step 6. With probability q, the simulator goes through the estimation phase and then attempts to force an accepting transcript onto the adversary. The expected number of iterations for the estimation phase is $\frac{t}{q}$ and the cut-off point for forcing the transcript is $\frac{t}{q} < \frac{t}{2q}$. Hence the total expected running time is bounded by

$$g(\lambda) \cdot q \cdot \left(\frac{t}{q} + \frac{2t}{q}\right) = g'(\lambda)$$

Thus $\mathcal{S}$ runs in expected polynomial time. □

Given the above claims, the following theorem follows.

Theorem 5.1. *Assuming the security of injective one-way functions, witness encryption for polynomial witness language and the unforgeability of the authentication scheme, the above protocol satisfies Definition 3.1 in the $\mathcal{F}'_{\Delta t}$-hybrid model.*

As mentioned in the introduction, in order to rely on the standard security of WE, we require the bulletin board's proof of publish to be implemented via unique signatures [43, 56]. If the bulletin board is implemented via standard signatures (e.g., in Google Transparency Certificates) or proofs of stake (e.g., in Etherium), then we require the stronger assumption of extractable witness encryption [19].

6 FAIRNESS FROM SECURE HARDWARE

A key limitation of our previous constructions is the need to use Witness Encryption (WE) to protect the output of the MPC protocol. Unfortunately, current proposed WE construction are inefficient, due to the high overhead of current constructions of multilinear maps. Moreover, the Witness Encryption paradigm requires the parties to compute a new WE ciphertext for each invocation of the MPC protocol.

In this section we investigate an alternative paradigm that uses *secure hardware*. Our work is motivated by the recent deployment of commodity virtualization technologies such as Intel's Software Guard Extensions (SGX). These technologies allow for the deployment of secure "enclave" functionalities that can store secrets and perform correct computation even when executed in an adversarial environment. Moreover, these systems allow an enclave to remotely *attest* to their correct functioning, which allows for the establishment of trustworthy communications between enclaves running on different machines.

Model Following the approach of Pass *et al.* [59] we model all available trusted hardware processors from a given manufacturer as a single, globally shared functionality denoted $\mathcal{G}_{\text{att}}$ (see Figure 5). We describe the functionalities required for our construction, and refer the reader to [59] for details. `install` loads the program prog onto the attested hardware. It returns an enclave identifier *eid*. (For simplicity, we skip the session identifier used in [59].) The enclave identifier is be used to identify the enclave upon resume. `resume` allows for a stateful resume using the unique enclave identifier generated. On running over a given input, the output produced is signed to attest that the enclave with identifier *eid* was installed with a program prog, which was then executed to produce the output. The program's input is not included in the attestation.

Description. We describe here the main ideas in this construction that differ from the previous construction. Upon loading the program onto the attested hardware (enclaves), there is an initial key exchange to establish a secure authenticated channel between the enclaves. Any information passed over this channel is hidden from the parties. It is important that enclaves attest to the fact that they are running the correct programs prior to the key exchange.

Next, the shares of the release token and the key are input to the enclave. The enclaves use the established secure authenticated channel to exchange this information and set up consistent parameters (over all enclaves) for the decryption circuit. The parties then run an MPC protocol external to the enclaves to compute an encrypted version of the output. As in our previous construction, the players exchange shares of the release token that they are required to post onto the bulletin board in order to decrypt.

For technical reasons, we need to ensure that the key share that a party sends to the enclave is the one used in the MPC. This is ensured by using a commitment scheme which the MPC computation verifier before returning the output.

Our protocol makes requires the following primitives:

(1) A one-way function f.[6]
(2) A signature schemes (Gen, Sign, Verify).

[6] In practice we suggest using a hash function.

(3) An authentication scheme with public verification (Gen, Tag, $\mathsf{Verify}_{\mathsf{BB}}$).
(4) A multi-party computation in the CRS model for computing $\mathcal{F}'$ defined as

$$\mathcal{F}'\left(\left\{x_i, k_i, \{\mathsf{com}_{ij}\}_{j\in[n]}, r_i\right\}_{i\in[n]}\right) =$$
$$\begin{cases} \perp & \text{if } \exists i, i' \text{ s.t. } \left(\{\mathsf{com}_{ij}\}_{j\in[n]}\right) \neq \left(\{\mathsf{com}_{i'j}\}_{j\in[n]}\right) \\ \perp & \text{if } \exists i \text{ s.t. } \mathsf{com}_{ii} \neq \mathsf{Com}(k_i; r_i) \\ y & \text{otherwise} \end{cases}$$

where $y = \mathsf{AE.Enc}_{\bigoplus_{i=1}^{n} k_i}(\mathcal{F}(x_1, \cdots, x_n))$. Essentially the MPC takes in a the input, key share, a commitment tuple and a decommitment from each party. It checks if the tuple pairs received are the same throughout and the commitment linked to each party decommits to the key share. If this check fails it just returns $\perp$, or it returns the output y.
(5) Two instances of AE scheme (Enc, Dec) with INT-CTXT security for authentication and semantic security.
(6) A commitment scheme (Com) with computational hiding and statistically binding.

We describe and prove the protocol in the two party setting. Both extended naturally to the multi-party setting. The protocol is described in Figure 2.

We note that there are two trapdoors installed into functionalities of $\mathsf{prog}_{\mathsf{fair}}$. These are used for the security reduction of the one-way function, and to program the output correspondingly. Specifically, the trapdoor is used to get the enclave to attest to a value of choice. These trapdoors can be used by an adversarial party, but this makes no difference to the security since these values are not sent across to the other party.

Theorem 6.1. *Assume that F is one-way, the signature scheme is existentially unforgeable under chosen message attacks, the authentication scheme satisfies standard notion of unforgeability, the encryption scheme is perfectly correct, authenticated encryption scheme that is perfectly correct and satisfies standard notions of INT-CTXT and semantic security, decisional Diffie-Hellman assumption holds in the algebraic group adopted. Then, the above protocol satisfies Definition 3.1 in the $(\mathcal{G}_{\mathsf{att}}, \mathcal{F}')$-hybrid model.*

The proof can be found in Appendix A.2.

7 INSTANTIATING THE BULLETIN BOARD

Our proposed paradigm relies on a *verifiable* public bulletin board that makes three guarantees about entries posted to it:

- The entry's presence can be cryptographically verified using a public operation.
- Once posted, the entry is available to all parties.
- Entries are assigned a unique monotonically increasing sequence number.

We now consider several existing techniques that we can leverage to obtain such a bulletin board.

Certificate Transparency Logs. Certificate Transparency (CT) [1] is a public audit log operated by a coalition of browser vendors and certificate authorities. CT allows individual certificate authorities to post newly-issued certificates to a public log. These entries are then (1) signed by the log maintainer, and (2) added to a Merkle hash tree. The root of the hash tree is also signed by (one or more) log maintainers and published to the world.

A collection of users known as *monitors* can access the CT log to view the contents of certificates. While the CT log is itself not fundamentally tamper resistant – since the servers operating it can remove portions and/or be disabled by remote network attacks – any tampering is detectable due to the structure of the Merkle hash tree. The location of the entry within the Merkle hash tree also serves to act as a proxy for a monotonically increasing sequence number.

Under the assumption that the existing CT logs are reliable and trustworthy, we can use CT to build fairness systems by entombing the required public data into a component of an X.509 certificate signing request and requesting the certificate from a free certificate authority such as LetsEncrypt [2]. Because LetsEncrypt submits all certificates to a public log[7] it is possible for any party to recover these certificates and verify a cryptographic proof that the entries have been published.

Public blockchains. Crypto-currencies such as Bitcoin or Etherium rely on a publicly available data structure called a *blockchain*. Blockchains are an append-only ledger that is maintained by an ad-hoc group of network peers. Blockchains come in two basic types. The first type use computational *proofs of work* to determine which peer should be allowed to add a new block of transactions to the blockchain. Clients accept the longest chain that contains well formed transactions; as a result the system is secure as long as a supermajority of the computational power in the network is controlled by honest peers. This approach is tolerant of churn, and thus we need not pick a set of honest parties in advance.

An alternative approach uses *proof of stake* [16]. In these systems a quorum of peers is sampled from the network with probability proportional to the fraction of monetary holdings controlled by each peer. This quorum is responsible for producing the next block and selecting the next quorum by the same mechanism. The peers authenticate the resulting block by signing it using a secure digital signature scheme. The security assumption here assumes that the parties with the largest share of the cryptocurrency have a vested interest in keeping it running. Proof of stake systems are in their infancy both in terms of deployment and theory. However, they provide an interesting middle ground between the costs of a pure proof of work approach and the challenges with selecting a set of trusted parties a priori to maintain the bulletin board.

8 IMPLEMENTATION

In this section we present an implementation of the protocol given in Section 6, and show that the protocol is efficient. Our implementation consists of three major pieces: the bulletin board instantiated using Bitcoin, the MPC protocol instantiated using the SPDZ framework [3, 29], and a "witness decryptor" instantiated using an Intel SGX secure enclave. We describe each component in more detail below.

[7] See https://crt.sh/?id=15707024

$\text{prog}_{\text{fair}}[\Delta t, P_0, P_1, \text{vk}_{\text{BB}}, i]$ where $i \in \{0,1\}$ //for party P_i

On input ("keyex"):
Let $a \xleftarrow{\$} \mathbb{Z}_p$, and return g^a

On input ("init", k, t, ρ, r):
let $k_i := k$, $t_i := t$, $\rho_i := \rho_i$
$\text{com}_i := \text{Com}(k_i; r)$, return com_i

On input ("send", g^b):
let $\text{sk} = \left(g^b\right)^a$, $\text{ct}_i \leftarrow \text{AE.Enc}_{\text{sk}}(k_i, t_i, \rho_i, \text{com}_i)$
return ct_i

On input ("receive", ct'_{1-i}):
assert "init" and "send" have been called.
$(k_{1-i}, t_{1-i}, \rho_{1-i}, \text{com}_{1-i}) := \text{AE.Dec}_{\text{sk}}\left(\text{ct}'_{1-i}\right)$
return com_{1-i}

On input ("getParams", v):
assert "init", "send" and "receive" have been called
$T := \max\{t_0, t_1\}$, $y := f(\rho_0 \oplus \rho_1)$, $K := k_0 \oplus k_1$
if $v \neq \perp$, return v, else return (T, y)

On input ("output", ct_{MPC}, t, ρ, σ, v):
if $v \neq \perp$, return v
assert "getParams" has been called
assert $t \in \{T, T+1, \cdots, T+\Delta t\}$
assert $f(\rho) = y$
assert $\text{Ver}_{\text{BB}}(t, \rho, \sigma)$
return $\text{AE.Dec}_K(\text{ct}_{\text{MPC}})$

$\text{Prot}_{\text{fair}}[\mathcal{F}, \Delta t, P_0, P_1, \text{vk}_{\text{BB}}, i]$ where $i \in \{0,1\}$ //for party P_i, $P_j = P_{1-i}$

Input: x_i

Protocol:

(1) let $eid_i \leftarrow \mathcal{G}_{\text{att}}.\texttt{install}(\text{prog}_{\text{fair}}[\Delta t, P_0, P_1, \text{vk}_{\text{BB}}, i])$.
(2) Initiate the key exchange procedure. Let $(g^a, \sigma_i) \leftarrow \mathcal{G}_{\text{att}}.\texttt{resume}(eid_i, \text{"keyex"})$. Send (eid_i, g^a, σ_i) to P_j, await $\left(eid_j, g^b, \sigma_j\right)$ from P_j. Check if $\text{Ver}_{\text{mpk}}((eid_j, \text{prog}_{\text{fair}}[\Delta t, P_0, P_1, \text{vk}_{\text{BB}}, j], g^b), \sigma_j) = 1$, else abort.
(3) $k_i \xleftarrow{\$} \{0,1\}^\lambda$, $\rho_i \xleftarrow{\$} \{0,1\}^\lambda$, $r_i \xleftarrow{\$} \{0,1\}^\lambda$, $t_i \leftarrow \text{BB}(\text{getCurrentCounter})$.
(4) Initialize the enclave with the values obtained in the previous step. $(\text{com}_i, \_) := \mathcal{G}_{\text{att}}.\texttt{resume}(eid_i, \text{"init"}, \rho_i, k_i, t_i, r_i)$.
(5) Set up the exchange of information between enclaves. Let $(\text{ct}_i, \_) \leftarrow \mathcal{G}_{\text{att}}.\texttt{resume}(eid_i, \text{"send"})$. Send ct_i to P_j and wait for ct_j in response. On receiving ct_j, send it to the enclave $(\text{com}_{1-i}, \_) \leftarrow \mathcal{G}_{\text{att}}.\texttt{resume}(eid_i, \text{"receive"}, \text{ct}_j)$. At this point, both commitments are available to the party.
(6) Get parameters for the MPC computation, $(T, y) \leftarrow \mathcal{G}_{\text{att}}.\texttt{resume}(eid_i, \text{"getParams"}, \perp)$
(7) Send $(x_i, k_i, \text{com}_0, \text{com}_1, r_i)$ to $\mathcal{F}'$ and receive ct_{MPC}. Abort if $\perp$ is received. // ct_{MPC} of the form $\text{AE.Enc}_K(\mathcal{F}(x_0, x_1))$
(8) Send token share ρ_i to P_j and wait for token share ρ_j.
- If ρ_j not received, or $f(\rho_0 \oplus \rho_1) \neq y$, then wait to see if P_j posts the right value on the bulletin board, when the counter is between T and $T + \Delta t$. If the counter goes beyond $T + \Delta t$, and no such value posted, abort. If the right value is posted at counter t_{BB}, $(\sigma_{\text{BB}}, \rho) \leftarrow \text{BB}(\text{getContent}, t_{\text{BB}})$.
- ρ_j received and $f(\rho_0 \oplus \rho_1) = y$. Wait for the counter value to get to T, and then post $\rho_0 \oplus \rho_1$ on the bulletin board to get the corresponding authentication tag, i.e. $(\sigma_{\text{BB}}, t_{\text{BB}}) \leftarrow \text{BB}(\text{post}, \rho_0 \oplus \rho_1)$.
- Output $\mathcal{G}_{\text{att}}.\texttt{resume}(eid_i, \text{"output"}, t_{\text{BB}}, \rho_0 \oplus \rho_1, \sigma_{\text{BB}}, \perp)$.

Figure 2: Two party fair protocol $\text{Prot}_{\text{fair}}$ in the $(\mathcal{G}_{\text{att}}, \mathcal{F}')$-hybrid model.

	Initialization	Decrypt
mean	1.180 ± 0.112	0.039 ± 0.001
mean	0.002 ± 0.000	0.037 ± 0.001

Table 1: Performance of SGX enclave setup and decryption (not MPC). Average and standard deviation of 500 runs.

Bitcoin as a bulletin board. For our prototype implementation we use the Bitcoin network, which supports a limited scripting system called Bitcoin Script. In Bitcoin each transaction contains a script that is evaluated to ensure the transaction is authorized. This scripting system supports an instruction named `OP_RETURN`, which allows the sender of a transaction to embed up to 40 bytes of arbitrary data into a transaction that is transmitted for inclusion in the Bitcoin blockchain. Each block of transactions in the blockchain contains a computational proof of work (PoW) that is computed by the network. This proof is bound cryptographically to all of the transactions within a block, as well as to the hash of the previous block. At current network difficulty, computing a proof of work for a single block requires an expected 2^{64} invocations of the SHA2 hash function on the standard Bitcoin network. To verify publication on the bulletin board, our implementation requires a fragment consisting of six consecutive blocks (where the transaction is located in the first block of the fragment). The cost of forging such a fragment scales linearly in the number of blocks required.

We note that the use of a computational proof of work bulletin board provides somewhat different fairness properties than a signature-based bulletin board, *e.g.,* Certificate Transparency or a proof-of-stake blockchain. Specifically, in this setting an attacker with sufficient time or computational power can always "forge" a satisfying chain of blocks, and use this private result as a witness to enable decryption. Such an attack would be economically costly, since the corresponding effort – if applied to crypto-currency mining – would be worth a substantial sum of money.[8] However, we can further restrict this attacker by employing a trusted clock within the witness decryptor (*e.g.,* Intel SGX) [9]. This optimization requires the attacker to complete the forgery within a pre-defined time limit that approximates the expected time for the full Bitcoin network. Thus a successful attacker must possess most of the available hashpower of the Bitcoin network (which currently approximates the electrical consumption of Turkmenistan).

For our experiments, we use the public Bitcoin testnet. The Bitcoin testnet functions similarly to the main Bitcoin network, but uses a zero-value currency and a low difficulty setting for the proof of work. We selected testnet for our experiments mainly because blocks are mined extremely rapidly and transactions require no monetary expenditure for "transaction fees". However our code can use the production Bitcoin blockchain without any code changes.

MPC Protocol. Our protocol can be used to extend any MPC scheme that supports efficient symmetric encryption. We note that one could employ Intel SGX directly to perform a naive form of MPC. However, our goal in this work is to demonstrate that our approach works efficiently even when instantiated with a "cryptographic" MPC protocol.[10]

Thus for our implementation we use the SPDZ-2 framework developed by the University of Bristol [3]. SPDZ-2 is designed to tolerate dishonest majorities during computation. In SPDZ circuits are designed in python and then compiled down into a circuit structure. The computation is done in two phases: an offline phase that does not require the computation inputs and an online phase that performs the actual computation. In order to optimize the running time of the online phase, the pre-computation and compilation phases are relatively more time consuming.

The maintainers of SPDZ-2 have implemented the AES-128 cipher in order to benchmark its efficiency. We repurpose this code to build a simple authenticated encryption system for that uses 3 rounds of AES to encrypt and authenticate one 128-bit block of data output from the computation. The encryption scheme takes as input each party's private computation input x_i and keyshare k_i. It computes as output a ciphertext C encrypted under msk. We also use this AES-128 cipher to implement a commitment scheme. The randomness of the commitment scheme is used as the key to the cipher, with the commitment message as the plaintext.

We construct an MPC circuit for SPDZ-2 that takes in a private input x_i, a keyshare k_i, a randomness share r_i, and commitment to the master key com(msk; r). The first circuit computes the output of the desired MPC functionality $f(x_1, \ldots, x_N)$. Next it computes $r = \oplus_{1 \le i \le N} r_i$ and opens the commitment. It compares $\oplus_{1 \le i \le N} k_i$ with the msk from the commitment. If they do not match, sets $f(x_1, \ldots, x_N) = 0$. Finally, the circuit computes the encryption of $f(x_1, \ldots, x_N)$ using msk and outputs the final ciphertext.

SGX as Witness Decryptor. Intel's SGX is a set of extensions to the x86 instruction set that allows for code to be executed in a protected enclave. SGX programs are segmented into two pieces: an untrusted *application* and a trusted *enclave*. The application consists of standard software running on a standard operating system and we assume that it may behave maliciously if the i^{th} player is corrupted. Code within an enclave is verified upon startup and isolated from inspection and tampering, even from an adversary that controls the system's operating system. The root of trust of an SGX enclave is the Intel processor, which enforces the enclave's isolation. It is worth noting that the code run within an enclave is not private; however secrets may be generated or retrieved after the enclave is initialized. Note that the enclave has no direct access to network communications, and must rely on the untrusted part of the application.[11]

We adapt an existing SGX-bitcoin client called Obscuro [4] to perform the role of the Witness Decryptor. This enclave is instantiated by each of the N parties participating in the protocol. A single *master* instance of the enclave uses the `sgx_read_rand` function, supplied by the SGX environment, to generate an AES master key msk that will eventually encrypt the output of the MPC circuit. Additionally, the *master* enclave generates a random 320-bit release token t that must be verifiably posted to a bulletin board before the ciphertext can be decrypted. Next, the master applies a secret sharing scheme to derive secret shares $(k_1, \ldots, k_N)$ of msk and $(t_1, \ldots, t_N)$ of t. Finally, the master computes a commitment com(msk; r) and secret shares the randomness into $r_1 \ldots r_N$. Now for $i = 1$ to N it distributes the tuple $(k_i, t_i, \text{msk}, \text{t}, \text{com}(\text{msk}; r), r_i)$ to the i^{th} enclave via a secure channel.[12]

Once all secrets have been distributed by the master enclave, the channels are closed and each enclave outputs its key share k_i to the application. The users now invokes SPDZ to conduct the MPC protocol, using as its private inputs x_i, k_i, r_i, and com(msk; r). If the MPC protocol does not complete successfully, the application aborts and a full restart is required. Otherwise, the application obtains a ciphertext C output by the MPC protocol and provides this as input to the enclave. The enclave attempts to decrypt the ciphertext under msk and if and only if this decryption check completes successfully (and the result is the proper format and length), it releases t_i, which the application then transmits to all of the remaining parties.

To access the encrypted output of the MPC, at least one party must re-compute the release token as $\text{t} = (t_1 \oplus \cdots \oplus t_N)$ and post

[8] At present rates as of July 2017, this opportunity cost is approximately $28,000 per block forged.

[9] Correctly accessing trusted time from within an enclave is part of the Intel SGX specification, but it is not yet supported as it relies on platform services which are not active. In our implementation, we include code to properly access trusted time, but do not include it in our measurements because of the lack of support.

[10] Additionally, we remark that if SGX is used to implement the MPC protocol itself, a security breach of the SGX system will result in the loss of all security properties provided by the MPC. On the other hand, if we employ a cryptographic MPC protocol, then a failure of Intel's SGX risks only the fairness property. We view this as a benefit of our approach.

[11] This enables the application to censor or tamper with communications between the application and the network.

[12] SGX supports the creation of authenticated, secure channels using attestation and DHKE.

this value to the Bitcoin network inside of a transaction. Each user's application monitors the Bitcoin network using RPC calls to a local Bitcoin client `bitcoincli` which is running on the user's machine. This userland code then feeds the resulting blockchain fragment (which consists of six consecutive blocks) back to the enclave, which confirms that the release token matches its stored value t, and also verifies the proofs of work on each block. While an adversarial user can block this response, they are unable to falsify or tamper with it due to the fact that such tampering would require an impractical amount of computation. The application also supplies the enclave with the output of the MPC C.

If all verifications succeed, the enclave decrypts the ciphertext C using an authenticated encryption scheme under msk, and outputs the resulting plaintext.

Optimizations. In a bid to optimize the implementation, there are a few differences from the described protocol in section 6. They do not make a difference to the security of the protocol, and are briefly described here:

- Instead of each party generating its key and token shares, a designated master enclave chooses them and distributes them to the other enclaves.
- Instead of a commitment for each share of the key, there is only a single "master commitment" of the key.

Sample computation and performance. For proof of concept, we implemented a search program that takes as input a search value x from one party and a list $(y_1, \ldots, y_n)$, from the other party. These circuits each calculate an integer output M and encrypt the result as $\mathsf{Enc}(\bigoplus_1^n k_i; M)$. Since these are two-party functions we tested with $N = 2$ and $n = (100, 500, 1000)$.

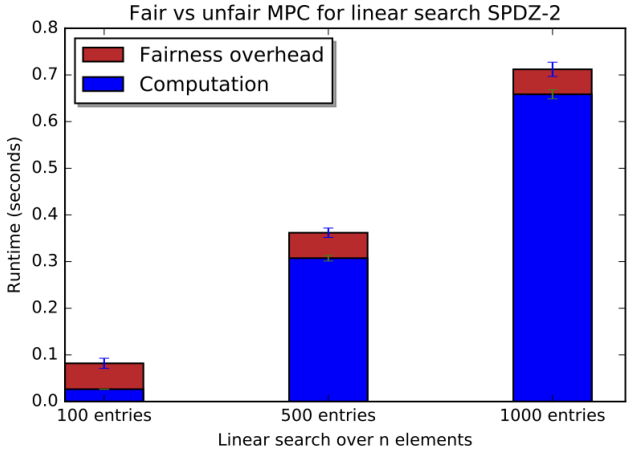

Figure 3: Mean runtimes for a linear search on n items using SPDZ taken over 50 iterations. Only the online portion of the MPC is shown. In blue, we show the cost of running the search without any provision for fairness. In red, the overhead from AES encryption needed for fairness.

Cost of fairness in the MPC. Our implementation demonstrates that our approach can be used add fairness to MPC schemes efficiently using current technology. We recall that fairness in MPC is particularly important when the output of the MPC is extremely valuable. While adding three rounds of AES to a simple MPC scheme represents a high cost, it adds a only a negligible cost when considering more time consuming computations. In Figure 3 we show the average runtime over 50 trials of a number of different circuits in SPDZ-2. The cost of encryption is clearly dwarfed by large search problems and set intersection.

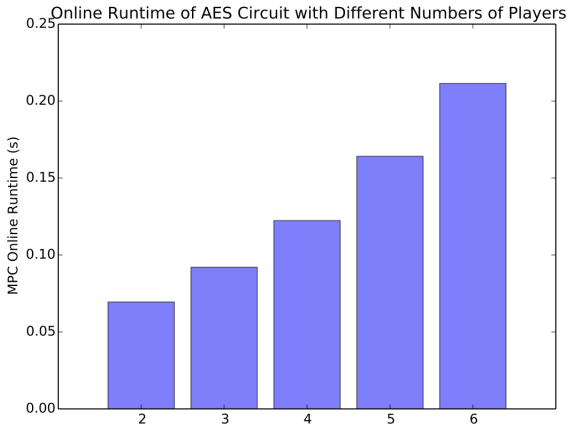

Figure 4: Mean runtimes for our AES circuit varied over the number of players participating. Only the online portion of the MPC is shown. This circuit is dominated by 3 AES operations.

While we ran the MPC experiments with N=2 players, SPDZ-2 allows computations with more players. In Figure 4 we consider *only the cost of running the encryption component* of the MPC protocol with higher numbers of players. Because each player contributes a key share, the cost of running the protocol increases with each player. While the runtime of the encryption operation does increase, we note that it is still adds only a fraction of one second of online computation time up to $N = 6$.

SGX Runtime. Intel SGX offers an extremely efficient method of trusted program execution. We benchmark our SGX Enclave over 500 trials of the two party protocol for some fixed parameters. We run our test on an Intel i5-6600K 3.5GHz processor with 16 GB of RAM running Ubuntu 14.04 and SGXSDK-1.7, running both the master and minion on the same hardware. For the purpose of benchmarking, we hardcode into the master enclave the master AES key and fix the release token to be the results of an OP_RETURN instruction in a known block of the Bitcoin Testnet. Additionally, we run the MPC protocol once to generate a valid ciphertext. With the pre-fixed values, we can effectively check the running time of the various parts of the enclave's execution. All key exchange and interaction with the `bitcoincli` is still run as in the real protocol. In Table 1 we show the average running times of the various segments of the enclave, both for the master instance and minion instance. For the minion's execution time, we pause the timer while it is waiting for the minion to open a network connection. It is clear that the slowest piece of the program is the enclave

initialization. This is because the enclave must provision all memory that it may require from the SGX driver during initialization. Our implementation allocates more memory than it will use to be conservative.

9 ACKNOWLEDGMENTS

This research was supported in part by the National Science Foundation under awards CNS-1010928, CNS-1228443, CNS-1653110, CNS-1414023 and EFMA-1441209; The Office of Naval Research under contract N00014-14-1-0333; DARPA/ARL Safeware Grant W911NF-15-C-0213; and the Mozilla Foundation.

REFERENCES

[1] 2017. Certificate Transparency. Available at https://www.certificate-transparency.org/. (2017).
[2] 2017. Let's Encrypt. Available at https://letsencrypt.org/. (2017).
[3] 2017. Multiparty computation with SPDZ online phase and MASCOT offline phase. Github. (2017). https://github.com/bristolcrypto/SPDZ-2.
[4] 2017. Obscuro. Github. (2017). https://github.com/BitObscuro/Obscuro.
[5] Bar Alon and Eran Omri. 2016. Almost-Optimally Fair Multiparty Coin-Tossing with Nearly Three-Quarters Malicious. In *TCC, Part I*. 307–335.
[6] Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Lukasz Mazurek. 2014. Secure Multiparty Computations on Bitcoin. In *IEEE Symposium on Security and Privacy*. 443–458.
[7] Gilad Asharov. 2014. Towards Characterizing Complete Fairness in Secure Two-Party Computation. In *TCC*. 291–316.
[8] Gilad Asharov, Amos Beimel, Nikolaos Makriyannis, and Eran Omri. 2015. Complete Characterization of Fairness in Secure Two-Party Computation of Boolean Functions. In *TCC, Part I*. 199–228.
[9] Gilad Asharov, Yehuda Lindell, and Tal Rabin. 2013. A Full Characterization of Functions that Imply Fair Coin Tossing and Ramifications to Fairness. In *TCC*. 243–262.
[10] Gilad Asharov, Yehuda Lindell, and Hila Zarosim. 2013. Fair and Efficient Secure Multiparty Computation with Reputation Systems. In *ASIACRYPT*. 201–220.
[11] N. Asokan, Matthias Schunter, and Michael Waidner. 1997. Optimistic Protocols for Fair Exchange. In *CCS '97, Proceedings of the 4th ACM Conference on Computer and Communications Security, Zurich, Switzerland, April 1-4, 1997*. 7–17.
[12] N. Asokan, Victor Shoup, and Michael Waidner. 1998. Optimistic Fair Exchange of Digital Signatures (Extended Abstract). In *EUROCRYPT*. 591–606.
[13] Donald Beaver and Shafi Goldwasser. 1989. Multiparty Computation with Faulty Majority. In *CRYPTO*. 589–590.
[14] Amos Beimel, Yehuda Lindell, Eran Omri, and Ilan Orlov. 2011. $1/p$-Secure Multiparty Computation without Honest Majority and the Best of Both Worlds. In *CRYPTO*. 277–296.
[15] Michael Ben-Or, Oded Goldreich, Silvio Micali, and Ronald L. Rivest. 1985. A Fair Protocol for Signing Contracts (Extended Abstract). In *ICALP*. 43–52.
[16] Iddo Bentov, Ariel Gabizon, and Alex Mizrahi. 2016. Cryptocurrencies without proof of work. In *International Conference on Financial Cryptography and Data Security*. Springer, 142–157.
[17] Iddo Bentov and Ranjit Kumaresan. 2014. How to Use Bitcoin to Design Fair Protocols. In *CRYPTO*. 421–439.
[18] Dan Boneh and Moni Naor. 2000. Timed Commitments. In *CRYPTO*. 236–254.
[19] Elette Boyle, Kai-Min Chung, and Rafael Pass. 2014. On Extractability Obfuscation. In *TCC*. 52–73.
[20] Christian Cachin and Jan Camenisch. 2000. Optimistic Fair Secure Computation. In *CRYPTO*. 93–111.
[21] Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. 2007. Universally Composable Security with Global Setup. In *Theory of Cryptography, 4th Theory of Cryptography Conference, TCC 2007, Amsterdam, The Netherlands, February 21-24, 2007, Proceedings*. 61–85.
[22] Ran Canetti, Abhishek Jain, and Alessandra Scafuro. 2014. Practical UC security with a Global Random Oracle. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*. 597–608.
[23] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. 2002. Universally composable two-party and multi-party secure computation. In *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*. 494–503.
[24] Liqun Chen, Caroline Kudla, and Kenneth G. Paterson. 2004. Concurrent Signatures. In *EUROCRYPT*. 287–305.
[25] Richard Cleve. 1986. Limits on the Security of Coin Flips when Half the Processors Are Faulty (Extended Abstract). In *STOC*. 364–369.
[26] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. 2013. Practical Multilinear Maps over the Integers. In *CRYPTO*. 476–493.
[27] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. 2015. New Multilinear Maps Over the Integers. In *CRYPTO*. 267–286.
[28] Ronald Cramer and Victor Shoup. 1998. A Practical Public Key Cryptosystem Provably Secure Against Adaptive Chosen Ciphertext Attack. In *Advances in Cryptology - CRYPTO '98, 18th Annual International Cryptology Conference, Santa Barbara, California, USA, August 23-27, 1998, Proceedings*. 13–25.
[29] Ivan Damgard, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. 2012. Practical Covertly Secure MPC for Dishonest Majority – or: Breaking the SPDZ Limits. Cryptology ePrint Archive, Report 2012/642. (2012). http://eprint.iacr.org/2012/642.
[30] Yevgeniy Dodis, Pil Joong Lee, and Dae Hyun Yum. 2007. Optimistic Fair Exchange in a Multi-user Setting. In *Public Key Cryptography - PKC 2007, 10th International Conference on Practice and Theory in Public-Key Cryptography, Beijing, China, April 16-20, 2007, Proceedings*. 118–133.
[31] Shimon Even, Oded Goldreich, and Abraham Lempel. 1985. A Randomized Protocol for Signing Contracts. *Commun. ACM* 28, 6 (1985), 637–647.
[32] Juan A. Garay and Markus Jakobsson. 2002. Timed Release of Standard Digital Signatures. In *Financial Cryptography*. 168–182.
[33] Juan A. Garay, Markus Jakobsson, and Philip D. MacKenzie. 1999. Abuse-Free Optimistic Contract Signing. In *CRYPTO*. 449–466.
[34] Juan A. Garay, Philip D. MacKenzie, Manoj Prabhakaran, and Ke Yang. 2006. Resource Fairness and Composability of Cryptographic Protocols. In *TCC*. 404–428.
[35] Juan A. Garay and Carl Pomerance. 2003. Timed Fair Exchange of Standard Signatures: [Extended Abstract]. In *Financial Cryptography, 7th International Conference, FC 2003, Guadeloupe, French West Indies, January 27-30, 2003, Revised Papers*. 190–207.
[36] Sanjam Garg, Craig Gentry, and Shai Halevi. 2013. Candidate Multilinear Maps from Ideal Lattices. In *EUROCRYPT*. 1–17.
[37] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. 2013. Witness Encryption and its Applications. Cryptology ePrint Archive, Report 2013/258. (2013). http://eprint.iacr.org/2013/258.
[38] Craig Gentry, Sergey Gorbunov, and Shai Halevi. 2015. Graph-Induced Multilinear Maps from Lattices. In *TCC, Part II*. 498–527.
[39] Craig Gentry, Allison B. Lewko, and Brent Waters. 2014. Witness Encryption from Instance Independent Assumptions. In *CRYPTO*. 426–443.
[40] Oded Goldreich and Ariel Kahan. 1996. How to Construct Constant-Round Zero-Knowledge Proof Systems for NP. *J. Cryptology* 9, 3 (1996), 167–190.
[41] Oded Goldreich, Silvio Micali, and Avi Wigderson. 1987. How to play any mental game. In *STOC*.
[42] Shafi Goldwasser and Leonid A. Levin. 1990. Fair Computation of General Functions in Presence of Immoral Majority. In *CRYPTO*. 77–93.
[43] Shafi Goldwasser and Rafail Ostrovsky. 1992. Invariant Signatures and Non-Interactive Zero-Knowledge Proofs are Equivalent (Extended Abstract). In *CRYPTO*. 228–245.
[44] S. Dov Gordon. 2010. On Fairness in Secure Computation. Ph.D. Dissertation. (2010). https://www.cs.umd.edu/~jkatz/THESES/gordon.pdf.
[45] S. Dov Gordon, Carmit Hazay, Jonathan Katz, and Yehuda Lindell. 2008. Complete fairness in secure two-party computation. In *STOC*. 413–422.
[46] S. Dov Gordon, Yuval Ishai, Tal Moran, Rafail Ostrovsky, and Amit Sahai. 2010. On Complete Primitives for Fairness. In *TCC*. 91–108.
[47] S. Dov Gordon and Jonathan Katz. 2009. Complete Fairness in Multi-party Computation without an Honest Majority. In *Theory of Cryptography, 6th Theory of Cryptography Conference, TCC 2009, San Francisco, CA, USA, March 15-17, 2009. Proceedings*. 19–35.
[48] S. Dov Gordon and Jonathan Katz. 2010. Partial Fairness in Secure Two-Party Computation. In *EUROCRYPT*. 157–176.
[49] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. 2017. Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol. In *CRYPTO '17*.
[50] Dafna Kidron and Yehuda Lindell. 2011. Impossibility Results for Universal Composability in Public-Key Models and with Fixed Inputs. *J. Cryptology* 24, 3 (2011), 517–544. https://doi.org/10.1007/s00145-010-9069-7
[51] Handan Kilinç and Alptekin Küpçü. 2016. Efficiently Making Secure Two-Party Computation Fair. In *Financial Cryptography and Data Security - 20th International Conference, FC 2016, Christ Church, Barbados, February 22-26, 2016, Revised Selected Papers*. 188–207. https://doi.org/10.1007/978-3-662-54970-4_11
[52] Ranjit Kumaresan and Iddo Bentov. 2016. Amortizing Secure Computation with Penalties. In *ACM CCS*. 418–429.
[53] Ranjit Kumaresan, Tal Moran, and Iddo Bentov. 2015. How to Use Bitcoin to Play Decentralized Poker. In *ACM CCS*. 195–206.
[54] Alptekin Küpçü and Anna Lysyanskaya. 2010. Usable Optimistic Fair Exchange. In *CT-RSA*. 252–267.
[55] Yehuda Lindell. 2009. Legally Enforceable Fairness in Secure Two-Party Communication. *Chicago J. Theor. Comput. Sci.* 2009 (2009).

[56] Anna Lysyanskaya. 2002. Unique Signatures and Verifiable Random Functions from the DH-DDH Separation. In *CRYPTO*. 597–612.
[57] Silvio Micali. 2003. Simple and fast optimistic protocols for fair electronic exchange. In *PODC*. 12–19.
[58] Rafael Pass, Elaine Shi, and Florian Tramèr. 2016. Formal Abstractions for Attested Execution Secure Processors. *IACR Cryptology ePrint Archive* 2016 (2016), 1027. http://eprint.iacr.org/2016/1027
[59] Rafael Pass, Elaine Shi, and Florian Tramèr. 2017. Formal Abstractions for Attested Execution Secure Processors. In *EUROCRYPT*. 260–289.
[60] Benny Pinkas. 2003. Fair Secure Two-Party Computation. In *EUROCRYPT*. 87–105.
[61] Tal Rabin and Michael Ben-Or. 1989. Verifiable Secret Sharing and Multiparty Protocols with Honest Majority (Extended Abstract). In *STOC*. 73–85.
[62] Andrew Chi-Chih Yao. 1982. Protocols for Secure Computations (Extended Abstract). In *FOCS*. 160–164.

A PROOFS

A.1 Proof of Claim 2

The analysis for the proof below is taken from [40, 44].

Proof. The simulator $\mathcal{S}$ outputs fail only if it has reached step 7 and then fails in producing an accepting transcript. $\mathcal{S}$ fails to reach step 7 with probability q.

We denote by p, the probability that the adversary does not aborts when given the witness encryption of the correct functionality, i.e., p is the probability when the adversary is given the witness encryption of $\mathcal{F}(x_1, \cdots, x_n)$. Recollect that q is the probability of the adversary not aborting when given the witness encryption of $\mathcal{F}(\widehat{x}_1, \cdots, \widehat{x}_n)$ where $\forall a, \widehat{x}_a = x_a$ and $\forall h, \widehat{x}_h = 0$. From the security of witness encryption, we require $|q-p|$ is negligible in the security. (probability is taken over the random coins used to generate the output of $\mathcal{F}'_{\Delta t}$.)

$$\begin{aligned}\Pr[\mathcal{S} \text{ outputs fail}] &= q \sum_i \left(\Pr\left[\frac{1}{\widetilde{q}} = i\right]\right)(1-p)^{t\cdot i} \\ &\leq q\Pr\left[\frac{q}{\widetilde{q}} \geq \frac{1}{2}\right](1-p)^{\frac{t}{\widetilde{q}}} + q\Pr\left[\frac{q}{\widetilde{q}} < \frac{1}{2}\right] \\ &\leq q(1-p)^{\frac{t}{2q}} + \mathsf{negl}(\lambda) \qquad (1)\end{aligned}$$

To show that the above equation is negligible in λ, we split the analysis into two cases:

- **Case 1:** $p \geq \frac{q}{2}$. Substituting, we get

$$(1-p)^{\frac{t}{2q}} \leq \left(1 - \frac{q}{2}\right)^{\frac{t}{2q}} < e^{-\frac{t}{4}}$$

  which is negligible is λ since t is polynomial in λ.
- **Case 1:** $p < \frac{q}{2}$. To the contrary, let us assume Equation 1 is non-negligible. Then, there is a polynomial poly and infinitely many values λ such that

$$q \geq q(1-p)^{\frac{t}{2q}} + \mathsf{negl}(\lambda) > \frac{1}{\mathsf{poly}(\lambda)}.$$

  Thus $q > \frac{1}{\mathsf{poly}'(\lambda)}$ for some polynomial poly′. This gives us

$$|q-p| > \left|\frac{q}{2}\right| > \frac{1}{2\mathsf{poly}'(\lambda)}.$$

  This breaks the security of the witness encryption scheme.

Thus $\mathcal{S}$ outputs fail with only negligible probability. □

A.2 Proof of Theorem 6.1

We consider the two party setting where P_1 is corrupted. The other case is symmetric. The simulator $\mathcal{S}$ works as follows:

(1) Unless otherwise mentioned, $\mathcal{S}$ passes through messages between adversary $\mathcal{A}(P_1)$ and $\mathcal{G}_{\mathsf{att}}$.
(2) $\mathcal{S}$ loads the program to get the corresponding eid_0, i.e. $eid_0 \leftarrow \mathcal{G}_{\mathsf{att}}.\mathtt{install}(\mathsf{prog}_{\mathsf{fair}}[\Delta t, \mathcal{P}_0, \mathcal{P}_1, \mathsf{vk}_{\mathsf{BB}}, 0])$.
(3) Next, $\mathcal{S}$ initiates the key exchange phase $(g^a, \sigma_0) \leftarrow \mathcal{G}_{\mathsf{att}}.\mathtt{resume}(eid_0, \text{“keyex”})$ and sends (eid_0, g^a, σ_0) message to $\mathcal{A}$.
(4) $\mathcal{S}$ waits to receive $\left(eid_1, g^b, \sigma_1\right)$ from $\mathcal{A}$.
The simulator sees messages between $\mathcal{A}$ and $\mathcal{G}_{\mathsf{att}}$, and can see if $\left(eid_1, g^b, \sigma_1\right)$ sent by $\mathcal{A}$ is different from the corresponding tuple it received from $\mathcal{G}_{\mathsf{att}}$. If the tuples differ and the signature verifies, output $\perp_{\mathcal{G}_{\mathsf{att}}}$ and exit.
(5) Pick $k_0 \xleftarrow{\$} \{0,1\}^\lambda, \rho_0 \xleftarrow{\$} \{0,1\}^\lambda$, $r_0 \xleftarrow{\$} \{0,1\}^\lambda$, $t_0 \leftarrow \mathsf{BB}(\text{getCurrentCounter})$ and initialize $\mathcal{G}_{\mathsf{att}}$ with these values, $(\mathsf{com}_0, \_)\mathcal{G}_{\mathsf{att}}.\mathtt{resume}(eid_0, \text{“init”}, \rho_0, k_0, t_0, r_0)$. Simulator sees the values (ρ_1, k_1, t_1, r_1) that $\mathcal{A}$ sends to $\mathcal{G}_{\mathsf{att}}$.
(6) $\mathcal{S}$ calls $(\mathsf{ct}_1, \_) \leftarrow \mathcal{G}_{\mathsf{att}}.\mathtt{resume}(eid_0, \text{“send”})$, sends ct_1 to $\mathcal{A}$ and waits for ct_0.
As before, the simulator observes if ct_0 sent by $\mathcal{A}$ is different from the value it received from $\mathcal{G}_{\mathsf{att}}$. If so, and $\mathcal{G}_{\mathsf{att}}$ doesn't throw an exception, output $\perp_{\mathsf{AE}_1}$ and exit.
(7) Make a call to $\mathcal{G}_{\mathsf{att}}$ to get the parameters, $(T, y, \_) \leftarrow \mathcal{G}_{\mathsf{att}}.\mathtt{resume}(eid_0, \text{“getParams”}, \perp)$.
(8) Wait for $\mathcal{A}$ to send $(x_1, k'_1, \mathsf{com}'_0, \mathsf{com}'_1, r'_1)$ intended for $\mathcal{F}'$. If the commitment values are not the same as the ones received earlier, send abort to the ideal functionality and send $\perp$ to the adversary. If $k'_1 \neq k_1$, i.e. the key shares sent at different points differ, and if $\mathsf{com}'_1 = \mathsf{Com}(k'_1; r'_1)$ output $\perp_{\mathsf{com}}$ and exit. Else, pick K' randomly and compute $\mathsf{ct}_{\mathsf{MPC}} \leftarrow \mathsf{AE}.\mathsf{Enc}_{K'}(\mathcal{F}(0, x_1))$ to send to $\mathcal{A}$.
(9) $\mathcal{S}$ obtains its token share from $\mathcal{G}_{\mathsf{att}}$, $(\rho_0, \_) \leftarrow \mathcal{G}_{\mathsf{att}}.\mathtt{resume}(eid_0, \text{“getTokenShare”}, \perp)$.
(10) If $\mathcal{A}$ aborts immediately after receiving the output from $\mathcal{F}'$ without the honest party getting it, send abort to the ideal functionality. But continue running $\mathcal{S}$. If the adversary sends to the bulletin board or enclave the correct pre-image of y, $\mathcal{S}$ outputs $\perp_f$ and exits.
(11) If $\mathcal{A}$ has not aborted, send ρ_0 to $\mathcal{A}$. If the adversary does not send ρ_1, or post a valid pre-image during the interval T to $T + \Delta T$, but queries $\mathcal{G}_{\mathsf{att}}$ for the output on a valid authentication tag, then we output $\perp_{\mathsf{BB}}$ and exit.
(12) Alternatively, we split the behavior of the simulator three cases:
   - If $\mathcal{A}$ responds with a valid ρ_1 (i.e. $f(\rho_0 \oplus \rho_1) = y$), then post to the bulletin board. Recollect that $\mathcal{S}$ has reached this point only if the key shares sent by $\mathcal{A}$ were consistent. Send x_1 to the ideal functionality to receive out. If $\mathcal{A}$ makes the correct query to the enclave, i.e. the ciphertext sent is the same as the one from the MPC, $\mathcal{S}$ programs the output by returning $\mathcal{G}_{\mathsf{att}}.\mathtt{resume}(eid_1, \text{“output”}, t_{\mathsf{BB}}, \sigma_{\mathsf{BB}}, \mathsf{out})$. If the ciphertext is different and authenticates under key K', then output $\perp_{\mathsf{AE}_2}$ and exit.

- If the adversary does not send ρ_1 but posts a pre-image of y during the interval T to $T + \Delta T$, $\mathcal{S}$ follows the same approach as the previous step.
- If $\mathcal{A}$ attempts to use the backdoor, forward the message to $\mathcal{G}_{\text{att}}$ without modification.

We prove indistinguishability of the real and ideal worlds through a sequence of hybrids.

Hyb_0: Identical to the real execution.

Hyb_1: Identical to Hyb_0 except that we introduce the following check. Observe the messages between $\mathcal{A}$ and $\mathcal{G}_{\text{att}}$, and can see if $\left(eid_1, g^b, \sigma_1\right)$ sent by $\mathcal{A}$ is different from the corresponding tuple it received from $\mathcal{G}_{\text{att}}$. If the tuples differ and the signature verifies, output $\perp_{\mathcal{G}_{\text{att}}}$ and exit.

Claim 4. *Assuming that the underlying signature scheme is secure,* Hyb_1 *is computationally indistinguishable from* Hyb_0.

Proof. Hyb_1 exits with output $\perp_{\mathcal{G}_{\text{att}}}$ with only negligible probability. If not, we can use $\mathcal{A}$ to construct an adversary that breaks the signature scheme. □

Hyb_2: Identical to Hyb_1 except that we replace all occurrences of $\text{sk} = g^{ab}$ with a random key.

Claim 5. *Assuming that DDH holds,* Hyb_2 *is computationally indistinguishable from* Hyb_1.

Proof. Follows directly from DDH security. □

Hyb_3: Identical to Hyb_2 except that we add the following additional checks:

- observe if ct_0 sent by $\mathcal{A}$ is different from the value it received from $\mathcal{G}_{\text{att}}$. If so, and $\mathcal{G}_{\text{att}}$ doesn't throw an exception, output $\perp_{\text{AE}_1}$ and exit.
- if $\mathcal{A}$ sends a different key share k_1' intended for $\mathcal{F}_{\Delta t}'$ and $\text{com}'_1 = \text{Com}(k_1'; r_1')$, output $\perp_{\text{com}}$ and exit.
- if $\mathcal{A}$ aborts immediately after receiving the output from $\mathcal{F}'$ (without the honest party getting it), send abort to the ideal functionality. Additionally, wait to see if the adversary sends to the bulletin board or enclave the correct pre-image of y. If so, outputs $\perp_f$ and exit.
- if the adversary does not send ρ_1 and does not post a valid pre-image during the interval T to $T + \Delta T$ but queries $\mathcal{G}_{\text{att}}$ on a valid authentication tag, output $\perp_{\text{BB}}$ and exit.

Claim 6. *Assuming the security of one-way permutation, statistical binding of the commitment scheme INT-CTXT security of AE and unforgeability of the authentication scheme* Hyb_3 *is computationally indistinguishable from* Hyb_2.

Proof. The only changes are in the checks performed. We argue that Hyb_3 will output a special abort with only negligible probability:

- $\perp_{\text{AE}_1}$ is output with only negligible probability. Else, we can leverage the adversary to break the INT-CTXT security of the AE scheme.
- $\perp_{\text{com}}$ is output with only negligible probability. Else, we can leverage the adversary to break the statistical binding property of the commitment scheme.
- $\perp_f$ is output with only negligible probability. Else we can break the security of the one way function. This follows from the fact that the simulator is see the queries that the adversary makes to the enclave and the bulletin board. Since we want to force the challenge value y^* onto the adversary, we use a backdoor in the function. This backdoor does not give the adversary any undue advantage as the value is not sent across to the other party.
- $\perp_{\text{BB}}$ is output with negligible probability. Else, we can leverage the adversary to break the unforgeability of the authentication scheme for the bulletin board. This is because the adversary was able to produce a signature that has not been queried for before.

□

Hyb_4: Identical to Hyb_3 except that we intercept the ciphertext query for the output, and program the output using the trapdoor to be $\text{AE.Dec}_K(\text{ct})$ if the other conditions are satisfied. Here K is the key in the enclave.

Claim 7. Hyb_4 *is statistically indistinguishable from* Hyb_3.

Proof. This follows from the fact that it was only a statistical change. This is because we moved the exact check to the outside of the enclave. □

Hyb_5: Identical to Hyb_4 except that replace com_2 to be a commitment of a random value.

Claim 8. *If the commitment scheme is computationally hiding,* Hyb_5 *is computationally indistinguishable from* Hyb_4.

Proof. If the two hybrids are distinguishable, we can leverage the adversary to break the computational hiding of the commitment scheme. □

Hyb_5: Identical to Hyb_4 except that we pick K' randomly and use K' to encrypt the output. Now, the output is programmed in the last round with respect to the key K'.

Claim 9. *If the semantic security of the AE scheme holds,* Hyb_5 *is computationally indistinguishable from* Hyb_4.

Proof. If the two hybrids are distinguishable, then we can build an adversary that breaks the semantic security of the AE scheme. □

Hyb_6: Identical to Hyb_5 except that we add the following additional checks. If the ciphertext differs from the MPC output and it authenticates under key K', then output $\perp_{\text{AE}_2}$ and exit.

Claim 10. *If INT-CTXT security of the AE scheme holds,* Hyb_6 *is computationally indistinguishable from* Hyb_5.

Proof. Since the only changes are additional checks, it is enough to show that $\mathcal{S}$ outputs $\perp_{\mathsf{AE}_2}$ with only negligible probability. This follows directly from the INT-CTXT security of the AE scheme. Specifically, we can receive the challenge ciphertext to the be the encryption of the function value (either under the challenge key, or a random key). If the adversary is able to produce a verifying ciphertext different from the one it receives it constitutes a forgery, thus breaking the INT-CTXT security of the AE scheme. □

Hyb_7: Identical to Hyb_6 except that if check did not result in a failure, send x_1 to the trusted party to obtain out. If the witness checks succeeds, program the output of the enclave to be out. Else program output to be $\perp$.

Claim 11. Hyb_7 *is statistically indistinguishable from* Hyb_6.

Proof. The change is only statistical since the execution thread reaches the point only if all prior checks pass. □

Hyb_8: Identical to Hyb_7 except that we replace the value inside the ciphertext to be $\mathcal{F}(0, x_1)$.

Claim 12. *If the semantic security of the encryption scheme holds,* Hyb_8 *is computationally indistinguishable from* Hyb_7.

Proof. If the two hybrids are distinguishable, then we can build an adversary that breaks the semantic security of the encryption scheme. □

Hyb_9: Identical to Hyb_8 except that we replace all occurrences of sk with g^{ab} again.

Claim 13. *Assume DDH is hard,* Hyb_9 *is computationally indistinguishable from* Hyb_8.

Proof. Follows directly from DDH security. □

Hyb_9 is the same as our simulator, and hence we're done.

B SGX FUNCTIONALITY

The SGX functionality is presented in Figure 5. Additional notation from [59] used is described below:

- $\mathcal{P}$ is the identifier of party.
- reg refers to the registry of machines with the trusted hardware.
- prog is the program.
- inp, outp refers to the input and output.
- mem is the program's memory tape.
- Σ is a signature scheme.

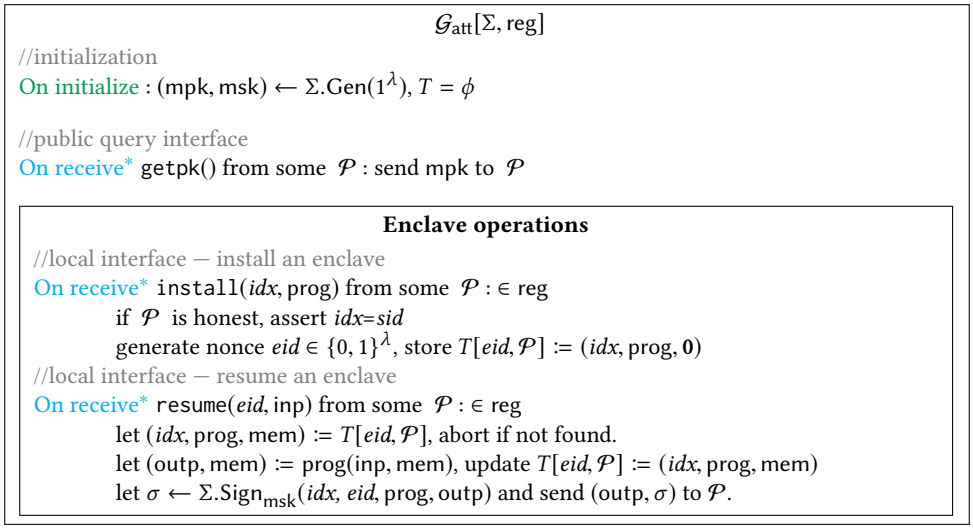

Figure 5: A global functionality modeling an SGX-like secure processor. Blue (and starred*) activation points denote reentrant activation points. Green activation points are executed at most once. The enclave program prog may be probabilistic and this is important for privacy-preserving applications. Enclave program outputs are included in an anonymous attestation σ. For honest parties, the functionality verifies that installed enclaves are parameterized by the session id *sid* of the current protocol instance.