Universally Composable Secure Two and Multi-party Computation in the Corruptible Tamper-Proof Hardware Token Model

Nishanth Chandran
Microsoft Research India, India
nichandr@microsoft.com

Wutichai Chongchitmate
UCLA, USA
wutichai@cs.ucla.edu

Rafail Ostrovsky*
UCLA, USA
rafail@cs.ucla.edu

Ivan Visconti
Università di Salerno, ITALY
visconti@unisa.it

Abstract

In this work we introduce the *corruptible token model*. This model generalizes the *stateless tamper-proof token model* introduced by Katz (EUROCRYPT '07) and relaxes the trust assumption. Our improved model is motivated by the real-world practice of outsourcing hardware production to possibly untrusted manufacturers and allows tokens created by honest parties to be corrupted at the time of their creation.

Assuming one-way functions, we show how to UC-securely realize the tamper-proof token functionality of Katz in the corruptible token model with n stateless tokens assuming that the adversary corrupts at most $n-1$ of them. We then apply this transformation to existing two and MPC protocols to achieve a UC-secure 2PC/MPC protocol in the corruptible token model assuming only the existence of one-way functions.

Finally, we further transform the above protocol to only use tokens of small size that take only short inputs. The technique in the last transformation can also be used to improve the assumption of UC-secure hardware obfuscation by Nayak *et al.* (NDSS '17) from collision-resistant hash functions to one-way functions, which can then be transformed into a protocol with n corruptible tokens in our model.

*Research supported in part by NSF grant 1619348, DARPA, US-Israel BSF grant 2012366, OKAWA Foundation Research Award, IBM Faculty Research Award, Xerox Faculty Research Award, B. John Garrick Foundation Award, Teradata Research Award, and Lockheed-Martin Corporation Research Award. The views expressed are those of the authors and do not reflect position of the Department of Defense or the U.S. Government.

1 Introduction

UC-secure MPC. Secure multi-party computation [GMW87] (MPC) allows mutually distrustful parties to jointly compute a function f preserving the privacy of their inputs/outputs. Canetti [Can01] introduced the notion of universal composability (UC) to model secure MPC in an environment where multiple concurrent executions of different protocols take place. Unfortunately UC security is significantly harder to achieve than plain secure computation. In fact, in the plain model (i.e., without trusted set-up assumptions, physical assumptions, superpolynomial-time simulation and so on) most functionalities can not be UC-realized [CF01, CKL06, Lin03].

One of the widely studied assumptions that enable the realization of UC-secure MPC is the existence of tamper-proof hardware tokens. This assumption was modeled by Katz [Kat07] through a functionality in order to abstractly capture physical tamper-proof hardware that are created and sent by a party (sender) to another party (receiver). The receiver can use the received token to execute the program stored in it multiple times as a black-box, on inputs of his choice. Tokens can be either stateful (i.e., they retain an updatable memory between executions) or stateless (all executions start with the same configurations). Motivated by the challenging open questions on (in)feasibility results and by the practical relevance of the model, UC-security with tamper-proof tokens has been widely explored with recent focus on the more challenging case of stateless tokens [CGS08, LPV09, GIS$^+$10, DSMRV13, CKS$^+$14, DKMQN15, HPV16, NFR$^+$17].

All previous work critically relies on the honest player being able to construct tamper-proof tokens[1]. This is clearly a very demanding assumption that concretely requires honest players to rely on the honesty of a token manufacturer that they trust. In turn, while the tamper-proof token model in theory consists of a physical assumption, in practice it degenerates into a model where the security of an honest player depends on the honest behavior of an external player chosen by the honest player[2]. The question that we ask in this work is: *"Can trust in a given tamper-proof token manufacturer be relaxed?"*

Weakening trust in the hardware model. We address the above open problem and solve it under minimal complexity assumptions. More specifically, we consider the concrete scenario where the sender of a token does not have the ability to physically create a tamper-proof token, but instead has to rely on possibly untrusted manufacturers. In case a manufacturer is corrupted (and may be colluding with other parties), the program embedded in the token may be leaked, or replaced in its entirety. In other words, tokens in this model can be tampered with at the time of its creation.

We define a functionality for UC-security allowing the design of protocols that make use of tokens generated by potentially adversarial manufacturers. In turn, we propose a new model extending the stateless version of Katz's tamper-proof token model in [Kat07], that we call *corruptible token model*. In our new model, the adversary is allowed to corrupt stateless tokens when they are created by honest parties. The attack happens during the token creation phase, and the adversary learns all information that the honest player wanted to store in the token. Moreover the adversary is allowed to replace the token with a different token of its choice, including a stateful one.

[1] An attempt to relax this assumption was done in [FPS$^+$11] focusing only on the set intersection functionality, without considering UC security and using informal definitions.

[2] Similar question for the CRS generation was answered in the multi-string model [GO07], where multiple corruptible CRS's are generated.

The corruptible token model abstractly represents the process of outsourcing the production of hardware tokens to possibly corrupted manufacturers. We only allow the corruption to occur at the time the tokens are created. Our model also allows adaptive corruption of the manufacturers, in the sense that the adversary may choose to corrupt new tokens based on what it learns from his view from other corrupted tokens. So, instead of preselecting the manufacturers beforehand, the adversary can corrupt tokens individually one by one. Finally, during the token generation phase we allow the adversary to replace tokens with stateful ones to represent the real-world hardware Trojan described in [DFS16].

1.1 Our Results

In this work, we provide the following results:

- We construct a protocol in the corruptible token model using n tokens that UC-realizes a variant of Katz's tamper-proof token functionality. We call this the *tamper-proof token with abort* functionality and UC-realize it assuming that the adversary corrupts at most $n-1$ tokens requested to be created by an honest party. The only (unavoidable) difference between the tamper-proof token with abort functionality and the original Katz's tamper-proof token functionality is that our variant allows the adversary to learn whenever a token is sent (even between honest parties), and can choose to abort and prevent the delivery of that token. Still the adversary learns nothing about the program in the token[3].
- We then show how to transform any protocol in Katz's tamper-proof token model to a protocol in our model having a much improved trust assumption. Indeed the transformed protocol remains secure even when $n-1$ out of the n tokens created by every honest party are corrupted at the time of creation. Our transformation preserves UC security and only assumes the existence of one-way functions (OWF). We focus on stateless tokens since this is a milder physical assumption and is the most challenging case. Requiring one token to be uncorrupted is unavoidable as this clearly leads to the impossibility results for UC in the standard model.
- Our transformation can now be applied to existing protocols in the Katz's token model to obtain new results in the corruptible token model. For instance, starting with the recent UC-secure two and MPC construction in the tamper-proof token model based on OWFs of [HPV16], we get the same result in the corruptible token model under the same assumption.
- Additionally, we also improve the result of [NFR+17] by removing the need of collision-resistant hash functions, and apply our transformation to obtain an obfuscation protocol in the corruptible token model based on OWFs.
- As building blocks for our constructions, we obtain a simultaneous resettable zero-knowledge (sim-res ZK) argument and a UC-secure MPC for any well-formed functionality in the correlated randomness model assuming OWFs only. In the correlated randomness model, each party has access to a private string honestly generated before the execution of the protocol by

[3]The need for abort in the functionality is motivated by the following reason. Suppose the tamper-proof token functionality (without abort) can be realized by n corruptible tokens. Then, the adversary in the corruptible token model corrupts all but one of the tokens and replaces them with corrupted tokens that do nothing. Now, if the tamper-proof token functionality without abort is realized with the remaining one (uncorrupted) token, then this token must hold the complete program and secrets of the honest party (so that it can carry out the computation by itself). However, this token is also susceptible to corruption, and if the adversary had instead corrupted only this token, would have learnt all secrets of the honest party resulting in the insecurity of the protocol. Hence, we must model the functionality to allow for aborts.

the correlated randomness functionality independently of the input. These protocols may be of independent interest.

We finally remark that token corruption was studied in [DFS16] for the case of "hardware Trojans". However, their solution, "the Trojan protection scheme" requires a "master circuit" to remain uncorrupted. Moreover their work is not about UC security.

1.2 Our Techniques

We now discuss the techniques that go into UC-realizing Katz's token functionality in the $(n, n-1)$-token-corruptible hybrid model, (i.e., the model where n tokens are generated by a honest player and at most $n-1$ are corrupted by the adversary at the time of generation). We refer to the final protocol as Π. Given a description of the program P for Katz's tamper-proof token (such a description is specified by the protocol in Katz's model) we create n shares of the description of P using an n-out-of-n threshold secret sharing scheme. Then n tokens are created as follows. The program of the i-th token includes 1) the i-th share; 2) commitments of all other shares; 3) correlated randomness to run a UC-secure n-party protocol Π' to reconstruct P from the shares and to run P on some given input; 4) a seed for a PRF; 5) commitments of all PRF seeds (we call these commitments the determining messages).

n-party UC computation Π for the evaluation of P. The execution of Π consists of running a UC-secure protocol Π' in the correlated randomness model[4]. The random tape needed by Π' is obtained by Π by computing the PRF on the determining messages. Each message m of Π' is followed by a simultaneous resettable ZK argument of knowledge proving that the message m is computed correctly according to the committed seeds, and committed shares. We will prove that as long as the adversary does not corrupt all n tokens, it cannot learn P even when he can run it multiple times on different inputs.

Simultaneous resettable ZK argument in the correlated randomness model from OWFs. The above discussion assumed the existence of a UC-secure MPC protocol Π' in the correlated randomness model. We construct Π' by first constructing a simultaneous resettable zero-knowledge (ZK) argument Π'_{ZK} with straight-line simulator in the correlated randomness model from a 3-round public-coin ZK argument, Σ, in the CRS model with straight-line simulation based on OWFs.

The construction of Π'_{ZK} is done in 2 steps. First, we add the argument of knowledge (AoK) property with straight-line witness extractor to Σ in the correlated randomness model. We use a technique similar to one used for Ω protocols in [MY04] but with a secret key encryption scheme and a commitment scheme instead of a public key encryption scheme. The resulting protocol is still 3-round, public-coin and with straight-line simulation. In the second step, we add a simultaneous resettable witness indistinguishability (sim-res WI) argument to construct a simultaneous resettable zero-knowledge argument in the correlated randomness model with straight-line simulation. The verifier uses a PRF to generate a string c to play in Σ instead of uniform sampling his message. Then the verifier runs the prover of the sim-res WI to prove c is generated honestly or that a given long string d is an output of a PRG on input a short seed. Since d is uniformly chosen as part of the correlated randomness, the verifier cannot maliciously manipulate c.

[4]The correlated randomness is the key that allows us to avoid the impossibility of resettably-secure computation in the standard model proven in [GKOV12].

UC-secure n-party computation in the correlated randomness model. We then construct Π', i.e., a UC-secure n-party computation protocol in the correlated randomness model for any well-formed functionality based on OWFs as follows. First, we apply a modified version of Beaver's technique in [Bea96] to construct UC-secure unbounded number of OTs from a small number of correlated OTs distributed as setup in the correlated randomness model. We then apply the IPS transformation [IPS08], which constructs a UC-secure MPC in the OT-hybrid model, to get the UC-secure MPC. Note that the transformation requires a large number of access to the OT. Thus, the OT extension technique is required. All the above techniques are then put together carefully to obtain our final result.

Practical tokens. In order to ensure that the queries to tokens are short and the size of each token is small, we consider a technique used in [NFR$^+$17] where a large input is fed into a token in blocks of small size. To ensure the consistency of the input, in [NFR$^+$17] a Merkle's tree based on CRHFs is used to commit to the input beforehand. We improve on this technique by replacing the Merkle's tree with a new construction based on OWFs. This result can be seen to be of independent interest as an improvement on the assumption of [NFR$^+$17].

1.3 Organization of the paper

We present the building blocks used in our construction (such as interactive argument systems, resettable zero-knowledge and so on) as well as describe the correlated randomness model and UC security in Section 2. In Section 3, we present our first result of a simultaneous resettable zero-knowledge protocol in the correlated randomness model, based solely on OWFs. In Section 4, we show how to construct a UC-secure MPC protocol in the correlated randomness model based on OWFs. We define our corruptible tamper-proof token model in Section 5 and we present our main compiler that converts any protocol in the Katz's tamper-proof token model into our $(n, n-1)-$corruptible tamper-proof token model in Section 6 of the paper. Finally, we show an application of our compiler to secure obfuscation in Section 7.

2 Preliminaries

2.1 Building Blocks

A polynomial-time relation R is a relation for which it is possible to verify in time polynomial in $|x|$ whether $R(x, w) = 1$. Let us consider an $\mathcal{NP}$-language L and denote by R_L the corresponding polynomial-time relation such that $x \in L$ if and only if there exists w such that $R_L(x, w) = 1$. We will call such a w a *valid witness for* $x \in L$. Let λ denote the security parameter. A *negligible* function $\nu(\lambda)$ is a non-negative function such that for any constant $c < 0$ and for all sufficiently large λ, $\nu(\lambda) < \lambda^c$. We will denote by $\Pr_r[X]$ the probability of an event X over coins r, and $\Pr[X]$ when r is not specified. The abbreviation "PPT" stands for probabilistic polynomial time. For a randomized algorithm A, let $A(x; r)$ denote running A on an input x with random coins r. If r is chosen uniformly at random with an output y, we denote $y \leftarrow A(x)$. For a pair of interactive Turing machines (P, V), let $\langle P, V\rangle(x)$ denote V's output after interacting with P upon common input x. We say V accepts if $\langle P, V\rangle(x) = 1$ and rejects if $\langle P, V\rangle(x) = 0$. We denote by $\mathtt{view}_{V(x,z)}^{P(w)}$ the view (i.e., its private coins and the received messages) of V during an interaction with $P(w)$

on common input x and auxiliary input z. We will use the standard notion of computational indistinguishability [GM84].

Definition 2.1 (interactive argument system in the correlated randomness model)**.** *An* interactive argument system *for the language L consists of a correlated random string generation algorithm K and a pair of interactive Turing machines (P, V) where V runs on input (σ_V, x) and P runs on input (σ_P, x, w) where w is a witness for x such that:*

- *Efficiency: K, P and V are* PPT*.*
- *(Perfect) Completeness: For every $\lambda \in \mathbb{N}$ and for every pair (x, w) such that $(x, w) \in R_L$,*

$$\Pr[(\sigma_P, \sigma_V) \leftarrow K(1^\lambda) : \langle P(\sigma_P, w), V(\sigma_V)\rangle(x) = 1] = 1.$$

- *Soundness: There exists a negligible function $\nu(\cdot)$ such that for any non-uniform* PPT *adversary $P^* = (P_1^*, P_2^*)$*

$$\Pr[(x, z) \leftarrow P_1^*(\sigma_P) : x \notin L \wedge \langle P_2^*(\sigma_P, z), V(\sigma_V)\rangle(x) = 1] < \nu(\lambda).$$

If K always outputs $\perp$, we say that (P, V) is an interactive argument (in the plain model). If K outputs $\sigma_P = \sigma_V$, we say that (P, V) is an interactive argument in the CRS model.

Definition 2.2 (zero-knowledge arguments)**.** *Let (K, P, V) be an interactive argument system for a language L. We say that (K, P, V) is zero knowledge (ZK) if, for any probabilistic polynomial-time adversary V^*, there exists probabilistic polynomial-time algorithms $S_{V^*} = (S_1, S_2)$ such that, for all auxiliary inputs z and all pairs $(x, w) \in R_L$*

$$|\Pr[(\sigma_P, \sigma_V) \leftarrow K(1^\lambda) : \langle P(\sigma_P, w), V^*(\sigma_V, z)\rangle(x) = 1]$$

$$-\Pr[(\sigma_P, \sigma_V, \tau) \leftarrow S_1(1^\lambda) : \langle S_2(\tau), V^*(\sigma_V, z)\rangle(x) = 1]| < \nu(\lambda)$$

Definition 2.3 (witness indistinguishability)**.** *Let L be a language in $\mathcal{NP}$ and R_L be the corresponding relation. An interactive argument (K, P, V) for L is* witness indistinguishable *(WI) if for every verifier V^*, every pair (w_0, w_1) such that $(x, w_0) \in R_L$ and $(x, w_1) \in R_L$ and every auxiliary input z, for $\sigma \leftarrow K(1^\lambda)$, the following ensembles are computationally indistinguishable:*

$$\{\mathtt{view}_{V^*(\sigma,x,z)}^{P(\sigma,w_0)}\} \quad \textit{and} \quad \{\mathtt{view}_{V^*(\sigma,x,z)}^{P(\sigma,w_1)}\}.$$

Definition 2.4 (argument of knowledge)**.** *Let (K, P, V) be an interactive argument system for a language L. We say that (K, P, V) is* argument of knowledge *if there exists probabilistic polynomial-time algorithms $\mathcal{E} = (\mathcal{E}_1, \mathcal{E}_2)$ such that*

- *for all non-uniform polynomial-time adversary $\mathcal{A}$,*

$$\Pr[(\sigma_P, \sigma_V) \leftarrow K(1^\lambda) : \mathcal{A}(\sigma_P) = 1] = \Pr[(\sigma_P, \sigma_V, \tau) \leftarrow \mathcal{E}_1(1^\lambda) : \mathcal{A}(\sigma_P) = 1]$$

- *for all non-uniform polynomial-time adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, the following experiments are indistinguishable:*

$\mathsf{Exp}_{\mathcal{A}}(\lambda)$:	$\mathsf{Exp}^{\mathcal{E}}_{\mathcal{A}}(\lambda)$:
1. $(\sigma_P, \sigma_V) \leftarrow K(1^\lambda)$.	1. $(\sigma_P, \sigma_V, \tau) \leftarrow \mathcal{E}_1(1^\lambda)$.
2. $(x, z) \leftarrow \mathcal{A}_1(\sigma_P)$.	2. $(x, z) \leftarrow \mathcal{A}_1(\sigma_P)$.
3. $b \leftarrow \langle \mathcal{A}_2(z), V(\sigma_V)\rangle(x)$.	3. $(b, w) \leftarrow \langle \mathcal{A}_2(z), \mathcal{E}_2(\tau)\rangle(x)$.
4. *Output* b.	4. *Output 1 iff* $b = 1$ *and* $(x, w) \in R_L$.

Definition 2.5 (resetting adversary)**.** *Let (K, P, V) be an interactive argument system for a language L, $t = \mathsf{poly}(\lambda)$, $\bar{x} = x_1, \ldots, x_t$ be a sequence of common inputs and $\bar{w} = w_1, \ldots, w_t$ the corresponding witnesses (i.e., $(x_i, w_i) \in R_L$) for $i = 1, \ldots, t$. Let $r_1, \ldots, r_t$ be independent random tapes. We say that a PPT V^* is a* resetting *verifier if for $(\sigma_P, \sigma_V) \leftarrow K(1^\lambda)$, it concurrently interacts with an unbounded number of independent copies of $P(\sigma_P)$ by choosing for each interaction the value i so that the common input will be $x_i \in \bar{x}$, and the prover will use witness w_i, and choosing j so that the prover will use r_j as randomness, with $i, j \in \{1, \ldots, t\}$. The scheduling or the messages to be sent in the different interactions with P are freely decided by V^*. Moreover we say that the transcript of such interactions consists of the common inputs $\bar{x}$ and the sequence of prover and verifier messages exchanged during the interactions. We refer to $\mathtt{view}^{P(\sigma_P, \bar{w})}_{V^*(\sigma_V, \bar{x}, z)}$ as the random variable describing the content of the random tape of V^* and the transcript of the interactions between P and V^* using $(\sigma_P.\sigma_V)$ as correlated random strings, where z is an auxiliary input received by V^*.*

Definition 2.6 (resettable zero knowledge)**.** *Let (K, P, V) be an interactive argument system for a language L. We say that (K, P, V) is resettable zero knowledge (rZK) if, for any PPT resetting verifier V^* there exists a expected probabilistic polynomial-time algorithm $S_{V^*} == (S_1, S_2)$ such that the for all pairs $(\bar{x}, \bar{w}) \in R_L$, for $(\sigma_P, \sigma_V, \tau) \leftarrow S_1(1^\lambda)$, the ensembles $\{\mathtt{view}^{P(\sigma_P, \bar{w})}_{V^*(\sigma_V, \bar{x}, z)}\}$ and $\{S_{V^*}(\tau, \bar{x}, z)\}$ are computationally indistinguishable.*

Definition 2.7 (resettable WI)**.** *Let L be a language in $\mathcal{NP}$ and R_L be the corresponding relation. An interactive argument system (K, P, V) for L is* resettable *witness indistinguishable (rWI) if for every PPT resetting verifier V^* every $t = \mathsf{poly}(\lambda)$, and every pair $(\bar{w}^0 = (w^0_1, \ldots, w^0_t), \bar{w}^1 = (w^1_1, \ldots, w^1_t))$ such that $(x_i, w^0_i) \in R_L$ and $(x_i, w^1_i) \in R_L$ for $i = 1, \ldots, t$, and any auxiliary input z, for $\sigma \leftarrow K(1^\lambda)$, the following ensembles are computationally indistinguishable:*

$$\{\mathtt{view}^{P(\sigma, \bar{w}^0)}_{V^*(\sigma, \bar{x}, z)}\} \quad \textit{and} \quad \{\mathtt{view}^{P(\sigma, \bar{w}^1)}_{V^*(\sigma, \bar{x}, z)}\}.$$

Definition 2.8 (resettably-sound arguments)**.** *A resetting attack of a cheating prover P^* on a resettable verifier V is defined by the following two-step random process, indexed by a security parameter λ.*

1. *Uniformly select and fix $t = \mathsf{poly}(\lambda)$ random-tapes, denoted $r_1, \ldots, r_t$, for V, resulting in deterministic strategies $V^{(j)}(x) = V_{\sigma_V, x, r_j}$ defined by $V_{\sigma_V, x, r_j}(\alpha) = V(\sigma_V, x, r_j, \alpha)$,[5] where $(\sigma_P, \sigma_V) \leftarrow K(1^\lambda)$, $x \in \{0,1\}^\lambda$ and $j \in [t]$. Each $V^{(j)}(\sigma_V, x)$ is called an incarnation of V.*

[5]Here, $V(\sigma_V, x, r, \alpha)$ denotes the message sent by the strategy V on the correlated random string σ_V, common input x, random-tape r, after seeing the message-sequence α.

2. *On input 1^λ, machine P^* is allowed to initiate $\mathsf{poly}(\lambda)$-many interactions with the $V^{(j)}(x)$'s. The activity of P^* proceeds in rounds. In each round P^* chooses $x \in \{0,1\}^\lambda$ and $j \in [t]$, thus defining $V^{(j)}(x)$, and conducts a complete session with it.*

Let (K, P, V) be an interactive argument for a language L. We say that (K, P, V) is a resettably-sound argument *for L if the following condition holds:*

- Resettable-soundness*: For every polynomial-size resetting attack, the probability that in some session the corresponding $V^{(j)}(x)$ has accepted and $x \notin L$ is negligible.*

An interactive argument system that is both resettable zero-knowledge and resettably-sound is called *simultaneous resettable zero-knowledge* argument.

Definition 2.9 (commitment scheme)**.** *Given a security parameter 1^λ, a commitment scheme* com *is a two-phase protocol between two* PPT *interactive algorithms, a sender S and a receiver R. In the commitment phase S on input a message m interacts with R to produce a commitment $c = \mathsf{com}(m)$. In the decommitment phase, S sends to R a decommitment information d such that R accepts m as the decommitment of c.*

Formally, we say that com *is a perfectly binding commitment scheme if the following properties hold:*

Correctness:

- *Commitment phase. Let $c = \mathsf{com}(m)$ be the commitment of the message m given as output of an execution of* com *where S runs on input a message m. Let d be the private output of S in this phase.*
- *Decommitment phase*[6]*. R on input m and d accepts m as decommitment of c.*

Statistical (resp. Computational) Hiding ([Lin10]): *for any adversary (resp.* PPT *adversary) $\mathcal{A}$ and a randomly chosen bit $b \in \{0,1\}$, consider the following hiding experiment $\mathsf{ExpHiding}^b_{\mathcal{A},\mathsf{com}}(\lambda)$:*

- *Upon input 1^λ, the adversary $\mathcal{A}$ outputs a pair of messages m_0, m_1 that are of the same length.*
- *S on input the message m_b interacts with $\mathcal{A}$ to produce a commitment of m_b.*
- *$\mathcal{A}$ outputs a bit b' and this is the output of the experiment.*

For any adversary (resp. PPT *adversary) $\mathcal{A}$, there exist a negligible function ν, s.t.:*

$$\left| \Pr[\mathsf{ExpHiding}^0_{\mathcal{A},\mathsf{com}}(\lambda) = 1] - \Pr[\mathsf{ExpHiding}^1_{\mathcal{A},\mathsf{com}}(\lambda) = 1] \right| < \nu(\lambda).$$

Statistical (resp. Computational) Binding: *for every commitment* $\mathtt{com}$ *generated during the commitment phase by a possibly malicious unbounded (resp. malicious* PPT*) sender S^* there exists a negligible function ν such that S^*, with probability at most $\nu(\lambda)$, outputs two decommitments (m_0, d_0) and (m_1, d_1), with $m_0 \neq m_1$, such that R accepts both decommitments.*

We also say that a commitment scheme is perfectly binding *iff $\nu(\lambda) = 0$.*

[6] In this paper we consider a non-interactive decommitment phase only.

Definition 2.10 (secret sharing scheme)**.** *Let K be a finite set of secrets. An n-*out-of-n secret sharing scheme *$\mathcal{S}$ consists of a randomized algorithm $\mathsf{share} : K \to K_1 \times \ldots \times K_n$ and a deterministic algorithm $\mathsf{recon} : K_1 \times \ldots \times K_n \to K$ satisfying*

- *Correctness: for any $s \in K$,*

$$\Pr[\mathsf{recon}(\mathsf{share}(s)) = s] = 1;$$

- *Privacy: for any $s, s' \in K$, $(s_1, \ldots, s_n) \in K_1 \times \ldots \times K_n$ and any $T \subsetneq [n]$,*

$$\Pr[\mathsf{share}(s)_T = (s_i)_{i \in T}] = \Pr[\mathsf{share}(s')_T = (s_i)_{i \in T}]$$

where $(s_1, \ldots, s_n)_T = (s_i)_{i \in T}$.

Definition 2.11 (pseudorandom function (PRF))**.** *A family of functions $\{f_s\}_{s \in \{0,1\}^*}$ is called* pseudorandom *if for all adversarial PPT machines $\mathcal{A}$, for every positive polynomial $p(\cdot)$, and sufficiently large $\lambda \in \mathbb{N}$, it holds that*

$$|\Pr[\mathcal{A}^{f_s}(1^\lambda) = 1] - \Pr[\mathcal{A}^{F}(1^\lambda) = 1]| \leq \frac{1}{p(\lambda)}.$$

where $|s| = n$ and F denotes a truly random function.

2.2 UC Security in the Correlated Randomness Model

In this section, we describe our setup model used in the construction of the resettable ZK and the MPC in the following sections. The correlated randomness model is an extension of the CRS model where each party has an access to a random string generated by an outside trusted party. Unlike in the CRS model, the random string for each party may be different, but possibly correlated. Also, unlike in the augmented CRS model of [CDPW07], honest parties can access their own private string. Thus, it can be considered a variant of the key registration (KR) model of [CDPW07].

Our correlated randomness model is defined to be consistent with that in [IKM+13], but more formally in the UC setting. A protocol ϕ in the correlated randomness model is defined with the corresponding correlated randomness functionality $\mathcal{F}_{corr}^{\phi}$, which generates a correlated random string for each party in the protocol ϕ independently of the parties' input. Each party can access its random string (but not other parties' random strings) by invoking $\mathcal{F}_{corr}^{\phi}$.

In the security proof, the ideal world simulator is allowed to obtain the correlated random strings associated to all parties, thereby having an advantage over the real world adversary.

Let ϕ be n-party protocol in the correlated randomness model. Let $\mathcal{D}$ be a distribution on $S_1 \times \ldots \times S_n$ where S_i is the set of possible random strings for party P_i. The correlated randomness functionality $\mathcal{F}_{corr}^{\phi}$ is defined in Figure 1.

Definition 2.12. *Let $\mathcal{F}$ be an ideal functionality and let ϕ be a multi-party protocol. Then the protocol ϕ is* UC-secure in the correlated randomness model *if ϕ UC realizes $\mathcal{F}$ in $\mathcal{F}_{corr}^{\phi}$-hybrid model. That is if for every PPT hybrid model adversary $\mathcal{A}$, there exists a uniform PPT simulator $\mathcal{S}$ such that for every non-uniform environment $\mathcal{Z}$, the following two ensembles are computationally indistinguishable*

$$\{\mathsf{View}_{\phi,\mathcal{A},\mathcal{Z}}^{\mathcal{F}_{corr}^{\phi}}(\lambda)\}_{\lambda \in \mathbb{N}} \approx^c \{\mathsf{View}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(\lambda)\}_{\lambda \in \mathbb{N}}.$$

$\mathcal{F}_{corr}^{\phi}$

When receiving (sid) from P_i:

1. If there is no tuple of the form $(\mathsf{sid}, \star, \ldots, \star)$,
   (a) Generate $(s_1, \ldots, s_n) \leftarrow \mathcal{D}(1^\lambda)$.
   (b) Store $(\mathsf{sid}, s_1, \ldots, s_n)$.

   Otherwise, retrieve the stored $(\mathsf{sid}, s_1, \ldots, s_n)$.
2. Send (sid, s_i) to P_i.

Figure 1: Correlated Randomness Functionality $\mathcal{F}_{corr}^{\phi}$

3 Sim. Resettable ZK in the Correlated Randomness Model

In this section, we construct a simultaneous resettable ZK argument in the correlated randomness model with straight-line simulator. Our construction only assumes the existence of OWFs. The main building block for the construction is a 3-round (3 messages) public-coin ZK argument protocol in the CRS model with straight-line simulator and based on OWFs (such as in [MY04]) We first construct a new protocol with the above properties and an argument of knowledge with straight-line witness extractor in the correlated randomness model without additional assumptions. We then convert it into a protocol that is simultaneously resettable in the following way: we have the verifier prove, using a simultaneous resettable WI (srWI) argument (based on OWFs [COPV13]), that: either the verifier random message c is the output of a PRF on input the transcript so far (using as seed a value that has been committed to in the correlated randomness), or that a long string d present in the correlated randomness is the output of a PRG on input a short string. The prover receives the commitment and d necessary to run as a verifier of the srWI as part of its correlated randomness.

3.1 ZKAoK in the correlated randomness model from OWFs

We first show how to convert a 3-round public-coin ZK argument in the CRS model with straight-line simulator (based on OWFs) into one that is also an argument of knowledge (with straight-line simulator and witness extractor) in the correlated randomness model. Let $\Pi_{ZK} = (K, P, V)$ be the ZK argument in the CRS model with straight-line simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ (e.g. [MY04]). Let $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be a CPA-secure secret key encryption scheme. Define (K', P', V') in the correlated randomness model as in Figure 2.

Lemma 3.1. *Π_{ZKAoK} is ZKAoK with straight-line simulator and witness extractor in the correlated randomness model.*

Proof. Zero-knowledge: we construct a straight-line simulator $\mathcal{S}' = (\mathcal{S}'_1, \mathcal{S}'_2)$ as follows. $\mathcal{S}'_1$ runs $\mathcal{S}'_1$ to generate (σ', τ), generates sk, k, γ_0 as in K_0, and outputs $s'_P = (\sigma', sk, k, \gamma_0)$, $s'_V = (\sigma', k)$

$$\Pi_{ZKAoK} = (K', P', V')$$

$K'(1^\lambda)$:

1. $\sigma \leftarrow K(1^\lambda)$, $sk \leftarrow \mathsf{KeyGen}(\lambda)$. Let $k = \mathsf{com}(sk)$ and γ_0 be the decommitment information.
2. K' outputs $s_P = (\sigma, sk, k, \gamma_0)$ and $s_V = (\sigma, k)$.

Execution phase: P on input (x, w) and private string s_P; V on input x and private string s_V

1. P' parses $s_P = (\sigma, sk, k, \gamma_0)$, computes $e \leftarrow \mathsf{Enc}(sk, w)$ and sends e to V'.
2. V' parses $s_V = (\sigma, k)$.
3. P' and V' run $\langle P(w'), V\rangle(\sigma, x')$ where $x' = (x, e, k)$ and $w' = (w, sk, \gamma_0)$ to prove that there exists w, sk, γ_0 such that $(x, w) \in R_L$ and $w = \mathsf{Dec}(sk, e)$ and k can be decommitted to sk using γ_0.
4. V' outputs the output of V.

Figure 2: ZKAoK argument protocol Π_{ZKAoK} in the correlated randomness model

and $\tau' = (sk, k, \gamma_0)$. $\mathcal{S}_2'(\tau')$ sends $e' \leftarrow \mathsf{Enc}(sk, 0^{|w|})$ and runs $\mathcal{S}_2(\tau)$ to generate messages. We show the indistinguishability by considering a hybrid $\mathsf{Hyb}_{\mathcal{S}}^{\mathsf{Enc}}$ where $\mathcal{S}$ is used to generate the CRS σ' and messages, but the prover sends $e \leftarrow \mathsf{Enc}(sk, w)$ as in Π_{ZKAoK}. Finally, $\mathsf{Hyb}_{\mathcal{S}}^{\mathsf{Enc}}$ outputs the verifier's output. This hybrid is indistinguishable from $\mathsf{Exp}_{\Pi_{ZKAoK}} = \langle P'(w, s_P), V'(s_V)\rangle(x)$ by the zero-knowledge property of Π_{ZK}. It is also indistinguishable from the experiment $\mathsf{Exp}_{\mathcal{S}'} = \langle \mathcal{S}_2'(\tau'), V'(s_V')\rangle(x)$ running the above simulator by the security of the encryption scheme.

Argument of knowledge: we construct a straight-line witness extractor $\mathcal{E} = (\mathcal{E}_1, \mathcal{E}_2)$ as follows. $\mathcal{E}_1$ generates s_P and s_V as in K and also outputs $\tau = sk$. $\mathcal{E}_2$ runs V' honestly, and if V' accepts, it decrypts and outputs $w = \mathsf{Dec}(sk, e)$. Otherwise, it outputs $\perp$. If V' accepts but $(x, w) \notin R_L$, we have (x', w') fails to satisfy the relation proved by Π_{ZK}. By the soundness of Π_{ZK}, this only occurs with negligible probability. □

If the protocol Π_{ZK} is 3-round and public-coin, the resulting protocol Π_{ZKAoK} is also 3-round and public-coin.

3.2 Simultaneous resettable ZK in the correlated randomness model from OWFs

We now construct a simultaneous resettable ZK protocol in the correlated randomness model based on OWFs. Let $\Pi_{ZKAoK} = (K_{ZKAoK}, P_{ZKAoK}, V_{ZKAoK})$ be a 3-round ZK argument of knowledge protocol in the correlated randomness model with transcript (m_1, c, m_2) where $c \in \{0,1\}^\lambda$ is chosen uniformly at random, a straight-line simulator $\mathcal{S}_{ZKAoK} = (\mathcal{S}_1, \mathcal{S}_2)$, and a straight-line witness extractor $\mathcal{E}_{ZKAoK} = (\mathcal{E}_1, \mathcal{E}_2)$ from Lemma 3.1. Let (P_{WI}, V_{WI}) be a srWI argument (e.g. [COPV13]). Let $\{f_s\}_s$ be a family of pseudorandom functions such that for $s \in \{0,1\}^{\ell_0(\lambda)}$, f_s outputs $c \in \{0,1\}^\lambda$. Let $f : \{0,1\}^{\ell_1(\lambda)} \to \{0,1\}^{\ell_2(\lambda)}$ be a PRG.

We define Π_{srZK} as in Figure 3.

$$\Pi_{srZK} = (K, P, V)$$

$K(1^\lambda)$:

1. $(\sigma_P, \sigma_V) \leftarrow K_{ZKAoK}(1^\lambda)$, $s \leftarrow U_{\ell_0(\lambda)}$, $d \leftarrow U_{\ell_2(\lambda)}$. Let $t = \mathsf{com}(s)$ and γ be the decommitment information.
2. K outputs $s_P = (\sigma_P, t, d)$ and $s_V = (\sigma_V, s, \gamma, t, d)$.

Execution phase: P on input (x, w) and private string s_P; V on input x and private string s_V

1. P parses $s_P = (\sigma_P, t, d)$, runs $P_{ZKAoK}(x, w, \sigma_P)$ to compute m_1, and sends m_1 to V.
2. V parses $s_V = (\sigma_V, s, \gamma, t, d)$, runs $V_{ZKAoK}(x, \sigma_V)$ to sends $c = f_s(x||m_1)$ on behalf of V_{ZKAoK} to P (running P_{ZKAoK}), and runs $P_{WI}(y, (s, \gamma))$, with $y = (t, c, d, x, m_1)$, proving to P running $V_{WI}(y)$ that one of the following statements hold
   - there exists s' and γ such that t can be decommitted to s' using γ and $c = f_{s'}(x||m_1)$.
   - there exists d' such that $d = f(d')$.
3. If V_{WI} accepts, P continues running $P_{ZKAoK}(x, w, \sigma_P)$ to compute m_2 and send it to V.
4. V runs V_{ZKAoK} on (m_1, c, m_2) and outputs the output of V_{ZKAoK}.

Figure 3: Simultaneous resettable ZK argument protocol Π_{srZK} in the correlated randomness model

The proof of resettable soundness goes as follows. We first consider an experiment with an imaginary protocol Π_F where a truly random function is used instead of the PRF, and the verifier uses an alternate witness for the sim-res WI. We will show that Π_F is resettably-sound by contradiction. Finally, we show that the probability that any resetting adversary can prove a false theorem in Π_{srZK} is negligible close to that of Π_F through a series of hybrids. This implies that Π_{srZK} is also resettably-sound.

Lemma 3.2. *The protocol Π_{srZK} in the correlated randomness model is resettably-sound.*

Proof. Let F be a truly random function. Consider an experiment Π_F in Figure 4.

Note that s_P generated in K_F is computationally indistinguishable from s_P generated in K. Furthermore, P behaves identically in the execution phase of Π_{srZK} and Π_F. We first prove that in Π_F, for any resetting prover $\widetilde{P}$, the probability that $\widetilde{P}$ makes V_F accepts $x \notin L$ in one of the resetting session is negligible.

Assume for contradiction that there exists a PPT resetting prover $\widetilde{P}$ that can prove a false statement $x \notin L$ in one of the resetting session of Π_F with non-negligible probability p. We construct a non-resetting prover $\widetilde{P}_{ZKAoK}$ that can prove a false statement $x \notin L$ in Π_{ZKAoK} as follows. Given σ_P, $\widetilde{P}_{ZKAoK}$ generates the additional parameters d_0, t, γ, d as in K_F. $\widetilde{P}_{ZKAoK}$ internally runs $\widetilde{P}$ on $s_P = (\sigma_P, t, d)$ in Π_F by either sending c from an honest verifier V_{ZKAoK} or

$$\Pi_F = (K_F, P, V_F)$$

K_F:

1. $(\sigma_P, \sigma_V) \leftarrow K_{ZKAoK}(1^\lambda)$, $d_0 \leftarrow U_{\ell_1(\lambda)}$. Let $t = \mathsf{com}(0^{\ell_0(\lambda)})$ and γ be the decommitment information, and $d = f(d_0)$.
2. K_F outputs $s_P = (\sigma_P, t, d)$ and $s_V = (\sigma_V, d_0, t, d)$.

Execution phase: P on input (x, w) and private string s_P; V_F on input x and private string s_V

1. P parses $s_P = (\sigma_P, t, d)$, runs $P_{ZKAoK}(x, w, \sigma_P)$ to compute m_1, and sends m_1 to V_F.
2. V_F parses $s_V = (\sigma_V, s, \gamma, t, d)$, runs $V_{ZKAoK}(x, \sigma_V)$ to send $c = F(x||m_1)$ on behalf of V_{ZKAoK} to P (running P_{ZKAoK}), and runs $P_{WI}(y, d_0)$, with $y = (t, c, d, x, m_1)$, proving to P running $V_{WI}(y)$ that one of the following statements hold
   - there exists s' and γ such that t can be decommitted to s' using γ and $c = f_{s'}(x||m_1)$.
   - there exists d' such that $d = f(d')$.
3. If V_{WI} accepts, P continues running $P_{ZKAoK}(x, w, \sigma_P)$ to compute m_2 and send it to V_F.
4. V_F runs V_{ZKAoK} on (m_1, c, m_2) and outputs the output of V_{ZKAoK}.

Figure 4: Experiment Π_F

generating c itself to $\widetilde{P}$ on behalf of V. At the beginning of the protocol, $\widetilde{P}_{ZKAoK}$ randomly selects $i \in [T]$ where T is the upper bound on the number of resetting sessions determined by $\widetilde{P}$. $\widetilde{P}_{ZKAoK}$ runs a verifier V_F for $\widetilde{P}$ for all but the ith resetting session. In the ith resetting session, $\widetilde{P}_{ZKAoK}$ passes the first message m_1 from $\widetilde{P}$ to V_{ZKAoK} and c from V_{ZKAoK} to $\widetilde{P}$. $\widetilde{P}_{ZKAoK}$ also provides a srWI argument to $\widetilde{P}$ using d_0 it generated as a witness. If $\widetilde{P}$ resets and sends the same m_1, $\widetilde{P}_{ZKAoK}$ will send the same c corresponding to the m_1 sent before. If $\widetilde{P}$ resets and sends a new m_1, $\widetilde{P}_{ZKAoK}$ will either pass m_1 to V_{ZKAoK} and relay c from V_{ZKAoK} to $\widetilde{P}$, or generate c itself. Since $\widetilde{P}$ cannot distinguish H_0 and H_3, it will also prove a false statement $x \notin L$ in one of the resetting session of Π_{srZK} with non-negligible probability $p' = p - \mathsf{negl}(\lambda)$. If $\widetilde{P}_{ZKAoK}$ guesses correctly which m_1 to pass to V_{ZKAoK} for $\widetilde{P}$ to complete the protocol for $x \notin L$, then it will convince V_{ZKAoK} to accept a false statement. This happens with probability $1/T$ where T is polynomial in the security parameter. Thus, $\widetilde{P}_{ZKAoK}$ can prove a false statement in Π_{ZKAoK} with probability p'/T which is non-negligible (in the security parameter). This contradicts the soundness of Π_{ZKAoK}. Hence, the protocol Π_F is resettably sound.

We then show that the original protocol Π_{srZK} is also resettably-sound by considering the following hybrid experiments whose outputs come from the joint distribution of the output of the verifier and the witness extractor $\mathcal{E}_2$: let $\widetilde{P}$ be a resetting prover.

H_0**:** This hybrid experiment runs the protocol Π_{srZK} $\langle \widetilde{P}, V \rangle$ with σ replaced by σ' generated in $(\sigma', \tau) \leftarrow \mathcal{E}_1(1^\lambda)$. If $\widetilde{P}$ convinces V to accept x in a resetting session, take the transcript (m_1, c, m_2) and run $\mathcal{E}_2(x, \tau, (m_1, c, m_2))$ to extract a witness w. Output (x, w) where $w = \perp$ if $\mathcal{E}_2$ fails to extract a witness, or abort if V rejects.

H_1**:** This hybrid is the same as H_0 except that d is generated by first sampling $d_0 \leftarrow U_{\ell_1(\lambda)}$ and computing $d = f(d_0)$, and also giving d_0 to V. By the property of the PRG, this hybrid is indistinguishable from H_0.

H_2**:** This hybrid is the same as H_1 except that V runs $P_{WI}(y, d_0)$ instead of $P_{WI}(y, s)$.

Claim. *Hybrid H_1 and H_2 are indistinguishable.*

Proof. Suppose a distinguisher D can distinguish H_1 and H_2 with non-negligible probability q. We construct $\widetilde{V}^*$ that can distinguish the interaction with $P_{WI}(y, (s, \gamma))$ and $P_{WI}(y, d_0)$ where $y = (t, c, d, x, m_1)$ with probability q as follows. $\widetilde{V}^*$ samples d_0, s and computes t, d, γ, m_1, c as in H_1, H_2. It then runs D while feeding the messages from $P_{WI}(x, (s, \gamma))$ or $P_{WI}(x, d_0)$ after sending c. □

H_3**:** This hybrid is the same as H_2 except that V is no longer given s, γ and instead sends $c \leftarrow F(x||m_1)$, where F is a truly random function. By the property of the PRF, this hybrid is indistinguishable from H_2.

H_4**:** This hybrid is the same as H_3 except that s is no longer generated and t is a commitment of $0^{\ell_0(\lambda)}$ instead of s. By the hiding property of the commitment scheme, this hybrid is indistinguishable from H_3.

Let $\epsilon_i = \epsilon_i(\lambda)$ be the (negligible) probability of distinguishing H_{i-1} from H_i for each i. Note that the Hybrid H_4 is the same as Hybrid H_0 but with protocol Π_F $\langle \widetilde{P}, V_F \rangle$ instead of Π_{srZK}. As we proved above, the probability that V_F accepts x but $\mathcal{E}_2$ cannot extract the witness $\Pr[H_4 = (x, \perp)]$ is negligible $\epsilon_0 = \epsilon_0(\lambda)$ by the property of the witness extractor $\mathcal{E}_{ZKAoK}$. Thus, $\Pr[H_0 = (x, \perp)] \leq \epsilon_0 + \epsilon_1 + \epsilon_2 + \epsilon_3 + \epsilon_4 = \epsilon'$ is negligible. Note that by the property of the PRF, the extractor $\mathcal{E}_2$ can fail to extract in the Hybrid H_0 with at most negligible probability $\epsilon_5 = \epsilon_5(\lambda)$. Therefore, for any resetting prover $\widetilde{P}$, the probability that $\widetilde{P}$ convinces V to accept $x \notin L$ is at most $\epsilon_5 + \epsilon'$ which is negligible.

□

Lemma 3.3. *The protocol Π_{srZK} is resettable ZK in the correlated randomness model with a straight-line simulator.*

Proof. Now we construct a zero-knowledge simulator $\mathcal{S}$ against a resetting verifier $\widetilde{V}$. $\mathcal{S}$ runs the simulator $\mathcal{S}_1$ for Π_0 to generate $(\sigma'_P, \sigma'_V, \tau)$. It gives σ'_V to $\widetilde{V}$ instead of honestly generated σ_V in the correlated randomness generation phase. $\mathcal{S}$ generates s, γ, t, d honestly. In the execution phase, $\mathcal{S}$ runs $\mathcal{S}_2(\tau)$ to generate m_1. $\mathcal{S}$ receives c from $\widetilde{V}$ and runs V_{WI} honestly. $\mathcal{S}$ continues running $\mathcal{S}_2(\tau)$ to generate m_2, and records the transcript (m_1, c, m_2) for later use. When $\widetilde{V}$ resets, $\mathcal{S}$ sends the recorded m_1. If $\widetilde{V}$ sends the previously seen c, $\mathcal{S}$ will send the corresponding recorded m_2. If $\widetilde{V}$ sends a new c, $\mathcal{S}$ aborts.

Claim. *The views of $\widetilde{V}$ interacting with an honest prover P and with $\mathcal{S}$ are indistinguishable.*

Proof. First consider a hybrid experiment where we replace $(\sigma_P, \sigma_V) \leftarrow K_{ZKAoK}(\lambda)$ with (σ'_P, σ'_V) from $(\sigma'_P, \sigma'_V, \tau) \leftarrow \mathcal{S}_1(\lambda)$, and m_1 and m_2 are generated by $\mathcal{S}_2(\tau)$. By the ZK property of Π_{ZKAoK}, $\widetilde{V}$ cannot distinguish this hybrid experiment from running Π_{ZK}. Now suppose that $\widetilde{V}$ can distinguish the hybrid experiment from running against the simulator $\mathcal{S}$ with non-negligible probability q. The only difference between the hybrid and the interaction with $\mathcal{S}$ is when $\widetilde{V}$ resets and sends different c. In this case, $\widetilde{V}$ needs to provide a sim-res WI for $y_1 = (t, c_1, d, x, m_1)$ and $y_2 = (t, c_2, d, x, m_1)$ with $c_1 \neq c_2$. We construct a PPT $\widetilde{P}_{WI}$ that can prove a false statement $y = (t, c, d, x, m_1) \notin R_{WI}$ as follows: $\widetilde{P}_{WI}$ generates the setup strings and runs the interaction above (with $(\mathcal{S}_1, \mathcal{S}_2)$). $\widetilde{P}_{WI}$ randomly chooses the session of sim-res WI to pass the WI prover message from $\widetilde{V}$ to V_{WI}. Since d is chosen uniformly at random, except with negligible probability, $d \neq f(d')$ for any $d' \in \{0,1\}^{\ell_1(\lambda)}$. Thus, if $y_1, y_2 \in R_{WI}$, there exists s' such that $t = \mathsf{com}(s')$ and $c_1 = c_2 = f_{s'}(x||m_1)$. At least one of the c's will make $y = (t, c, d, x, m_1) \notin R_{WI}$. The probability that $\widetilde{P}_{WI}$ passes the sim-res WI messages corresponding to such y to V_{WI} is at least $1/T'$, where T' is the number of different c's sent by $\widetilde{V}$. Therefore, the probability that $\widetilde{P}_{WI}$ can prove $y \notin R_{WI}$ is at least q/T', which contradicts the soundness of (P_{WI}, V_{WI}). □

Since $(\mathcal{S}_1, \mathcal{S}_2)$ is straight-line, $\mathcal{S}$ is also straight-line. □

This gives us the following theorem:

Theorem 3.4. *Assuming the existence of OWFs, there exists a simultaneous resettable ZK argument protocol in the correlated randomness model with a straight-line simulator.*[7]

4 MPC in the Correlated Randomness Model from OWFs

In this section, we construct a UC-secure MPC protocol in the correlated randomness model based on OWFs. The key ingredient is a protocol to generate unbounded number of OTs (from a bounded number of OTs that are received by every party as part of the correlated randomness) and the IPS transformation [IPS08]. In [IPS08], Ishai, Prabhakaran and Sahai construct an MPC protocol in the OT-hybrid model assuming only a PRG. This protocol requires a large number of OTs, proportional to the circuit size.

We first construct a UC-secure protocol for unbounded number of OTs in the correlated randomness model. At a very high level, the idea is as follows. We have the sender construct a super-polynomial size Yao's garbled circuit that computes the OTs. Instead of sending the circuit to the receiver, the sender commits to the first layer of the circuit and the seed for the PRF that is used to generate the rest of the circuit. When the receiver queries for the ith OT, the sender sends a section of the garbled circuit that suffices to compute the output followed by the ZK argument that it is consistent with committed values. However, this section of the circuit is now of polynomial size. More details of this construction are presented in Section 4.1. Using this, we then combine it with the [IPS08] transformation, to obtain our UC-secure MPC protocol in the correlated randomness model from OWFs (Section 4.2).

[7] Our ZK argument protocol also has a straight-line witness extractor, but it is not necessary for our purpose.

4.1 Unbounded Number of OTs in the Correlated Randomness Model

We first construct a UC-secure protocol computing unbounded number of OTs in the correlated randomness model assuming only OWFs. We make use of Beaver's OT setup [Bea95] and extension [Bea96] as follows. In [Bea96], the sender constructs a garbled circuit that takes a seed for a PRF as an input, and then expands this to a long string. Each bit of the string is matched with a random bit from the sender. In case of a match, a secret bit of the sender is revealed. Now, in order to get a garbled input corresponding to the receiver's seed and the garbled circuit, the sender and the receiver only need to perform a small number of OTs for each bit of the seed. This OT extension technique reduces λ^c OTs to λ OTs for any constant c and the security parameter λ. This small number of OTs can be precomputed [Bea95] as part of the correlated randomness.

Since our final goal is for each token to have a small-size memory, this OT extension does not suffice. We consider the following modification. Instead of sending the whole garbled circuit as in [Bea96], the sender first commits to another seed for the PRF whose output is used to generate the garbled circuit. The receiver uses the small number of OTs to obtain the garbled inputs associated to its seed. For each i, the receiver then sends the index i for the OT. The sender replies with a part of the garbled circuit that suffices to compute the ith OT along with UC-secure ZK argument that the part is computed correctly using the committed seed. This way, the garbled circuit is allowed to have super-polynomial size while the part for computing each OT is of polynomial size. Since the whole garbled circuit is fixed given the committed values, the sender cannot change the circuit and still successfully provide the ZK argument. Sender security is proved by arguing that the receiver does not learn more than the intended output by the property of the garbled circuits. More details follow.

Let $\{f_s\}_s$ be a family of PRFs. Let $\mathcal{G} = (\mathsf{GC}, \mathsf{GI}, \mathsf{GE})$ be a projective garbling scheme. Let $\mathsf{OT}_0 = (K_0, S_0, R_0)$ be the OT protocol in the correlated randomness model for small number of OTs. Let $\Pi_{ZK} = (K_{ZK}, P_{ZK}, V_{ZK})$ be a UC-secure ZK argument in the CRS model based on OWFs. We describe the unbounded OT protocol in Figure 5.

Theorem 4.1. *Assuming OWFs, the protocol in Figure 5 is a UC-secure protocol computing unbounded number of OTs in the correlated randomness model.*

Proof. (Sketch) We construct a simulator Sim in the ideal world running Adv as follows. Sim generates the correlated randomness for the small number of OTs for each party. In the case that Adv corrupts the receiver, Sim first commits to a zero string. When Adv queries the garbled input for its seed, Sim extracts the seed using the correlated randomness. For each i, Sim queries the OT functionality for the ith output, and generates a garbled circuit that takes the extracted seed and outputs the ith output from the functionality. Finally, Sim runs the simulator for the ZK argument instead of the ZK prover. In the case that Adv corrupts the sender, Sim uses the correlated randomness to extract both garbled inputs for each bit of the seed. Given the ith garbled input, it can learn both sender's inputs, and send them to the OT functionality.

The indistinguishability proof is through a series of hybrids. First, we use the simulator for the ZK argument instead of the ZK prover. Then, using the hiding property of com, we replace the commitment of the seed to the commitment of zero. Sim then generates the correlated randomness for the correlated randomness functionality, thereby learning the private string for both parties. Finally, Sim extracts the sender's garbled inputs and the receiver's seed and proceed as above, using the security of the garbled circuit. □

$$\mathsf{OT}^N = (K, S, R)$$

$K(1^\lambda)$:

1. $(\sigma_S, \sigma_R) \leftarrow K_0(1^\lambda)$, $\sigma_{ZK} \leftarrow K_{ZK}(1^\lambda)$, $s_1, s_2 \leftarrow U_{\ell(\lambda)}$. For $i = 1, 2$, let $c_i = \mathsf{com}(s_i)$ with decommitment information γ_i.
2. K outputs $s_S = (\sigma_S, \sigma_{ZK}, s_1, \gamma_1, c_2)$ and $s_R = (\sigma_R, \sigma_{ZK}, s_2, \gamma_2, c_1)$.

Execution phase: S on input $X = \{x_0^i, x_1^i\}_{i\in[N]}$, $x_b^i \in \{0,1\}$, and private string s_S; R on input $\{b_i\}_{i\in[N]}$, $b_i \in \{0,1\}$, and private string s_R

1. S parses $s_P = (\sigma_S, \sigma_{ZK}, s_1, c_2)$. Let $C_{X,s_1,c_2}(s_2, \gamma_2)$ be a circuit that computes $r = f_{s_1}(s_2)$ and uses Beaver's expansion described above to output $\{x_{b_i}^i\}_{i\in[N]}$ only if c_2 is decommitted to s_2 using γ_2. S samples $s_3 \leftarrow U_{\ell(\lambda)}$ and let $(G, \pi) = \mathsf{GC}(C; f(s_3))$ with G_i the part of G necessary to evaluate $x_{b_i}^i$. Let $c_3 = \mathsf{com}(s_3)$ with decommitment information γ_3. S sends c to R.
2. R parses $s_V = (\sigma_S, \sigma_{ZK}, s_2, c_1)$. S runs $S_0(\sigma_S)$ with R running $R_0(\sigma_R)$ to send $S = \mathsf{GI}(\pi, (s_2, \gamma_2))$, garbled inputs corresponding to (s_2, γ_2).
3. For each $i \in [N]$,
   (a) R sends i to S.
   (b) S sends G_i to R, S then runs $P_{ZK}(\sigma_{ZK}, (s_1, s_3, \gamma_1, \gamma_3)))$ to prove to R running $V_{ZK}(\sigma_{ZK})$ that G_i is generated using s_1 and s_3 that are committed earlier. R aborts if V_{ZK} rejects or G_i is inconsistent with previously received G_j for $j < i$.
   (c) R evaluates $x_{b_i}^i = \mathsf{GE}(G_i, S)$.

Figure 5: UC-Secure Unbounded OT Protocol

4.2 IPS Transformation

In [IPS08], Ishai *et al.* construct a compiler that turns an MPC protocol that is secure against adversary corrupting less than half of the parties (honest majority) into a UC-secure MPC protocol in the OT-hybrid model. They apply this transformation to a variant of the MPC protocol from [DI05] to obtain the following theorem.

Theorem 4.2 ([IPS08])**.** *Assuming a PRG, for any $n \geq 2$, there exists an n-party constant-round MPC protocol in the OT-hybrid model that is UC-secure against an active adversary adaptively corrupting at most $n - 1$ parties.*

Combining this theorem with our UC-secure protocol for unbounded number of OTs in the correlated randomness model, we get the following corollary.

Corollary 4.3. *Assuming a PRG and OWFs, for any $n \geq 2$, there exists an n-party constant-round MPC protocol in the correlated randomness model that is UC-secure against an active adversary adaptively corrupting at most $n-1$ parties.*

5 Corruptible Tamper-Proof Token Model

We consider a generalization of the Katz's tamper-proof token model [Kat07] where tokens can be corrupted by adversaries even when they are created by honest parties. Our model is inspired by the real world application where honest users cannot create tokens themselves. They instead rely on a number of manufacturers, some of whom could be malicious. Thus, the secrets embedded in the token description can be revealed to the adversary. Furthermore, the adversary can replace the tokens with ones of its choice.

5.1 Katz's Stateless Tamper-Proof Token Functionality $\mathcal{F}_{\mathsf{token}}$

Our model is based on the stateless version of Katz's tamper-proof token model [Kat07] defined in Figure 6. In this model, each user can create a stateless token by sending its description to $\mathcal{F}_{\mathsf{token}}$. The token is tamper-proof in the sense that the receiver can only access it through $\mathcal{F}_{\mathsf{token}}$ functionality in a black-box manner. We consider the case of stateless tokens where the tokens do not keep information between each access and use the same random tape. Hence, without loss of generality, we can assume that the function computed by the token is deterministic. In this case, we may represent the function with a circuit.

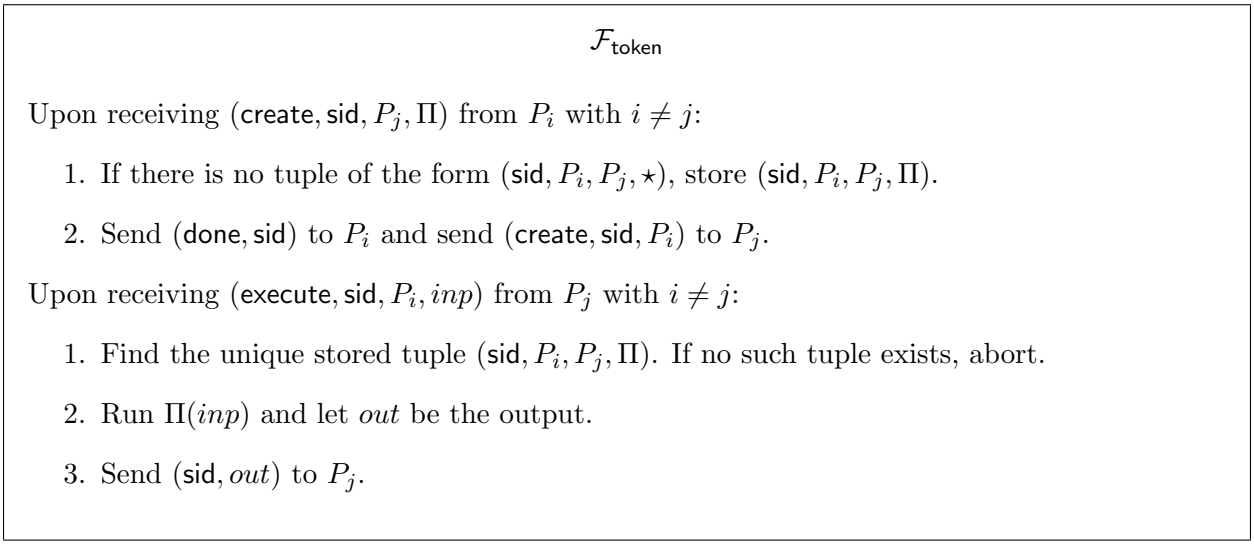

Figure 6: Token Functionality $\mathcal{F}_{\mathsf{token}}$

Our protocol however will UC-realize a variant of $\mathcal{F}_{\mathsf{token}}$, called $\mathcal{F}_{\mathsf{token}}^{\mathsf{abort}}$, described in Figure 7 in which the adversary is notified whenever a party creates a token and can choose to interrupt its delivery. The receiver will not receive the token, but will be notified with the special message $\mathsf{interrupted}$. In such a case, the receiver aborts the protocol. This change can be avoided by restricting the adversary to corrupt less than half of the corruptible tokens, which will allow the receiver

to compute the output using the remaining uncorrupted tokens, but will weaken the threshold of corruptions tolerated.

$\mathcal{F}_{\mathsf{token}}^{\mathsf{abort}}$

Upon receiving $(\mathsf{create}, \mathsf{sid}, P_j, \Pi)$ from P_i with $i \neq j$:

1. If there is no tuple of the form $(\mathsf{sid}, P_i, P_j, \star)$, store $(\mathsf{sid}, P_i, P_j, \Pi, \mathsf{creating})$.
2. Send $(\mathsf{create}, \mathsf{sid}, P_i, P_j)$ to Adv.

Upon receiving $(\mathsf{interrupt}, \mathsf{sid}, P_i, P_j)$ from Adv:

1. Find and remove the unique stored tuple $(\mathsf{sid}, P_i, P_j, \Pi, \mathsf{creating})$. If no such tuple exists, abort.
2. Send $(\mathsf{interrupted}, P_i)$ to P_j.

Upon receiving $(\mathsf{notinterrupt}, \mathsf{sid}, P_i, P_j)$ from Adv:

1. Find and remove the unique stored tuple $(\mathsf{sid}, P_i, P_j, \Pi, \mathsf{creating})$. If no such tuple exists, abort.
2. Store $(\mathsf{sid}, P_i, P_j, \Pi)$.
3. Send $(\mathsf{done}, \mathsf{sid})$ to P_i and send $(\mathsf{create}, \mathsf{sid}, P_i)$ to P_j.

Upon receiving $(\mathsf{execute}, \mathsf{sid}, P_i, inp)$ from P_j with $i \neq j$:

1. Find the unique stored tuple $(\mathsf{sid}, P_i, P_j, \Pi)$. If no such tuple exists, abort.
2. Run $\Pi(inp)$ and let out be the output.
3. Send (sid, out) to P_j.

Figure 7: Token with Abort Functionality $\mathcal{F}_{\mathsf{token}}^{\mathsf{abort}}$

5.2 Corruptible Tamper-Proof Token Functionality $\mathcal{F}_{\mathsf{token}}^{\mathsf{corruptible}}$

We generalize the tamper-proof token model to accommodate such a scenario by allowing an adversary to corrupt each token upon its creation. We define corruptible tamper-proof token functionality $\mathcal{F}_{\mathsf{token}}^{\mathsf{corruptible}}$ in Figure 8 by modifying $\mathcal{F}_{\mathsf{token}}$ as follows. Every time a user sends create command to the functionality $\mathcal{F}_{\mathsf{token}}^{\mathsf{corruptible}}$, it first notifies the adversary and waits for one of two possible responses. The adversary may choose to learn the description of the token, and replace it with another (possibly stateful) token of its choice. We call the token chosen by the adversary a *corrupted* token. Alternately, the adversary may ignore the creation of that token, and therefore, that token creation is completed successfully and in this case, the adversary will not learn the description of the token.

After uncorrupted tokens are created, they are tamper-proof in the same sense as in Katz's model. The stateful program for the corrupted token can be represented by a Turing machine.

In the case that the adversary chooses not to corrupt any token created by honest users, our model is identical to the model of Katz. Thus, our model generalizes the standard tamper-proof token model. In this work, we show that we can achieve UC-secure 2PC/MPC in the corruptible tamper-proof token model allowing the adversary to corrupt one party and all but one token generated by every honest party.

6 A Compiler from Katz's Model to the Corruptible Token Model

6.1 Protocol for corruptible tokens

In this section, we describe a multi-party protocol that the n corruptible tokens will run in order to emulate the Katz' stateless token functionality.

Let $(K_{srZK}, P_{srZK}, V_{srZK})$ be a simultaneous resettable ZK argument in the correlated randomness model with straight-line simulator. Let $\mathcal{S} = (\mathsf{share}, \mathsf{recon})$ be an n out of n secret sharing scheme. Let Γ_0 be a UC-secure MPC protocol in the correlated randomness model for functionality $\mathcal{F}$ described in Figure 9.

We define a multi-party protocol $\Gamma = \Gamma(\Pi)$ on input $(x_1, \ldots, x_n)$ to compute $\Pi(x)$ when $x = x_i$ for all $i \in [n]$ in Figure 10.

6.2 Realizing Tamper-Proof Token with Corruptible Tokens

Now we are ready to describe our protocol realizing the tamper-proof token with n corruptible tokens. To compute Π, the corruptible tokens are given the setup parameters for the MPC protocol $\Gamma(\Pi)$. Up on execution with input x_i, they will run $\Gamma(\Pi)$ to compute $\Pi(x)$ only if $x = x_i$ for all $i \in [n]$. Since the token receiver, possibly malicious, will deliver $\Gamma(\Pi)$ messages between the tokens we need an additional wrapper layer to encrypt the messages and the saved states of the tokens for authenticity.

Let $(\mathsf{SetUp}, \mathsf{Enc}, \mathsf{Dec})$ be a symmetric key encryption scheme. For $1 \leq i < j \leq n$, let $sk_{i,j} = sk_{j,i} \leftarrow \mathsf{SetUp}(1^\lambda)$. Let Γ_i be the Turing machine representing Party P_i in Γ with embedded secret keys $\{sk_{i,j}\}_{j \neq i}$ such that

$$\Gamma_i(\mathsf{state}_{k-1}, in) = (\mathsf{state}_k, out)$$

where state_k is the internal state of the Turing machine in round k and

- $in = (c_j)_{j \neq i}$ where $\mathsf{Dec}(sk_{i,j}, c_j)$ is the incoming message from Party j to Party i in round $k-1$.
- $out = (\mathsf{Enc}(sk_{i,j}, m_{j,k}))_{j \neq i}$ where $m_{i,j,k}$ is the corresponding outgoing message from Party i to Party j in round k (or 0 if Party i does not send a message to Party j in this round).

Let $T_i = T(s_i)$ be defined in Figure 11.

Finally, we define the protocol Λ in $\mathcal{F}_{\mathsf{token}}^{\mathsf{corruptible}}$-hybrid model realizing $\mathcal{F}_{\mathsf{token}}^{\mathsf{abort}}$ in Figure 12.

$\mathcal{F}_{\mathsf{token}}^{\mathsf{corruptible}}$

Upon receiving $(\mathsf{create}, \mathsf{sid}, P_j, \Pi)$ from P_i with $i \neq j$:

1. If there is no tuple of the form $(\mathsf{sid}, P_i, P_j, \star, \star)$, store $(\mathsf{sid}, P_i, P_j, \Pi, \mathsf{creating})$.
2. Send $(\mathsf{create}, \mathsf{sid}, P_i, P_j)$ to Adv.

Upon receiving $(\mathsf{corrupt}, \mathsf{sid}, P_i, P_j)$ from Adv:

1. Find the unique stored tuple $(\mathsf{sid}, P_j, P_i, \Pi, \mathsf{creating})$. If no such tuple exists, abort.
2. Send Π to Adv.

Upon receiving $(\mathsf{replace}, \mathsf{sid}, P_i, P_j, \Pi^*, \mathsf{state}_0)$ from Adv:

1. Find and remove the unique stored tuple $(\mathsf{sid}, P_j, P_i, \Pi, \mathsf{creating})$. If no such tuple exists, abort.
2. Store $(\mathsf{sid}, P_i, P_j, \Pi^*, \mathsf{state}_0)$, send $(\mathsf{done}, \mathsf{sid})$ to P_i and send $(\mathsf{create}, \mathsf{sid}, P_i)$ to P_j.

Upon receiving $(\mathsf{notcorrupt}, \mathsf{sid}, P_i, P_j)$ from Adv:

1. Find and remove the unique stored tuple $(\mathsf{sid}, P_j, P_i, \Pi, \mathsf{creating})$. If no such tuple exists, abort.
2. Store $(\mathsf{sid}, P_i, P_j, \Pi, \bot)$, send $(\mathsf{done}, \mathsf{sid})$ to P_i and send $(\mathsf{create}, \mathsf{sid}, P_i)$ to P_j.

Upon receiving $(\mathsf{execute}, \mathsf{sid}, P_i, inp)$ from P_j with $i \neq j$:

1. Find the unique stored tuple $(\mathsf{sid}, P_i, P_j, \Pi, \mathsf{state})$. If no such tuple exists, abort.
2. Run $\Pi(\mathsf{state}, inp)$ and let (state', out) be the output. If $\mathsf{state} \neq \bot$, set $\mathsf{state} = \mathsf{state}'$.
3. Send (sid, out) to P_j.

Upon receiving $(\mathsf{read}, \mathsf{sid}, P_i, P_j)$ from Adv:

1. Find the unique stored tuple $(\mathsf{sid}, P_j, P_i, P, \mathsf{state})$. If no such tuple exists, abort.
2. Send $(\mathsf{sid}, \mathsf{state})$ to Adv.

Figure 8: Token Functionality $\mathcal{F}_{\mathsf{token}}^{\mathsf{corruptible}}$

$\mathcal{F}$

On input $((\Pi_1, x_1), \ldots, (\Pi_n, x_n))$

1. If $x_i \neq x_j$ for some $i \neq j$, output $\perp$. Otherwise, let x be the common input.
2. $\Pi = \mathsf{recon}(\Pi_1, \ldots, \Pi_n)$.
3. Output $\Pi(x)$.

Figure 9: Function $\mathcal{F}$

6.3 Proof of Security

Let $\mathcal{S}_{srZK} = (\mathcal{S}_1, \mathcal{S}_2)$ be the straight-line simulator for the simultaneous resettable ZK, and Sim_{MPC} be the UC simulator for the UC-secure MPC.

Let Adv be an adversary corrupting up to $n-1$ tokens. Let n_c be the number of corrupted tokens, and $n_h = n - n_c$ be the number of honest (uncorrupted) tokens. We construct a UC simulator Sim in Figure 13 internally running Adv such that any environment $\mathcal{E}$ cannot distinguish between interacting with Adv running Λ in the real world and interacting with Sim running $\mathcal{F}_{\mathsf{token}}^{\mathsf{abort}}$ in the ideal world.

Now consider the series of hybrids:

Hybrid H_0: This hybrid is the real world execution.

Hybrid H_1: This hybrid is similar to H_0 except that every message to $\mathcal{F}_{\mathsf{token}}^{\mathsf{corruptible}}$ goes to Sim, and Sim acts honestly on behalf of $\mathcal{F}_{\mathsf{token}}^{\mathsf{corruptible}}$ while recording the messages. This hybrid is identical to H_0.

Let Hybrid $H_{2.0} = H_1$. For $k = 1, \ldots, n_h \cdot n_c$,

Hybrid $H_{2.k}$: This hybrid is similar to $H_{2.(k-1)}$ except that Sim uses $\mathcal{S}_1$ to generate $(\sigma_{i,j,P}, \sigma_{i,j,V}, \tau_{i,j})$ instead of K_{srZK} for honest token i and corrupted token j with $k = i(n_c - 1) + j$, and runs $\mathcal{S}_2(\tau_{i,j})$ to generate the sim-res ZK messages for token i by feeding the sim-res ZK messages from corrupted tokens. Sim records the transcript leading to each sim-res ZK session.

Lemma 6.1. *Hybrid $H_{2.(k-1)}$ and $H_{2.k}$ are indistinguishable.*

Proof. Suppose there exists a poly-time D that can distinguish $H_2.(k-1)$ and $H_2.k$ with non-negligible probability. We construct a distinguisher D' that can distinguish an interaction of P_{srZK} with a resetting verifier V^*_{srZK} and $\mathcal{S}_2(\tau_{i,j})$ for the sim-res ZK as follows. Given setup strings for the sim-res ZK, D' generates the setup for other pairs of tokens and the inputs for $\Gamma(\Pi)$. D' then runs $H_2.(k-1)$ or $H_2.k$ until Adv queries the honest token to prove a statement x using the sim-res ZK. D' runs the interaction and passes the messages from and to Adv as V^*_{srZK}'s messages. When V^*_{srZK} resets P_{srZK} or $\mathcal{S}_2(\tau_{i,j})$, D' queries the token using the saved state of the earlier round in the sim-res ZK. Finally, D' outputs the output of D. □

$\Gamma(\Pi)$

Setup:

1. Let $(\Pi_1, \ldots, \Pi_n) \leftarrow \mathsf{share}(\Pi)$. Let $c_i = \mathsf{com}(\Pi_i)$ for $i \in [n]$.
2. For $i \neq j \in [n]$, let $(\sigma_{i,j,P}, \sigma_{i,j,V}) \leftarrow K_{srZK}(1^\lambda)$ be the correlated randomness for the simultaneous resettable ZK argument with prover P_i and verifier P_j.
3. Let $(\sigma_1, \ldots, \sigma_n) \leftarrow \mathsf{SetUp}_{\Gamma_0}(1^\lambda)$ be the correlated randomness for Γ_0.
4. For $i \in [n]$, send $\Sigma_i = (\Pi_i, \sigma_i, \{c_j, \sigma_{i,j,P}, \sigma_{j,i,V}\}_{j \neq i})$ to Party P_i.

Execution: Party P_i on input x_i, correlated string Σ_i, and a random tape R_i

1. For each $i \in [n]$, P_i generates a seed s_i for the PRF from $R_i[0]$, and sends the determining message $M_i = \mathsf{com}(\alpha_i; R_i[1])$ where $\alpha_i = (x_i, s_i)$.
2. For each $i \in [n]$, P_i computes $R'_i = PRF_{s_i}(M_1 || \ldots || M_n)$ consisting of $R'_i[0]$ for running Γ_0 and $R'_i[1]$ for running (P_{srZK}, V_{srZK}).
3. For each $i \in [n]$, P_i executes as the ith party in Γ_0 where P_i follows kth round message $m_{i,k}$ by running P_{srZK} to prove that there exists $\alpha_i = (x_i, s_i)$ and Π_i such that
   (a) M_i can be decommitted to α_i;
   (b) c_i can be decommitted to Π_i;
   (c) $m_{i,k}$ is correctly computed using $R'_i[0]$ in Γ_0 with (Π_i, x_i) as an input where $R'_i = PRF_{s_i}(M_1 || \ldots || M_n)$.

Figure 10: Multi-party protocol $\Gamma(\Pi)$ computing Π

Claim. *Fixed a combined determining message $M = M_1 || \ldots || M_n$, any polynomial-time resetting machine* Adv *can find only one transcript of Γ_0 in $\Gamma(\Pi)$ that every following sim-res ZK argument convinces the verifier to accept.*

Proof. Suppose not. Let $tr = (\ldots, m_{i,k})$ and $tr' = (\ldots, m'_{i,k})$ be the partial transcripts of Γ_0 generated by Adv up to the differing messages $m_{i,k}, m'_{i,k}$ with accepting sim-res ZK argument. Note that we cannot have both $(c_i, M_i, M, tr), (c_i, M_i, M, tr') \in R_{rsZK}$. Otherwise, either M_i or c_i can be decommitted to two different values, and thus can be reduced to the security of the commitment scheme. Hence, either $(c_i, M_i, M, tr) \notin R_{rsZK}$ or $(c_i, M_i, M, tr') \notin R_{rsZK}$. Thus, we can construct a resetting prover P^*_{srZK} that can prove a false statement. □

Let Hybrid $H_{3.0} = H_{2.(n_h \cdot n_c)}$. Let m be the number of distinct sessions of Γ_0 based on combined determining message $M_1 || \ldots || M_n$ generated through Adv querying the tokens. For $k = 1, \ldots, m$,

On input $(\mathsf{Initialize}, x_i)$

1. Parse $s_i = (i, sk_i, R_i, \Pi_i, \sigma_i, \{c_j, \sigma_{i,j,P}, \sigma_{j,i,V}, sk_{i,j}\}_{j\in[n]})$.
2. Initiate Γ_i on setup parameters $(\Pi_i, \sigma_i, \{c_j, \sigma_{i,j,P}, \sigma_{j,i,V})\}_{j\in[n]})$, random tape R_i and input x_i.
3. Let msg be the first message from Γ_i and let state_0 be its initial state.
4. Output $(msg, \mathsf{Enc}(sk_i, \mathsf{state}_0))$.

On input $(msg, \overline{\mathsf{state}}_i)$

1. Parse $s_i = (i, sk_i, R_i, \Pi_i, \sigma_i, \{c_j, \sigma_{i,j,P}, \sigma_{j,i,V}, sk_{i,j}\}_{j\in[n]})$.
2. Initiate Γ_i on setup parameters $(\Pi_i, \sigma_i, \{c_j, \sigma_{i,j,P}, \sigma_{j,i,V})\}_{j\in[n]})$, random tape R_i and input x_i.
3. Decrypt $\mathsf{state}_i = \mathsf{Dec}(sk_i, \overline{\mathsf{state}}_i)$, abort if fail.
4. Load Γ_i with state state.
5. Feed Γ_i, msg and let out be its reply and state' be its new state.
6. Output $(out, \mathsf{Enc}(sk_i, \mathsf{state}'))$.

Figure 11: Token $T_i = T(s_i)$

Hybrid $H_{3.k}$: This hybrid is similar to $H_{3.(k-1)}$ except that Sim runs Sim_{MPC} to generate the MPC messages for uncorrupted tokens by feeding the MPC messages from corrupted tokens in the execution of $\Gamma(\Pi)$ following kth combined determining message.

Lemma 6.2. *Hybrid $H_{3.(k-1)}$ and $H_{3.k}$ are indistinguishable.*

Proof. Suppose there exists a poly-time D that can distinguish $H_{3.(k-1)}$ and $H_{3.k}$ with non-negligible probability. We construct a distinguisher D' for Sim_{MPC} as follows. Given the correlated randomness for the MPC, D' generates the rest of the setup parameters for $\Sigma(\Pi)$ as in the experiment. D' then passes the MPC messages from Adv to D followed by the srZK messages from Sim_{ZK}. Since the accepting transcript is unique by the claim above, Adv cannot change the messages. D' outputs the output of D. □

Hybrid H_4: This hybrid is similar to $H_{3.m}$ except that Sim passes token creation request from honest parties to $\mathcal{F}_{\mathsf{token}}^{\mathsf{abort}}$ and uses it to compute the output for Sim_{MPC}.

Lemma 6.3. *Hybrid $H_{3.m}$ and H_4 are indistinguishable.*

Proof. Note that if Adv generates messages for the MPC honestly using the same input x_i and the share Π_i given in the setup, then the output from $\mathcal{F}_{\mathsf{token}}^{\mathsf{abort}}$ must be the same as the output of

Λ

To create a token running Π for P_j, P_i does the following:

1. Generate the setup parameters for $\Gamma(\Pi)$: $(\Pi_k, \sigma_k, \{c_l, \sigma_{k,l,P}, \sigma_{l,k,V})\}_{l\in[n]})$ for $k \in [n]$ as defined in Figure 10.
2. Generate secret keys for decrypting share/state $sk_k \leftarrow \mathsf{KeyGen}(1^\lambda)$ for all $k \in [n]$.
3. Generate secret keys for secure channel between token k and l $sk_{k,l} \leftarrow \mathsf{KeyGen}(1^\lambda)$ for all $k, l \in [n]$.
4. Let $s_k = (k, sk_k, R_k, \Pi_k, \sigma_k, \{c_l, \sigma_{k,l,P}, \sigma_{l,k,V}, sk_{k,l}\}_{l\in[n]})$.
5. Send $(\mathsf{create}, \mathsf{sid}_k, P_j, T_k)$ to $\mathcal{F}_{\mathsf{token}}^{\mathsf{corruptible}}$ where $T_k = T(s_k)$ for $k \in [n]$.

To execute a token running Π sent by P_i, P_j does the following:

1. For $k \in [n]$, initialize T_k by sending $(\mathsf{execute}, \mathsf{sid}_k, S, (\mathsf{initialize}, inp))$ to $\mathcal{F}_{\mathsf{token}}^{\mathsf{corruptible}}$ to compute $T_k(\mathsf{initialize}, inp) = (\overline{\mathsf{state}}_k, out_k)$.
2. While $\overline{\mathsf{state}}_k \neq \mathsf{done}$ for all $k \in [n]$, for $k \in [n]$
   (a) Parse $out_k = (c_{k,l})_{l\neq k}$. Let $in_k = (c_{l,k})_{l\neq k}$.
   (b) Send $(\mathsf{execute}, \mathsf{sid}_k, P_i, (\overline{\mathsf{state}}_k, in_k))$ to $\mathcal{F}_{\mathsf{token}}^{\mathsf{corruptible}}$ to compute $T_k(\overline{\mathsf{state}}_k, in_k) = (\overline{\mathsf{state}}'_k, out_k)$.
   (c) Replace $\overline{\mathsf{state}}_k$ by $\overline{\mathsf{state}}'_k$.
3. Let $out = out_k$ for $k \in [n]$ such that $\overline{\mathsf{state}}_k = \mathsf{done}$.

Figure 12: Protocol Λ in $\mathcal{F}_{\mathsf{token}}^{\mathsf{corruptible}}$-hybrid model UC realizing $\mathcal{F}_{\mathsf{token}}^{\mathsf{abort}}$

the MPC by the correctness of the MPC. Suppose there exists a poly-time D that can distinguish H_3 and H_4 with non-negligible probability p. There must be at least one MPC message m^* from Adv that is not generated honestly. Thus, we construct a resetting prover P^*_{srZK} for the sim-res ZK argument following m^* by randomly choosing a Γ_0 message and passing the following prover messages to V. When Adv sends a different message using the same token state, P^*_{srZK} resets the verifier. It has at least $1/T$ probability of choosing m^* where T is the number of Γ_0 messages sent by Adv. Thus, it has at least p/T probability of proving a false statement, contradicting the resettable soundness of the sim-res ZK. □

Hybrid H_5: This hybrid is similar to H_4 except that Sim generates secret share of zero string $0^{|\Pi|}$ instead of the one received from an honest party. This hybrid is the ideal world execution.

Lemma 6.4. *Hybrid H_4 and H_5 are indistinguishable.*

Sim

Whenever Sim receives $(\mathsf{create}, \mathsf{sid}, P_i, P_j)$ from $\mathcal{F}_{\mathsf{token}}^{\mathsf{abort}}$, Sim does the followings:

1. For each $k \in [n]$, send $(\mathsf{create}, \mathsf{sid}_k, P_i, P_j)$ to Adv.
2. If Adv replies with $(\mathsf{corrupt}, \mathsf{sid}_k, P_i, P_j)$ for any $k \in [n]$,
   (a) Follow the protocol of Λ for creating a token, except that Sim uses zero string $0^{|\Pi|}$ instead of the actual token Π to create secret shares $\Pi_1, \ldots, \Pi_n$, and uses $\mathcal{S}_1$ to generate $(\sigma_{i,j,P}, \sigma_{i,j,V}, \tau_{i,j})$ instead of K_{srZK} and Sim_{MPC} to generate σ_i instead of $\mathsf{SetUp}_{\Gamma_0}$ for the setup parameters of Γ for the corruptible tokens. Sim stores the secret shares for later comparison.
   (b) Send T_k to Adv for each k Adv chose to corrupt.
   (c) Store $(\mathsf{replace}, \mathsf{sid}_k, P_i, P_j, T'_k, \mathsf{state}_k)$ from Adv.
   (d) Send $(\mathsf{interrupt}, \mathsf{sid}, P_i, P_j)$ to $\mathcal{F}_{\mathsf{token}}^{\mathsf{abort}}$.
3. Otherwise, send $(\mathsf{notinterrupt}, \mathsf{sid}, P_i, P_j)$ to $\mathcal{F}_{\mathsf{token}}^{\mathsf{abort}}$.

Whenever Adv runs the protocol for execution that involves both corrupted and uncorrupted tokens, Sim does the followings:

1. Sim generates the Γ messages for uncorrupted T_k using $\mathcal{S}_2(\tau_k)$ and Sim_{MPC} as follows:
   (a) Sim generates and commits to α_k honestly as in Γ.
   (b) Sim runs Sim_{MPC} to generate messages for Γ_0.
   (c) Sim For each message generated by Sim_{MPC}, runs $\mathcal{S}_2(\tau_k)$ to generate messages for the following sim-res ZK argument.
   (d) When Sim_{MPC} queries the functionality of the function F on input $((\Pi'_1, x'_1), \ldots, (\Pi'_n, x'_n))$, if x'_k's are all equal to x', send $(\mathsf{execute}, \mathsf{sid}, P_i, x)$ to $\mathcal{F}_{\mathsf{token}}^{\mathsf{abort}}$ and passes the output to Sim_{MPC}. Otherwise, Sim aborts.
2. Sim encrypts the messages and states as in Λ
3. Sim records all inputs/outputs to the tokens. If Adv queries with the same input (state and incoming messages), Sim returns the recorded output (new state and outgoing messages).

Figure 13: UC Simulator Sim for Λ

Proof. Suppose there exists a poly-time D that can distinguish H_4 and H_5 with non-negligible probability. We construct a distinguisher D' for the secret sharing scheme $\mathcal{S}$ as follows. D' runs the experiment for D until it is given Adv shares consisting of less than n shares. D' then continues the experiment and D distinguishes between H_4 and H_5. Using the result of D, D' can distinguish between less than n shares of 0 and some program Π, contradicting the security of $\mathcal{S}$. □

Finally, we have proved our main theorem.

Theorem 6.5. *Assuming an existence of OWFs, there exists a protocol with n corruptible tokens in $\mathcal{F}_{\mathsf{token}}^{\mathsf{corruptible}}$-hybrid model UC-realizing $\mathcal{F}_{\mathsf{token}}^{\mathsf{abort}}$.*

7 Obfuscation with stateless hardware token from OWFs

7.1 Very high level description of protocol

At a very high level, the protocol of Nayak *et al.* makes use of OWFs and CRHFs. First, they make use of CRHFs for the authenticated ORAM structure. We observe that we can replace the authenticated ORAM used in Nayak *et al.* with an authenticated ORAM based on OWF (that can be built from the work of Ostrovsky and Goldreich, Ostrovsky). Next, in order to obtain a single starting seed for randomness that depends on the specific execution of the program and input, Nayak *et al.* require the user to first feed in a hash of the input to the token and then use this hash to derive all randomness (along with a unique program id). This gives them a unique execution id. We derive a unique value based on the input and program by having the user feed the input one-by-one to the token. Upon receiving one input, the token will authenticate it and provide an authentication tag (this process is deterministic) that will then allow the user to input the next input. This process continues until the last input is inserted into the token, upon which the authentication tag produced at this stage is a unique id that can be used (in combination with the program id) to derive all randomness needed by the token for program execution. This process is similar in spirit to the GGM construction of deriving a PRF from a PRG.

7.2 High level description of protocol

Program Authentication. At a high level, the program creation by the sender works as follows. Let the program to be obfuscated be $\mathsf{RAM} := (\mathsf{cpustate}, \mathsf{mem})$ where mem is a list of program instructions and $\mathsf{cpustate}$ is the initial cpu state. Let the program comprise of t instructions. The sender first creates the token containing a hardwired secret key K where $K := (K_e, K_{\text{prf}})$. K_e is used as the encryption key for encrypting state, K_{prf} is used as the key to a pseudorandom function used by the token to generate all randomness needed for executing the ORAM, creating ciphertexts, and so on. The sender creates a unique execution identity $\mathsf{id}_{\text{exec}}$, which is unique for every program created. The sender then encrypts $\mathsf{mem}||\mathsf{id}_{\text{exec}}||\mathsf{id}_{\text{instr}}$ (one instruction at a time) to obtain $\overline{\mathsf{mem}}$ ($\overline{\mathsf{mem}}_i$ denotes the ciphertext obtained upon encrypting $\mathsf{mem}_i||\mathsf{id}_{\text{exec}}||i$). The sender also computes a "tag" of the start ciphertext, $\overline{\mathsf{mem}}_1$, as $\tau_1 = \mathsf{PRF}_{K_{\text{prf}}}(\mathtt{start}, \overline{\mathsf{mem}}_1)$. The sender creates an encrypted program header $\overline{\mathsf{Header}} := \mathsf{Enc}_{K_e}(\mathsf{cpustate}, \mathsf{id}_{\text{exec}}, t)$. The receiver is sent $\overline{\mathsf{mem}}, \overline{\mathsf{mem}}_1^*$ and $\overline{\mathsf{Header}}$ as the obfuscated program.

Program Feed. At a very high level, the receiver will feed in the program, one instruction at a time, to the token, as follows:

1. As the first message, the token receives $(\texttt{programauth}, 1, \overline{\mathsf{mem}}_1, \tau_1, \overline{\mathsf{mem}}_2, \overline{\mathsf{Header}})$. It will check that $\mathsf{PRF}_{K_{\text{prf}}}(\texttt{start}, \overline{\mathsf{mem}}_1, \overline{\mathsf{Header}}) = \tau_1$ and output $\perp$ otherwise. Similarly, for all $2 \leq i < t-1$, it receives $(\texttt{programauth}, i, \tau_{i-1}, \overline{\mathsf{mem}}_i, \tau_i, \overline{\mathsf{mem}}_{i+1}, \overline{\mathsf{Header}})$. It will check that $\mathsf{PRF}_{K_{\text{prf}}}(\tau_{i-1}, \overline{\mathsf{mem}}_i, \overline{\mathsf{Header}}) = \tau_i$ and output $\perp$ otherwise.

2. Next, it decrypts $\overline{\mathsf{mem}}_i$ and $\overline{\mathsf{mem}}_{i+1}$ to get mem_i and mem_{i+1}, and $\mathsf{id}_{\text{exec}}$ as well as decrypt $\overline{\mathsf{Header}}$ to get $\mathsf{id}_{\text{exec}}$ and t. It will check that $\mathsf{id}_{\text{instr}} = i$ and $i+1$ respectively (also that these values are $\leq t$) and that the two $\mathsf{id}_{\text{exec}}$ values are the same and equal to the $\mathsf{id}_{\text{exec}}$ value in $\overline{\mathsf{Header}}$. If these checks do not pass, it will respond with $\perp$. If the checks pass, the token will output $\tau_{i+1} = \mathsf{PRF}_{K_{\text{prf}}}(\tau_i, \overline{\mathsf{mem}}_{i+1}, \overline{\mathsf{Header}})$.

Input Feed. Let the input to the program be denoted by $x_1, \cdots, x_n$. The receiver will send the following instructions, step-by-step, for every input, to the token.

1. On input, $(\texttt{inputauth}, 1, \tau_{t-1}, \overline{\mathsf{mem}}_t, \tau_t, x_1, n, \overline{\mathsf{Header}})$, it checks that $\mathsf{PRF}_{K_{\text{prf}}}(\tau_{t-1}, \overline{\mathsf{mem}}_t, \overline{\mathsf{Header}}) = \tau_t$, that $\overline{\mathsf{mem}}_t$ is the t^{th} program instruction (by decrypting $\overline{\mathsf{mem}}_t$ to get $\mathsf{id}_{\text{instr}}$ and $\overline{\mathsf{Header}}$ to get t and comparing) and output $\perp$ otherwise. It outputs $\tau_{t+1} = \mathsf{PRF}_{K_{\text{prf}}}(\tau_t, 1, x_1, n, \overline{\mathsf{Header}})$.

2. On input, $(\texttt{inputauth}, j, \tau_{t+j-2}, x_{j-1}, \tau_{t+j-1}, x_j, n, \overline{\mathsf{Header}})$, for $2 \leq j \leq n$, it checks that $\mathsf{PRF}_{K_{\text{prf}}}(\tau_{t+j-2}, j-1, x_{j-1}, n, \overline{\mathsf{Header}}) = \tau_{t+j-1}$ and outputs $\perp$ otherwise. It then outputs $\tau_{t+j} = \mathsf{PRF}_{K_{\text{prf}}}(\tau_{t+j-1}, j, x_j, n, \overline{\mathsf{Header}})$.

Program/Input ORAM Insertion. Once the program and input authentication is done, the program and input, henceforth collectively referred to as memory, must be inserted into the Authenticated Oblivious RAM structure. There are t program instructions and n inputs that must be inserted. Let $\ell = t + n$ be the total memory requirement of the program (we can assume this without loss of generality as any additional memory needed by the program can be thought of as dummy program instructions). First, a set of ℓ "zeroes" are inserted into the ORAM structure (i.e., the values of memory in all locations is set to 0)[8]. The insertion of a set of ℓ "zeroes" into the ORAM structure is done as follows:

1. For every memory location $1 \leq i \leq \ell$, the user prepares the message $(\texttt{ORAMsetup}, i, \ell, \tau_{i-1}^{\text{oraminit}}, n, x_n, \overline{\mathsf{Header}}, \tau_i^{\text{oraminit}}, \tau_\ell)$ and gives it to the token, with $\tau_1^{\text{oraminit}} = \tau_\ell$ and $\tau_0^{\text{oraminit}} = \tau_{\ell-1}$. The token checks that $\mathsf{PRF}_{K_{\text{prf}}}(\tau_0^{\text{oraminit}}, n, x_n, n, \overline{\mathsf{Header}}) = \tau_1^{\text{oraminit}}$ (for $i = 1$) and $\mathsf{PRF}_{K_{\text{prf}}}(\text{ORAMsetup}, i, \ell, \overline{\mathsf{Header}}, \tau_{i-1}^{\text{oraminit}}, \tau_\ell) = \tau_i^{\text{oraminit}}$ (for all other i) and outputs $\perp$ otherwise.

2. Otherwise, the token derives a key for the ORAM structure − this ORAM key is derived as $K_{\text{oram}} = \mathsf{PRF}_{K_{\text{prf}}}(\texttt{ORAMKey}, \tau_\ell)$.

[8]Whenever, the state of the program (ORAM or otherwise) needs to be modified, this is done by appending encrypted `state` with $\overline{\mathsf{Header}}$ and then authenticating, similar to Nayak *et al.* [NFR+17]

(a) It creates an ORAM initialization structure (that is, creates an initial random mapping of all virtual addresses to their real address); this initialization is done using randomness from the ORAM key K_{oram}.

(b) In this map let address a_j have been mapped to address i. In this case, the token creates an authenticated encryption of $(a_j, 0)$ (again using keys and randomness derived from K_{oram}) to be inserted into the ORAM structure at virtual address i.

(c) The token then outputs $\tau_{i+1}^{\text{oraminit}} = \mathsf{PRF}_{K_{\text{prf}}}(\text{ORAMsetup}, i, \ell, \overline{\mathsf{Header}}, \tau_i^{\text{oraminit}}, \tau_\ell)$.

Once all ℓ memory locations have been inserted with 0 values, the user then inserts the real input and program into the ORAM structure. This is done as follows: in the reverse order, starting with the n^{th} input to the first input, and then the t^{th} to the first program instruction. We now describe this process at a high level. For ease of exposition, we shall assume that every ORAM operation is a single step denoted as $\mathsf{oram}_{\sigma,K_{\text{oram}}}(i, v_i, \texttt{read/write}, \perp/v_i^*)$ (this can be easily extended to the case when the ORAM read/write is a set of operations, similar to Nayak *et al.* [NFR+17]). The protocol is as follows:

1. The user will insert the i^{th} memory location ($\ell \geq i \geq 1$, which is either an input or a program instruction) by sending the message $(\text{MemORAMInsert}, i, \ell, \tau_{2\ell-i}^{\text{oraminit}}, n, x_n, n, \overline{\mathsf{Header}}, \tau_{2\ell-i+1}^{\text{oraminit}}, \tau_i, \tau_{i-1}, w_i)$, where $w_i = (i - t, x_{i-t}, n)$ if the i^{th} location has an input (i.e., $\ell \geq i \geq t+1$) and $w_i = \overline{\mathsf{mem}}_i$ if the i^{th} location has a program instruction (i.e., $t \geq i \geq 1$).

2. If the i^{th} location has an input:

   (a) The token will check that $\tau_i = \mathsf{PRF}_{K_{\text{prf}}}(\tau_{i-1}, i-t, x_{i-t}, n, \overline{\mathsf{Header}})$ and that $\tau_{2\ell-i+1}^{\text{oraminit}} = \mathsf{PRF}_{K_{\text{prf}}}(\text{ORAMsetup}, i, \ell, \overline{\mathsf{Header}}, \tau_{2\ell-i}^{\text{oraminit}}, \tau_\ell)$.

   (b) The token will then execute the ORAM instruction $\mathsf{oram}_{\sigma,K_{\text{oram}}}(i, 0, \texttt{write}, x_{i-t})$.

   (c) The token then outputs $\tau_{2\ell-i+2}^{\text{oraminit}} = \mathsf{PRF}_{K_{\text{prf}}}(\text{ORAMsetup}, i-1, \ell, \overline{\mathsf{Header}}, \tau_{2\ell-i+1}^{\text{oraminit}}, \tau_\ell)$.

3. If the i^{th} location has a program instruction:

   (a) The token will check that $\tau_i = \mathsf{PRF}_{K_{\text{prf}}}(\tau_{i-1}, \overline{\mathsf{mem}}_i, \overline{\mathsf{Header}})$ and that $\tau_{2\ell-i+1}^{\text{oraminit}} = \mathsf{PRF}_{K_{\text{prf}}}(\text{ORAMsetup}, i, \ell, \overline{\mathsf{Header}}, \tau_{2\ell-i}^{\text{oraminit}}, \tau_\ell)$.

   (b) The token decrypts $\overline{\mathsf{mem}}_i$ to get $\mathsf{mem}_i||\mathsf{id}_{\text{exec}}||i$ and executes the ORAM instruction $\mathsf{oram}_{\sigma,K_{\text{oram}}}(i, 0, \texttt{write}, \mathsf{mem}_i||\mathsf{id}_{\text{exec}}||i)$.

   (c) The token then outputs $\tau_{2\ell-i+2}^{\text{oraminit}} = \mathsf{PRF}_{K_{\text{prf}}}(\text{ORAMsetup}, i-1, \ell, \overline{\mathsf{Header}}, \tau_{2\ell-i+1}^{\text{oraminit}}, \tau_\ell)$.

Program Execution. The program execution works in a similar manner to that of Nayak *et al.*

8 Practical Tokens

In this section, we improve the corruptible token protocol in Section 6 to ensure that the tokens created in the protocol only require small memory (for the embedded secret parameters) and only take short strings as inputs. We apply the technique in Section 7 to the construction, and thus the resulting protocol is still based on OWFs.

We consider a variant of $\mathcal{F}_{\mathsf{token}}^{\mathsf{corruptible}}$ in Figure 8, called $\mathcal{F}_{\mathsf{token}}^{\mathsf{corruptible,short},L_1,L_2}$ in Figure 14 where create and execute only take Π and inp of short size. We also allow a token sender to send a message along with the token created through the functionality. This allows the adversary to intercept the message when it chooses to corrupt a token without neither sender nor receiver knowledge. This is unavoidable as we represent a token in the standard corruptible model with both a token and an additional auxiliary string from the sender. We use $\mathcal{F}_{\mathsf{token}}^{\mathsf{corruptible,short}}$ when L_1 and L_2 are clear from the context.

In theory, we would like L_1 and L_2 be of constant size in security parameter. Though [NFR+17] suggests using logarithmic size in practice for better performance.

We define an implementation Token of $\mathcal{F}_{\mathsf{token}}^{\mathsf{corruptible}}$ in $\mathcal{F}_{\mathsf{token}}^{\mathsf{corruptible,short}}$-hybrid model as follows:

Theorem 8.1. *The protocol* Token *UC-realizes* $\mathcal{F}_{\mathsf{token}}^{\mathsf{corruptible}}$ *in* $\mathcal{F}_{\mathsf{token}}^{\mathsf{corruptible,short}}$*-hybrid model.*

Proof. (Sketch) The proof follows closely to the proof of Theorem 1 in [NFR+17] when the adversary Adv does not corrupt a token. We construct a simulator Sim as follows: Whenever Sim receives a create message from $\mathcal{F}_{\mathsf{token}}^{\mathsf{corruptible}}$, it passes the message to Adv. If Adv chooses to corrupt the token, Sim sends corrupt to $\mathcal{F}_{\mathsf{token}}^{\mathsf{corruptible}}$ and receives Π. It creates RAM from Π and generates $m = (\mathsf{mem}, \mathsf{header}, \tau_1)$ honestly as in Token and sends them to Adv. When Sim receives $\overline{T}_K^*$ and m^* from Adv, it parses $K^* = (K_e^*, K_{prf}^*)$ and $m^* = (\mathsf{mem}^*, \mathsf{header}^*, \tau_1^*)$. Sim then uses K_e^* to decrypt to get $\mathsf{RAM}^* = (\mathsf{cpustate}^*, \mathsf{mem}^*)$ representing a program Π^*. If Sim fails to get Π^* from this steps, it aborts. Otherwise, it sends Π^* to $\mathcal{F}_{\mathsf{token}}^{\mathsf{corruptible}}$. Sim runs Token honestly for this token, and sends execution message to $\mathcal{F}_{\mathsf{token}}^{\mathsf{corruptible}}$ whenever Adv complete the execution phase.

If Adv chooses not to corrupt the token created by an honest party, Sim passes notcorrupt to $\mathcal{F}_{\mathsf{token}}^{\mathsf{corruptible}}$, and creates a token for $\mathsf{RAM}_0 = (0^{|\mathsf{cpustate}|}, 0^{|\mathsf{mem}_0|})$ instead of the actual program RAM representing Π. Sim maintains a memory mem' to keep track of $\mathsf{mem} = \overline{\mathsf{mem}}_0 || inp || 0$.

Sim runs input and program authentication phases as described in Token, and also records the input authenticated. If Adv resets and sends different input, Sim aborts.

Whenever $\mathcal{F}_{\mathsf{token}}^{\mathsf{corruptible,short}}$ sends an encrypted state $\overline{\mathsf{state}}$ to Adv, Sim instead sends $\mathsf{Enc}_K(0^{|\mathsf{state}|})$.

To simulate ORAM algorithm, it uses the ORAM simulator and the PRF is replaced by a truly random function. Sim records all transcripts with Adv and sends the recorded response when Adv resets and sends the recorded messages. It aborts if it receives messages different from recorded ones.

Upon completion of the execution phase, Sim sends the execution message to $\mathcal{F}_{\mathsf{token}}^{\mathsf{corruptible}}$ with the recorded input. Sim sends the output it receives from $\mathcal{F}_{\mathsf{token}}^{\mathsf{corruptible}}$ to Adv.

We show the indistinguishability through a series of hybrids:

Hybrid H_0: This hybrid is the real world execution. Sim runs Token honestly in place of honest parties given their inputs using $\mathcal{F}_{\mathsf{token}}^{\mathsf{corruptible,short}}$ functionality.

Hybrid H_1: This hybrid is similar to Hybrid H_0 except that when Adv chooses to corrupt a token created by an honest party, Sim maintains a memory mem' to keep track of mem. Sim records all transcripts with Adv and sends the recorded response when Adv resets and sends the recorded messages. This hybrid is identical to H_1.

$\mathcal{F}_{\mathsf{token}}^{\mathsf{corruptible},\mathsf{short},L_1,L_2}$

Upon receiving $(\mathsf{create}, \mathsf{sid}, P_j, \Pi, m)$ from P_i with $i \neq j$:

1. If $|\Pi| > L_1$, ignore.
2. Otherwise, if there is no tuple of the form $(\mathsf{sid}, P_i, P_j, \star, \star)$, store $(\mathsf{sid}, P_i, P_j, \Pi, m, \mathsf{creating})$.
3. Send $(\mathsf{create}, \mathsf{sid}, P_i, P_j)$ to Adv.

Upon receiving $(\mathsf{corrupt}, \mathsf{sid}, P_i, P_j)$ from Adv:

1. Find the unique stored tuple $(\mathsf{sid}, P_j, P_i, \Pi, m, \mathsf{creating})$. If no such tuple exists, abort.
2. Send (Π, m) to Adv.

Upon receiving $(\mathsf{replace}, \mathsf{sid}, P_i, P_j, \Pi^*, m^*, \mathsf{state}_0)$ from Adv:

1. Find and remove the unique stored tuple $(\mathsf{sid}, P_j, P_i, \Pi, m, \mathsf{creating})$. If no such tuple exists, abort.
2. Store $(\mathsf{sid}, P_i, P_j, \Pi^*, \mathsf{state}_0)$, send $(\mathsf{done}, \mathsf{sid})$ to P_i and send $(\mathsf{create}, \mathsf{sid}, P_i, m^*)$ to P_j.

Upon receiving $(\mathsf{notcorrupt}, \mathsf{sid}, P_i, P_j)$ from Adv:

1. Find and remove the unique stored tuple $(\mathsf{sid}, P_j, P_i, \Pi, m, \mathsf{creating})$. If no such tuple exists, abort.
2. Store $(\mathsf{sid}, P_i, P_j, \Pi, \bot)$, send $(\mathsf{done}, \mathsf{sid})$ to P_i and send $(\mathsf{create}, \mathsf{sid}, P_i, m)$ to P_j.

Upon receiving $(\mathsf{execute}, \mathsf{sid}, P_i, inp)$ from P_j with $i \neq j$:

1. If $|inp| > L_2$, ignore.
2. Otherwise, find the unique stored tuple $(\mathsf{sid}, P_i, P_j, \Pi, \mathsf{state})$. If no such tuple exists, abort.
3. Run $\Pi(\mathsf{state}, inp)$ and let (state', out) be the output. If $\mathsf{state} \neq \bot$, set $\mathsf{state} = \mathsf{state}'$.
4. Send (sid, out) to P_j.

Upon receiving $(\mathsf{read}, \mathsf{sid}, P_i, P_j)$ from Adv:

1. Find the unique stored tuple $(\mathsf{sid}, P_j, P_i, P, \mathsf{state})$. If no such tuple exists, abort.
2. Send $(\mathsf{sid}, \mathsf{state})$ to Adv.

Figure 14: Token with Short Input Functionality $\mathcal{F}_{\mathsf{token}}^{\mathsf{corruptible},\mathsf{short},L_1,L_2}$

Token

To create a token running Π for P_j, P_i does the following:

1. Construct a RAM program $\mathsf{RAM} = (\mathsf{cpustate}, \mathsf{mem}_0)$ that runs Π.
2. Sample $K_e \leftarrow \mathsf{KeyGen}(1^\lambda)$ and $K_{prf} \leftarrow U_\ell$.
3. Let $\tau_1 = \mathsf{PRF}_{K_{prf}}(\mathsf{start}, \overline{\mathsf{mem}}_1)$ where $\overline{\mathsf{mem}}_k = \mathsf{Enc}_{K_e}(\mathsf{mem}_0[k]||\mathsf{id}||k)$, and $\overline{\mathsf{header}} = \mathsf{Enc}_{K_e}(\mathsf{cpustate}||\mathsf{id}||t)$.
4. Send $(\mathsf{create}, \mathsf{sid}, P_j, \overline{T}_K, (\overline{\mathsf{mem}}_0, \overline{\mathsf{header}}, \tau_1))$ to $\mathcal{F}_{\mathsf{token}}^{\mathsf{corruptible,short}}$ where $K = (K_e, K_{prf})$ and $\overline{\mathsf{mem}}_0 = \{\overline{\mathsf{mem}}_k\}_{k\in[t]}$.

To execute a token on input $inp = \{inp_k\}_{k\in[n]}$, P_j, given $(\overline{\mathsf{mem}}_0, \overline{\mathsf{header}}, \tau_1)$, does the following:

1. Authenticate the program feed and input feed as described in Section 7 to get $\tau_2, \ldots, \tau_{t+n}$ by sending $(\mathsf{execute}, \mathsf{sid}, P_i, -)$ to $\mathcal{F}_{\mathsf{token}}^{\mathsf{corruptible,short}}$ with appropriate command.
2. Execute as described in Section 7.

Figure 15: Token Implementation Token in $\mathcal{F}_{\mathsf{token}}^{\mathsf{corruptible,short}}$-hybrid model

Hybrid H_2: This hybrid is similar to Hybrid H_1 except that when Adv chooses to corrupt a token created by an honest party, Sim uses a truly random function instead of the PRF. This hybrid is computationally indistinguishable from H_1 by the property of the PRF.

Hybrid H_3: This hybrid is similar to Hybrid H_2 except that when Adv chooses to corrupt a token created by an honest party, Sim checks the authenticity of the program in $\overline{\mathsf{header}}$ and input in the execution phase directly from its recorded memory mem' instead of checking tags $\tau_1, \ldots, \tau_{t+n}$. When Adv initializes the program and the input, Sim records the pairing of tags and the program and the input. If Adv restarts the program and input feed, Sim compares with the recorded pairs and outputs the same tags for the same program and input. Since the tags are generated uniformly at random, the probability that poly-time Adv can find a different program or input with the same tag is negligible. This hybrid is indistinguishable from H_2 by the randomness of the truly random function.

Hybrid H_4: This hybrid is similar to Hybrid H_3 except that when Adv chooses to corrupt a token created by an honest party, *simu* compares state from Adv with the recorded state previously outputed on behalf of $\mathcal{F}_{\mathsf{token}}^{\mathsf{corruptible,short}}$ instead of using the authentication of Dec. This hybrid is computationally indistinguishable from H_3 by the authenticity of the encryption scheme.

Hybrid H_5: This hybrid is similar to Hybrid H_4 except that when Adv chooses to corrupt a token created by an honest party, Sim replaces all encryptions from $\mathcal{F}_{\mathsf{token}}^{\mathsf{corruptible,short}}$ to Adv with encryption

of zero strings, This hybrid is computationally indistinguishable from H_4 by the security of the encryption scheme.

Hybrid H_6: This hybrid is similar to Hybrid H_5 except that when Adv chooses to corrupt a token created by an honest party, Sim uses the ORAM simulator instead of the actual ORAM algorithm. This hybrid is statistically indistinguishable from H_6 by the property of ORAM simulator. This is the ideal world interaction between Sim and Adv.

□

Combining the above result with the result in Section 6 gives the following corollary.

Corollary 8.2. *Assuming OWFs, there exists a protocol that UC realizes $\mathcal{F}_{\mathsf{token}}^{\mathsf{abort}}$ functionality in $\mathcal{F}_{\mathsf{token}}^{\mathsf{corruptible},\mathsf{short}}$-hybrid model using n corruptible tokens with short inputs and small size against an adversary corrupting up to $n-1$ tokens.*

References

[Bea95] Donald Beaver. Precomputing oblivious transfer. *CRYPTO '95*, pages 97–109, 1995.

[Bea96] Donald Beaver. Correlated pseudorandomness and the complexity of private computations. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 479–488. ACM, 1996.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Foundations of Computer Science, 2001. Proceedings. 42nd IEEE Symposium on*, pages 136–145. IEEE, 2001.

[CDPW07] Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. Universally composable security with global setup. In *TCC '07*, pages 61–85. Springer, 2007.

[CF01] Ran Canetti and Marc Fischlin. Universally composable commitments. In *Advances in Cryptology?Crypto 2001*, pages 19–40. Springer, 2001.

[CGS08] Nishanth Chandran, Vipul Goyal, and Amit Sahai. New constructions for uc secure computation using tamper-proof hardware. In *Proceedings of the theory and applications of cryptographic techniques 27th annual international conference on Advances in cryptology*, pages 545–562. Springer-Verlag, 2008.

[CKL06] Ran Canetti, Eyal Kushilevitz, and Yehuda Lindell. On the limitations of universally composable two-party computation without set-up assumptions. *Journal of Cryptology*, 19(2):135–167, 2006.

[CKS+14] Seung Geol Choi, Jonathan Katz, Dominique Schröder, Arkady Yerukhimovich, and Hong-Sheng Zhou. (efficient) universally composable oblivious transfer using a minimal number of stateless tokens. In *Theory of Cryptography Conference*, pages 638–662. Springer, 2014.

[COPV13] Kai-Min Chung, Rafail Ostrovsky, Rafael Pass, and Ivan Visconti. Simultaneous resettability from one-way functions. In *Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on*, pages 60–69. IEEE, 2013.

[DFS16] Stefan Dziembowski, Sebastian Faust, and François-Xavier Standaert. Private circuits iii: Hardware trojan-resilience via testing amplification. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 142–153. ACM, 2016.

[DI05] Ivan Damgård and Yuval Ishai. Constant-round multiparty computation using a black-box pseudorandom generator. In *Annual International Cryptology Conference*, pages 378–394. Springer, 2005.

[DKMQN15] Nico Döttling, Daniel Kraschewski, Jörn Müller-Quade, and Tobias Nilges. General statistically secure computation with bounded-resettable hardware tokens. In *TCC (1)*, pages 319–344, 2015.

[DSMRV13] Dana Dachman-Soled, Tal Malkin, Mariana Raykova, and Muthuramakrishnan Venkitasubramaniam. Adaptive and concurrent secure computation from new adaptive, non-malleable commitments. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 316–336. Springer, 2013.

[FPS+11] Marc Fischlin, Benny Pinkas, Ahmad-Reza Sadeghi, Thomas Schneider, and Ivan Visconti. Secure set intersection with untrusted hardware tokens. In *Topics in Cryptology - CT-RSA 2011 - The Cryptographers' Track at the RSA Conference 2011, San Francisco, CA, USA, February 14-18, 2011. Proceedings*, volume 6558 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2011.

[GIS+10] Vipul Goyal, Yuval Ishai, Amit Sahai, Ramarathnam Venkatesan, and Akshay Wadia. Founding cryptography on tamper-proof hardware tokens. In *Proceedings of the 7th international conference on Theory of Cryptography*, pages 308–326. Springer-Verlag, 2010.

[GKOV12] Sanjam Garg, Abishek Kumarasubramanian, Rafail Ostrovsky, and Ivan Visconti. Impossibility results for static input secure computation. In *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, volume 7417 of *Lecture Notes in Computer Science*, pages 424–442. Springer, 2012.

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 218–229, 1987.

[GO07] Jens Groth and Rafail Ostrovsky. Cryptography in the multi-string model. In *Proceedings of the 27th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO'07, pages 323–341, Berlin, Heidelberg, 2007. Springer-Verlag.

[HPV16] Carmit Hazay, Antigoni Polychroniadou, and Muthuramakrishnan Venkitasubramaniam. Composable security in the tamper-proof hardware model under minimal complexity. In *TCC '16*, pages 367–399. Springer, 2016.

[IKM+13] Yuval Ishai, Eyal Kushilevitz, Sigurd Meldgaard, Claudio Orlandi, and Anat Paskin-Cherniavsky. On the power of correlated randomness in secure computation. In *Theory of Cryptography*, pages 600–620. Springer, 2013.

[IPS08] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer–efficiently. *Lecture Notes in Computer Science*, 5157:572–592, 2008.

[Kat07] Jonathan Katz. Universally composable multi-party computation using tamper-proof hardware. In *EUROCRYPT '07*, pages 115–128. Springer, 2007.

[Lin03] Yehuda Lindell. General composition and universal composability in secure multiparty computation. In *Foundations of Computer Science, 2003. Proceedings. 44th Annual IEEE Symposium on*, pages 394–403. IEEE, 2003.

[Lin10] Yehuda Lindell. Foundations of cryptography 89-856. 2010.

[LPV09] Huijia Lin, Rafael Pass, and Muthuramakrishnan Venkitasubramaniam. A unified framework for concurrent security: universal composability from stand-alone non-malleability. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 179–188. ACM, 2009.

[MY04] Philip MacKenzie and Ke Yang. On simulation-sound trapdoor commitments. In *EUROCRYPT '04*, pages 382–400. Springer, 2004.

[NFR+17] Kartik Nayak, Christopher W Fletcher, Ling Ren, Nishanth Chandran, Satya Lokam, Elaine Shi, and Vipul Goyal. Hop: Hardware makes obfuscation practical. In *NDSS '17*, 2017.