Efficient Resettably Secure Two-Party Computation

Tobias Nilges

Dept. of Computer Science, Aarhus University

Abstract

In 2000, Canetti, Goldreich, Goldwasser and Micali (STOC'00) proposed the notion of resettable zero-knowledge, which considers the scenario where a malicious verifier can reset the prover and force it to reuse its random tape. They provided a construction that resists such attacks, and in the following, the notion of resettability was considered in various other scenarios. Starting with resettably-sound zero-knowledge, over general resettable computation with one resettable party, to protocols where all parties are resettable.

Most of these results are only concerned with the feasibility of resettable computation, while efficiency is secondary. There is a considerable gap in the round- and communication-efficiency between actively secure protocols and resettably secure protocols. Following the work of Goyal and Sahai (EUROCRYPT'09), we study the round- and communication-efficiency of resettable two-party computation in the setting where one of the two parties is resettable, and close the gap between the two notions of security:

- We construct a fully simulatable resettable CRS in the plain model that directly yields constant-round resettable zero-knowledge and constant-round resettable two-party computation protocols in the plain model.
- We present a new resettability compiler that follows the approach of Ishai, Prabhakaran and Sahai (CRYPTO'08) and yields constant-rate resettable two-party computation.

1 Introduction

Resettability of cryptographic protocols was first investigated in the context of zero-knowledge protocols by Canetti et al. [CGGM00]. They observed that an adversarial verifier in a zero-knowledge proof can break the zero-knowledge property of the protocol if he is able to reset the prover. Reset attacks are therefore a stronger form of active attacks, where the adversary can force the honest party to reuse the same random tape for several protocol executions. This type of attack is not just of theoretical interest, but can occur in real scenarios where the prover program runs e.g. on a smartcard. In such a case, it is not difficult for the verifier to cut the power to the smartcard, and thereby reset it to its initial state.

Canetti et al. [CGGM00] showed that resettable zero-knowledge protocols exist, albeit with poorer efficiency than "standard" zero-knowledge protocols. Inspired by this, Barak et al. [BGGL01] considered the case of a resettable verifier in zero-knowledge protocols, i.e. the question whether soundness of a protocol holds if the verifier can be forced to reuse its randomness. In standard zero-knowledge protocols, such a resetting prover can basically employ the strategy of the simulator and create false proofs, thereby breaking the soundness. As it turns out, the case of a resettable verifier is "more challenging" in the sense that it requires an additional lever, namely non-black-box simulation, to prove the security. Non-black-box simulation means that the simulator gets the *code* of the verifier, instead of having only oracle access to it.

Finally, Deng et al. [DGS09] showed how to achieve simultaneously resettable zero-knowledge protocols, i.e. both the prover and verifier are resettable.

Goyal and Sahai [GS09] were the first to consider resettability not just for zero-knowledge protocols but for general two-party and multi-party computation protocols. They present a compiler that transforms any semi-honest MPC protocol into a resettable one. Their general approach is to use the passive-to-active transformation of Goldreich et al. [GMW87] and replace the zero-knowledge proofs with resettable and resettably-sound variants. Later, Goyal and Maji [GM11] even showed a SFE protocol where all parties are resettable, again using the GMW approach. However, their result does not provide general two-party computation, they can only realize all functions that do not behave like worst-case pseudoentropy generators. It was shown in [GKOV12] that general two-party computation is actually impossible, if all parties are resettable.

Recently, there have been several results in the context of resettably-sound zero-knowledge that reduced the computational assumptions [BP13, CPS13] and also the round-complexity of such protocols [COP$^+$14] to the same level as standard zero-knowledge proofs. For general resettable two-party computation, however, there is a large gap with respect to both the communication complexity and the round-complexity compared to non-resettable protocols. While constant-round [LP07] and constant-rate [IPS08, GIP15] two-party computation protocols are known, for the resettable setting the compiler of [GS09] requires $\mathcal{O}(\kappa)$ rounds of interaction, as does [GM11]. Regarding the round- and communication efficiency of resettable computation, not much is known and one might assume that a lower bound might be $\mathcal{O}(\log \kappa)$ rounds, which is the best known construction for resettable zero-knowledge (which is implied by resettable two-party computation). The only constant-round constructions for resettable zero-knowledge are in public key models such as the Bare Public Key model [CGGM00, DPV04, YZ07, DFG$^+$11], the Weak Public Key model [MR01] or the Upperboubded Public Key model [ZDLZ03]. So the obvious question is:

Is there an inherent gap between two-party computation and resettable two-party computation regarding the round-complexity and communication complexity?

Here by resettable two-party computation we mean protocols where only *one* party is resettable (i.e. we do not consider simultaneously resettable protocols). In the following we resolve this question by providing round-efficient and communication-efficient resettable two-party protocols, thereby essentially closing the gap.

1.1 Our Results

We observe that a lot of round-efficient resettably secure protocols already exist: there are non-interactive protocols for both zero-knowledge and two-party computation (we are referring to non-interactive secure computation (NISC) [IKO$^+$11]). However, all these protocols live in the CRS-hybrid model and thus do not yield improvements in the plain model. Our first contribution is a construction of a *resettable CRS* in the plain model that is fully simulatable (using non-black-box simulation techniques, which is inherent since resettable two-party computation implies resettably-sound zero-knowledge). This allows us to use the non-interactive protocols in the plain model, which implies several improvements over the current state of the art.

Constant-round resettable zero-knowledge argument of knowledge: We obtain the first constant-round resettable zero-knowledge argument of knowledge in the plain model. To the best of our knowledge, previously no constant-round construction has been achieved in the plain model, regardless of the simulation technique.

Constant-round resettable two-party computation: The resettable CRS in conjunction with NISC provides a simple and direct construction of a constant-round resettable two-party protocol.

While the above described resettable two-party computation protocol is round-efficient, it is based on garbled circuits and thus typically incurs a large communication overhead. Considering typical application scenarios of resettable computation, e.g. low bandwidth physical devices, this can be a severe drawback.

Therefore, as our second contribution, we present a protocol compiler inspired by the IPS compiler [IPS08] that combines two protocols with weak security guarantees in such a way that the resulting protocol is *resettably UC-secure*. There are other approaches that potentially offer communication-efficient and resettable 2PC, e.g. [GIP15], but we chose the IPS paradigm because of its versatility. Nevertheless, we believe that some of our techniques can be applied to other protocols in the literature and it might be interesting to compare the practical efficiency to our approach.

We will now briefly sketch the approach of [IPS08] for the two-party case. Take an honest majority protocol, e.g. [BOGW88, CCD88], where two clients provide the inputs for n servers and which tolerates up to t corrupted servers. Now, use a passively secure two-party protocol ([GMW87, Yao86]) to implement each of the servers. Of course, an adversary can now cheat in the passively secure protocol, and the servers' computation is not correct, i.e. the server is corrupted. Due to the honest majority setting, this is no problem as long as the adversary can only corrupt up to t of the servers. The key technical ingredient in the IPS compiler is a means to enforce this property, namely the setup of oblivious watchlists. A watchlist for a server is an encrypted channel over which each party has to send all its inputs into the respective server. The IPS compiler requires that before the emulation of the servers via the passively secure protocol, each party selects a subset of the servers to be on its watchlist via an oblivious transfer (OT). Thus, both parties get the complete view of some of the emulated servers, but each party does not know which servers the other party chose. The watchlists allow the parties to verify that the emulation of the servers was done correctly by the other party, and allows to aborts if an inconsistency is detected. With a correct choice of parameters, this approach guarantees that at most t servers are corrupted if no inconsistency is detected, and by the honest majority guarantee, this does not compromise the security of the compiled protocol.

In comparison to the IPS compiler, our compiler does not live in the OT-hybrid model, but requires a related primitive that we call deterministic randomized OT (dROT). Similar to the IPS compiler, the outer protocol has to be secure only in the honest majority setting, while the inner protocol must be secure in a slightly stronger model than the semi-honest setting. Using this compiler, we achieve several improvements over previous results.

Constant-rate resettable computation: We present the first resettably secure protocol with constant-rate communication (in the dROT-hybrid model). Our approach is the same as in [IPS08] with some minor modifications to the outer protocols.

Non-interactive UC-secure computation from tamper-proof hardware: We can plug an instantiation of our compiler directly into the construction of Döttling et al. [DMMQN13] to obtain non-interactive UC-secure computation of resettable functionalities.

Compared to the generality of the IPS compiler we have to make some drawbacks. Unconditional security with unbounded resets is not possible, because the resettable party only has a fixed amount of randomness. Thus, computational assumptions are necessary to create enough randomness to deal with an unbounded number of resets. Also, we require a slightly stronger security guarantee from the inner protocol.

Our protocols are the first protocols that achieve resettably secure computation without (extensively) relying on resettable and/or resettably-sound zero-knowledge proofs.

1.2 Our Techniques

Resettable CRS in the plain model. The protocol for the CRS is quite simple and follows the coin-toss protocol of Blum [Blu81]. The non-resettable party (called U, or user) commits to its input, and sends the commitment to the resettable party (called S or smartcard in the following). By deriving its own input via a PRF from the commitment, S ensures that resets will not influence the CRS. Instead of decommitting to its input, U sends its input and proves its knowledge of a decommitment (via a resettably-sound zero-knowledge proof).

This proof allows the simulator to change its input after receiving S's input. In order to simulate against U, we use a slightly modified variant of an extractable commitment and requires rewinding.
Resettable IPS compiler. At first sight it might seem unreasonable to assume that an OT-based compiler can be made resettable, because OT cannot be resettable in the first place. The crucial observation is that the IPS compiler only relies on random OT. For random OT, however, it is possible to derive a resettable variant.

We call this OT *deterministic randomized OT* (dROT), and the idea is that the receiver first sends a message which determines his choice bit. Based on this message, the sender derives his random inputs into the OT. This guarantees that for each input, the messages of the sender change, while the receiver still only learns one of the values. We present a simple adaptation of the OT protocol of Peikert et al. [PVW08] that satisfies this notion and is constructed in the CRS-hybrid model (thus the resettable CRS can be used to implement the protocol in the plain model).

We now show how we can realize a resettable version of the IPS compiler based on this primitive. In particular, we have to find a way to setup a watchlists infrastructure in a resettably secure way. First, assume that we simply use the dROT primitive to setup the watchlists deterministically. U inputs its choices into dROT, and S gets as output the keys for the watchlists. Then, S uses the random strings generated from the inputs in the watchlist selection to select its own watchlist, e.g. by applying a PRF to the values. Now the watchlist choices of S are unknown to U, but there is an attack that allows U to learn the choices: U simply corrupts a server and checks if S aborts. If yes, then this server was on the watchlist. Using this approach, U can learn all servers that S is watching, and cheat in the emulation of the other servers. The problem seems to be that the watchlists are selected before the computation takes place. Indeed, looking at other resettable protocols for e.g. zero-knowledge, it seems likely that the selection of the watchlist has to be *public*, otherwise a resetting attack might always leak the input in the OT. However, this stands in stark contrast to the approach of the IPS compiler.

We would thus like to fix the watchlists *after* the computation of the inner protocols, but this of course leaks too much information to U. But indeed, this approach is a step in the right direction. Instead of waiting until the end of the computation of the servers, we *split* the computation of the inner protocols in two phases: the computation phase, and the output phase. S and U compute the inner protocols in parallel, but only execute the computation phase. We require that the inner protocol remains private (even against active cheating of U) until U learns an the output. This holds for most passive secure protocols in the literature. Then, U commits to its views of the servers (similar to the MPC-in-the-head approach of Ishai et al. [IKOS07]) and sends the commitments to S. S deterministically derives a challenge from these commitments, and U has to unveil the challenged commitments. Now, S can verify that the computation of the inner protocol was performed correctly in much the same way that the non-resettable party U can. Only if this challenge is passed, the output is computed.

Of course, all the randomness used by the compiled protocol has to be derived from the inputs of U, and we therefore precompute the dROTs before the actual simulation of the outer protocol. Once these precomputed values are available, we start a derandomization of the dROTs where U uses his actual inputs in the computation of the inner protocols. At that point, all inputs of U are fixed, and S can derive the randomness for execution of both the inner and the outer protocol.

Due to the modularity of our constructions, the security analysis of the plain model protocols is very simple, compared to the complex analysis in [GS09] (due to concurrency issues). Along the way, we also give a precise simulation [MP06], which was left as an open problem by [GS09].

1.3 Structure of the Paper

After introducing the necessary tools and definitions in the preliminaries, we first define a resettable two-party UC-functionality in Section 3. After that we present the resettable CRS in Section 4. Then we describe our solutions for constant-round protocols in Section 5 and the resettable IPS compiler in Section 6

2 Preliminaries

Notation. We write $\mathbf{x}$ to denote a vector of elements $x_1, \ldots, x_n$. For convenience, we let $\mathbf{x} \oplus \mathbf{y}$ denote the component-wise XOR of $\mathbf{x}$ and $\mathbf{y}$ (for vectors of the same length). Typically, all messages that the adversary sends is marked with an asterisk, e.g. a^*. Messages that the simulator obtains or chooses are marked with a hat, e.g. $\hat{s}$. We use the standard cryptographic notions of negligible and overwhelming functions, and denote a probabilistic polynomial time algorithm by PPT.

2.1 Universal Composability Framework

The universal composability (UC) framework was introduced by Canetti [Can01]. Protocols proven secure in this framework have strong security guarantees with respect to their security when composed with other protocols in arbitrary environments.

UC-security of protocols is argued by comparing the real protocol with in ideal implementation of the functionality of the protocol. In particular, for every adversary in the real protocol there has to exist an ideal adversary that only interacts with the ideal functionality, such that any environment (coordinating the inputs of the parties in the real and ideal model) cannot distinguish the executions.

We also consider JUC [CR03], a variant of UC that considers protocols that have a joint state in form of a shared setup (e.g. a CRS or OTs). The JUC composition basically states that we can analyze a single instance of a multi-instance protocol and derive security even for arbitrary composition with other protocols.

2.2 Zero-Knowledge

We consider two notions of zero-knowledge argument systems. Let $\mathcal{R}_{\mathcal{L}}$ denote the witness relation of a language $\mathcal{L}$. Resettable zero-knowledge was first proposed by Canetti et al. [CGGM00] and cover the case of a resettable prover.

Definition 1. *A resettable zero-knowledge argument of knowledge system for a language $\mathcal{L} \in \mathcal{NP}$ consists of two PPT algorithms (P, V), where the prover P is resettable, and there exists two PPT algorithms Sim and Ext such that*

- *Completeness: For every $(x, w) \in \mathcal{R}_{\mathcal{L}}$,*

$$\Pr[\langle \mathsf{P}(w), \mathsf{V}\rangle(x) = 1] = 1$$

- *Soundness: For every $x \notin \mathcal{L}$ and every resettable PPT P^*,*

$$\Pr[\langle \mathsf{P}^*, \mathsf{V}\rangle(x) = 1] \leq \mathsf{negl}(()|x|)$$

- *Zero-Knowledge: For every $(x, w) \in \mathcal{R}_{\mathcal{L}}$ and every PPT V^*, the distributions $\mathsf{Real}^{=}_{\{}\rangle \mathsf{P}(w), \mathsf{V}^*\langle(x)\}$ and $\mathsf{Ideal}^{=}_{\{}\mathsf{Sim}(x, \mathsf{V}^*)\}$ are computationally indistinguishable.*

- *Proof of Knowledge: For every $x \in \mathcal{L}$ and every PPT algorithm P^*, there exists a negligible function ν such that*

$$\Pr[\mathsf{Ext}(x, \mathsf{P}^*) \in w_{\mathcal{L}}(x)] > \Pr[\langle \mathsf{P}^*, \mathsf{V}\rangle(x) = 1] - \nu$$

Resettably-sound zero-knowledge proofs are a variant of zero-knowledge systems, where the verifier can be reset.

Definition 2. *A resettably-sound zero-knowledge argument of knowledge system for a language $\mathcal{L} \in \mathcal{NP}$ consists of two PPT algorithms (P, V), where the verifier V is resettable, and there exists two PPT algorithms Sim and Ext such that*

- *Completeness: For every $(x, w) \in \mathcal{R}_{\mathcal{L}}$,*

$$\Pr[\langle \mathsf{P}(w), \mathsf{V}\rangle(x) = 1] = 1$$

- *Soundness: For every $x \notin \mathcal{L}$ and every malicious PPT P^*,*

$$\Pr[\langle \mathsf{P}^*, \mathsf{V}\rangle(x) = 1] \leq \mathsf{negl}(()|x|)$$

- *Zero-Knowledge: For every* $(x, w) \in \mathcal{R}_{\mathcal{L}}$ *and every resettable PPT* V^**, the distributions* $\mathsf{Real}_{\{}^{=}\rangle \mathsf{P}(w), \mathsf{V}^*\langle(x)\}$ *and* $\mathsf{Ideal}_{\{}^{=}\mathsf{Sim}(x, \mathsf{V}^*)\}$ *are computationally indistinguishable.*

- *Proof of Knowledge: For every* $x \in \mathcal{L}$ *and every PPT algorithm* P^**, there exists a negligible function* ν *such that*

$$\Pr[\mathsf{Ext}(x, \mathsf{P}^*) \in w_{\mathcal{L}}(x)] > \Pr[\langle \mathsf{P}^*, \mathsf{V}\rangle(x) = 1] - \nu$$

Recent constructions of resettably-sound zero-knowledge arguments based on one-way functions can be found in [BP13, CPS13].

2.3 Commitments Schemes

A commitment scheme consists of two phases: the commit phase and the unveil phase. In the commit phase, the sender commits to a value in such a way that the receiver cannot learn the value. This property is called *hiding*. At the same time, in the unveil phase, the sender must not be able to unveil a different value than the one he committed to, which is enforced by the *binding* property.

We will use a non-interactive commitment that is perfectly binding, which can be based on any one-way permutation, e.g. [Gol01].

2.3.1 Extractable Commitments

For our protocols, we need an extractable variant of such a commitment.

Definition 3. *A commitment scheme* $\mathsf{ExtCom} = (\mathsf{Commit}, \mathsf{Open})$ *is* extractable*, if there exists a PPT algorithm* Ext *that, given black-box access to a malicious* $\mathcal{A}_{\mathsf{S}}$*, outputs a pair* $(\hat{s}, \tau)$ *such that*

- *Simulation:* τ *is identically distributed to the view of* $\mathcal{A}_{\mathsf{S}}$ *at the end of interacting with an honest receiver* R *in the commit phase,*

- *Extraction: The probability that* τ *is accepting and* $\hat{s} = \perp$ *is negligible, and*

- *Binding: If* $\hat{s} \neq \perp$*, then it is infeasible to open* τ *to any value other than* $\hat{s}$*.*

Via the generic transformation shown in [PW09], any commitment can be made extractable. We will present a short sketch of the construction.

The sender draws κ random values r_i of size $|s|$ and commits to both r_i and $r_i \oplus s$ using any commitment scheme. These commitments (c_i^0, c_i^1) are sent to the receiver, who in turn draws a random challenge $e = (e_1, \ldots, e_\kappa)$ and sends it to the sender. The sender sends the decommitments for $c_i^{e_i}$ to the receiver, who checks if the decommitments are correct.

To unveil, the sender simply sends his input s and all decommitments to the receiver, who can check if all commitments are correct.

2.3.2 Extractable Commitments with Security Against a Resetting Sender

We will later additionally require that the extractable commitment remains binding even if the receiver is resettable. This can be achieved by a very simple enhancement of the above protocol: instead of sending a completely random challenge e, the receiver computes $e \leftarrow \mathsf{PRF}(\mathbf{c}^0, \mathbf{c}^1)$ (i.e. using all commitments from the sender) and sends this pseudorandom challenge to the sender. The security proof of the above protocol can be easily adapted to this modified protocol.

3 Resettably Secure Computation

Not all functionalities are meaningful in the setting of resettable computation. The ideal functionality $\mathcal{F}_{2PC}^{res}$ which is shown in Figure 1 allows—in principle—any function f, but functionalities like OT are learnable by repeated queries, and security against reset attacks does not prevent this. This problem was also addressed by Goyal and Sahai [GS09], since it is inherent to the setting. As they point out, most functionalities that provide a cryptographic output, e.g. when the resettable party holds as input a cryptographic key for a signature scheme, maintain their security also in the resettable setting.

In the following we only consider the two party case between a resettable party S, the smartcard, and a non-resettable party U, the user. The ideal functionality $\mathcal{F}_{2PC}^{res}$ takes the inputs from both parties and computes the output. An adversarial U is allowed to reset the computation, possibly forcing S to execute f several times with the same input.

Functionality $\mathcal{F}_{2PC}^{res}$

Implicitly parametrized by a function f.

Input phase:

1. Upon receiving a message $(\mathsf{sid}, \mathsf{ssid}, \mathtt{input}, \mathsf{P}, x_\mathsf{P})$ from party $\mathsf{P} \in \{\mathsf{S}, \mathsf{U}\}$, store $(\mathsf{sid}, \mathsf{ssid}, \mathsf{P}, x_\mathsf{P})$ and send $(\mathsf{sid}, \mathsf{ssid})$ to the adversary. Ignore any further messages $(\mathsf{sid}, \mathsf{ssid}, \mathtt{input}, \mathsf{P}, \hat{x}_\mathsf{P})$ from P once $(\mathsf{sid}, \mathsf{ssid}, \mathsf{P}, x_\mathsf{P})$ has been stored.

Output phase:

2. Once both inputs $(\mathsf{sid}, \mathsf{ssid}, \mathsf{S}, x_\mathsf{S})$ and $(\tilde{\mathsf{sid}}, \tilde{\mathsf{ssid}}, \mathsf{U}, x_\mathsf{U})$ with $\tilde{\mathsf{sid}} = \mathsf{sid}$ and $\tilde{\mathsf{ssid}} = \mathsf{ssid}$ are stored, compute $(y_\mathsf{S}, y_\mathsf{U}) = f(x_\mathsf{S}, x_\mathsf{U})$. Send $(\mathsf{sid}, \mathsf{ssid}, y)$ to the adversary.
3. Upon receiving a message $(\mathsf{sid}, \mathsf{ssid}, \mathtt{ack})$ from the adversary, send $(\mathsf{sid}, \mathsf{ssid}, y_\mathsf{S})$ to S and $(\mathsf{sid}, \mathsf{ssid}, y_\mathsf{U})$ to U.

Reset (malicious U only:)

4. Upon receiving a message $(\mathsf{sid}, \mathsf{ssid}, \mathtt{reset})$ from the adversary, delete the tuple $(\mathsf{sid}, \mathsf{ssid}, \mathsf{U}, x_\mathsf{U})$ corresponding to sid and ssid and return to the input phase.

Figure 1: Ideal functionality for resettable two-party computation.

In the next section we show a very simple and efficient way to realize constant-round resettable computation based on the resettable CRS from the previous section. Then we present our main result, a modification of the IPS compiler, which allows constant-rate resettable computation.

4 Fully Simulatable Resettable CRS in the Plain Model

We present a plain model realization of a resettable CRS, i.e. a CRS that can be reset by an adversarial user. The resettable CRS is straight-line simulatable against a corrupted smartcard, while it requires rewinding to simulate against a corrupted user. This actually implies that the CRS generation is concurrently secure, i.e. a malicious user that runs several copies of the protocol at the same time (in an arbitrary manner) will not break the security of the protocol.

Of course, achieving straight-line simulatability does not come for free; we require that the simulator gets the code of the corrupted S, i.e. we use a non-black-box simulation technique. In the bigger picture, this is not a problem: in order to achieve resettable two-party computation in the plain model, and thus in particular resettably-sound zero-knowledge (rsZK), non-black-box techniques are necessary [BGGL01]. Another advantage of this approach is that we require this possibly expensive step only once at the beginning of the protocols for the creation of a resettable CRS.

4.1 Security Notion

Let us first define a natural simulation-based definition for a resettable CRS in the two-party case. It is inspired by the simulation-based definition of resettable computation by Goyal and Sahai [GS09], but applied to a specific functionality. We distinguish between an ideal model and a real model, and show that any adversary in the real model can be emulated by a simulator in the ideal model. Our definition is basically a special case of the real/ideal paradigm for multiparty computation.

Ideal: In the ideal model we have a trusted party $\mathcal{F}_{\text{CRS}}^{\text{res}}$ (cf. Figure 2) that provides the CRS to both parties. The functionality initially samples a random string via a random oracle. As usual, parties can query the ideal functionality for the CRS, but an adversarial U is allowed to change the CRS via a reset.

We denote by $\mathsf{Ideal}_{\mathcal{A}}^{\mathcal{F}_{\text{CRS}}^{\text{res}}}$ the output of the honest parties and the output of $\mathcal{A}$ after interacting with $\mathcal{F}_{\text{CRS}}^{\text{res}}$.

$\mathcal{F}_{\text{CRS}}^{\text{res}}$

Let H be a random oracle that maps to strings of length ℓ. Set $\mathsf{CRS} = \mathsf{H}(1)$.

Query:

- Upon receiving a message $(\texttt{get}, \mathsf{P}_i, \mathsf{P}_j)$ from P_i, send $(\texttt{crs}, \mathsf{CRS})$ to P_i.

Reset (malicious U only):

- Upon receiving a message $(\texttt{reset}, j)$, set the common reference string to $\mathsf{CRS} = \mathsf{H}(j)$.

Figure 2: Ideal functionality for a resettable CRS.

Real: In the real model, the parties execute the protocol $\Pi_{\text{pCRS}}^{\text{res}}$. Honest parties will follow the protocol, while adversarial parties are allowed to deviate arbitrarily, in particular an adversarial U is allowed to reset S at any point, forcing a restart of the protocol. We denote by $\mathsf{Real}_{\mathcal{B}}^{\Pi_{\text{pCRS}}^{\text{res}}}$ the output of the honest parties and the complete view of the adversary $\mathcal{B}$ after the protocol execution.

Definition 4. *We say that $\Pi_{\text{pCRS}}^{\text{res}}$ is a secure implementation of $\mathcal{F}_{\text{CRS}}^{\text{res}}$ if for every PPT $\mathcal{A}$ corrupting either player in the real model there exists a PPT $\mathcal{S}$ in the ideal model such that*

$$\mathsf{Ideal}_{\mathcal{S}}^{\mathcal{F}_{\text{CRS}}^{\text{res}}} \approx \mathsf{Real}_{\mathcal{A}}^{\Pi_{\text{pCRS}}^{\text{res}}}$$

As noted in [GS09], this model translates to the *multiple incarnation non-interleaving* setting as defined in [CGGM00]. This means that the security of $\Pi^{\text{res}}_{\text{pCRS}}$ in the single instance scenario implies that the protocol remains secure even when U is allowed to interact with any number of incarnations *concurrently.*

4.2 A Fully Simulatable Resettable CRS

We now describe a simple protocol that securely implements $\mathcal{F}^{\text{res}}_{\text{CRS}}$ in the plain model according to Definition 4. The protocol structure follows a simple Blum coin-toss [Blu81], where the user commits to his random coins b and sends the commitment to S. In lack of a source of randomness, S then uses a pseudorandom function to derive its own coins a. Now U can open the commitment and send b to S. In order to make the protocol simulatable against S, U does not send the opening information, but instead uses a resettably-sound zero-knowledge argument of knowledge to prove the validity of the commitment. To simulate against U, we use an extractable commitment (the extraction requires rewinding), which allows the simulator to obtain the value b before sending a. Care has to be taken that the extractable commitment remains binding even if the sender can reset the receiver. The construction sketched in Section 2.3.2 satisfies this requirement.

Protocol $\Pi^{\text{res}}_{\text{pCRS}}$

Let ExtCom be an extractable commitment scheme and let (P, V) be a resettably-sound ZKAoK for the language $\mathcal{L} = \{(c_b, b) \mid \exists d_b \text{ s.t. } \mathsf{ExtCom.Open}(c_b, d_b, b) = 1\}$. Let ℓ be the desired length of CRS, and PRF_k be a pseudorandom function with key k that maps to strings of length ℓ.

1. U: Draw a random $b \in \{0,1\}^\ell$ and compute $(c_b, d_b) \leftarrow \mathsf{ExtCom_r.Commit}(b)$ and send c_b to the S.
2. S: Compute $a \leftarrow \mathsf{PRF}_\mathsf{k}(c_b)$ and send a to U.
3. U: Send b to S. Start the prover P with witness d_b and input (c_b, b). Forward messages between the S and P.
4. S: Start the verifier V with input (c_b, b) and forward the messages between U and V. If V accepts, set $\mathsf{CRS} = a \oplus b$ and output CRS.
5. U: Compute $\mathsf{CRS} = a \oplus b$ and output CRS.

Figure 3: Protocol $\Pi^{\text{res}}_{\text{pCRS}}$ that realizes $\hat{\mathcal{F}}^{\text{res}}_{\text{CRS}}$ in the plain model.

Theorem 1. *The protocol $\Pi^{\text{res}}_{\text{pCRS}}$ securely implements $\mathcal{F}^{\text{res}}_{\text{CRS}}$ with computational security.*

Proof. **Corrupted S.** The simulation strategy of $\mathcal{S}_\mathsf{S}$ is straightforward (cf. Figure 4). The simulator first commits to a random value $\hat{b}$, and then learns the value a^* from $\mathcal{A}_\mathsf{S}$. Instead of proving the correctness of the commitment $c_{\hat{b}}$, he first queries the ideal functionality $\mathcal{F}_\mathsf{CRS}$ for CRS and then sets $b = \mathsf{CRS} \oplus a$. Since the simulator gets access to the code of $\mathcal{A}_\mathsf{S}$, he can use the non-black-box simulator of the rsZKAoK to provide a false proof, i.e. he proves that the commitment contained b instead of $\hat{b}$.

We sketch the indistinguishability of the real protocol and the simulated protocol in a series of hybrid experiments.

Experiment 0: This is the real protocol.

Simulator $\mathcal{S}_{\mathsf{S}}$

1. Sample a random $\hat{b} \in \{0,1\}^{\ell}$ and compute $(c_{\hat{b}}, d_{\hat{b}}) \leftarrow \mathsf{ExtCom.Commit}(\hat{b})$. Send $c_{\hat{b}}$ to $\mathcal{A}_{\mathsf{S}}$.
2. Upon receiving a message a^*, query $\mathcal{F}_{\mathrm{CRS}}^{\mathrm{res}}$ to obtain CRS. Compute $b = \mathsf{CRS} \oplus a^*$ and send it to $\mathcal{A}_{\mathsf{S}}$. Construct a malicious verifier V^* from $\mathcal{A}_{\mathsf{S}}$ that basically simulates the zero-knowledge step of $\mathcal{A}_{\mathsf{S}}$. Start the non-black-box simulator Sim on V^* with input $(c_{\hat{b}}, b)$. Use the output of Sim to continue the simulation of $\mathcal{A}_{\mathsf{S}}$ and let $\hat{\mathsf{CRS}}$ be the output. Abort if $\hat{\mathsf{CRS}} \neq \mathsf{CRS}$.

Figure 4: Straight-line simulator against a corrupted S.

Experiment 1: Identical to Experiment 0, except that instead of starting the prover P with input (c_b, b) and witness d_b, $\mathcal{S}_1$ starts the non-black-box simulator Sim with input (V^*, c_b, b) as described in Figure 4.

Experiment 2: Identical to Experiment 1, except that $\mathcal{S}_2$ replaces b by $\hat{b} = \mathsf{CRS} \oplus a$. This corresponds to the simulation of $\mathcal{S}_{\mathsf{S}}$.

Computational indistinguishability of Experiments 0 and 1 follows directly from the computational zero-knowledge property of (P, V), while Experiment 1 and Experiment 2 are computationally indistinguishable due to the computational hiding property of ExtCom.

Corrupted U. Against a corrupted $\mathcal{A}_{\mathsf{U}}$, the simulator $\mathcal{S}_{\mathsf{U}}$ uses the black-box extractor of the extractable commitment scheme to learn $\mathcal{A}_{\mathsf{U}}$'s input b. It can then replace the value a such that the resulting CRS fits the CRS from $\mathcal{F}_{\mathrm{CRS}}^{\mathrm{res}}$. A formal description of $\mathcal{S}_{\mathsf{U}}$ is shown in Figure 5.

Simulator $\mathcal{S}_{\mathsf{U}}$

Setup a counter $j = 1$.

1. Upon receiving a message c_b^*, check if a tuple $(c_b^{*\prime}, \hat{b}', \hat{a}', j')$ with $c_b^{*\prime} = c_b^*$ and has been stored.
   - If yes, set $\hat{a} = \hat{a}'$ and send $(\mathtt{reset}, j')$ to $\mathcal{F}_{\mathrm{CRS}}^{\mathrm{res}}$.
   - If not, set $j = j + 1$, send $(\mathtt{reset}, j)$ and $(\mathtt{get})$ to $\mathcal{F}_{\mathrm{CRS}}^{\mathrm{res}}$ to learn CRS_j. Start the extractor Ext with input $(c_b^*, \mathcal{A}_{\mathsf{U}})$ to obtain $\hat{b}$ and set $\hat{b} = 0^{\ell}$ if Ext outputs $\perp$. Set $\hat{a} = \mathsf{CRS}_j \oplus \hat{b}$.

   Send $\hat{a}$ to $\mathcal{A}_{\mathsf{U}}$.
2. Upon receiving a value b^*, start the verifier V with input (c_b^*, b^*) and abort if V aborts. Otherwise, check if $\hat{b} \neq b^*$ and abort if that is the case. Store the tuple $(c_b^*, \hat{b}, \hat{a}, j)$.

Figure 5: Simulator against a corrupted U.

The following hybrid experiments show the indistinguishability of the simulated and the real protocol.

Experiment 0: This is the real protocol.

Experiment 1: Identical to Experiment 0, except that $\mathcal{S}_1$ uses the extractor Ext of ExtCom on input c_b to learn a value $\hat{b}$ and aborts if V accepts but $\hat{b} \neq b^*$.

Experiment 2: Identical to Experiment 1, except that instead of setting $a = \mathsf{PRF}_{\mathsf{k}}(c_b)$, $\mathcal{S}_2$ computes a as follows. If a tuple $(c_b^{*\prime}, \hat{b}', \hat{a}', j')$ with $c_b^{*\prime} = c_b$ has been stored, set $a = \hat{a}'$. Otherwise draw a uniformly random CRS and set $a = \mathsf{CRS} \oplus \hat{b}$. This corresponds to the simulation of $\mathcal{S}_{\mathsf{U}}$.

The computational indistinguishability of Experiments 1 and 2 follows directly from the fact that (a) CRS is uniformly random, i.e. $\hat{a}$ is uniformly distributed, and (b) the output of $\mathsf{PRF_k}$ is pseudorandom. A distinguisher that distinguishes between both experiments can be used to break the security of the PRF.

It remains to show that Experiment 0 and Experiment 1 are indistinguishable. This hold due to the extractability of ExtCom and the soundness of (P, V). The only way to distinguish both experiments is to provoke an abort of $\mathcal{S}_1$ for $\hat{b} \neq b^*$, otherwise both experiments are identically distributed. Thus $\mathcal{A}_{\mathsf{U}}$ must convince the sender S of a different b^* than the one extracted by $\mathcal{S}_1$.

Assume for the sake of contradiction that $\mathcal{A}_{\mathsf{U}}$ succeeds with advantage ϵ. We will construct a malicious prover P^* from $\mathcal{A}_{\mathsf{U}}$ that breaks the soundness of (P, V) with non-negligible probability. Let $m = \mathsf{poly}(\kappa)$ be the number of times that $\mathcal{A}_{\mathsf{U}}$ invokes S.

P^* chooses $i \in [m]$ uniformly at random and then simulates the interaction of $\mathcal{A}_{\mathsf{U}}$ and $\mathcal{S}_1$ until $\mathcal{A}_{\mathsf{U}}$ makes the i-th call. P^* now announces the statement (c_b^*, b^*) to the verifier $\hat{\mathsf{V}}$ he interacts with and redirects the prover messages that $\mathcal{A}_{\mathsf{U}}$ sends to $\mathcal{S}_1$ to $\hat{\mathsf{V}}$ instead. P^* terminates after the proof phase.

Since the commitment scheme ExtCom is statistically binding, there exists with overwhelming probability only one value $\hat{b}$ that can be hidden inside the commitment c_b^*, and the extractor of ExtCom will extract this value with probability $1 - \nu$, where ν is a negligible function. Thus, one of the two statements that $\mathcal{A}_{\mathsf{U}}$ proves must be wrong.

We claim that P^* succeeds in convincing $\hat{\mathsf{V}}$ of a false statement with probability at least $\frac{\epsilon}{m} - \nu$. The probability that such a false statement occurs in Experiment 1 is at least ϵ, otherwise the simulation would be identical. Since at most m statements are proven, P^*'s probability of hitting the false statement via i is $\frac{1}{m}$. Combined, P^* proves a false statement with probability at least $\frac{\epsilon}{m} - \nu$, which is non-negligible. This contradicts the soundness of the argument system (P, V). □

Round complexity of $\Pi_{\text{pCRS}}^{\text{res}}$. Using the protocol of [COP+14], the resettably-sound ZK argument of knowledge requires 4 rounds, while the extractable commitment requires 3 rounds [PW09]. Combined, this yields 9 rounds for the complete protocol. However, a more careful analysis shows that we can actually collapse several rounds:

1. By deriving a directly with the challenge for the extractable commitment scheme using the PRF, U can send b together with the responses to the challenge, saving two rounds;

2. The 4-round WiAoK used in the resettably-sound zero-knowledge system of [COP+14] is generic, i.e. we can replace it by an input-delayed variant [LS91] where the statement does not have to be fixed at the start of the protocol (at the cost of requiring OWP). We can thus additionally send the first message of the zero-knowledge protocol with a and the challenge for the extractable commitment.

Considering all these optimizations, the resulting protocol has only 5 rounds.

5 Constant-Round Resettable Functionalities

Combining the resettable CRS from the previous section with results from the literature, we directly obtain constant-round protocols for several functionalities. The approach for all results presented in this section is the same: establish the resettable CRS and start a non-interactive protocol in the CRS-hybrid model that realizes the desired functionality. Then apply the standard technique proposed by Canetti et al. [CGGM00] to make the complete protocol resettable, i.e. derive the randomness for the non-interactive protocol from the transcript of the CRS generation via a PRF.

5.1 Constant-Round Resettable Zero-Knowledge Argument of Knowledge

As a first result, we obtain a constant-round resettable zero-knowledge argument of knowledge (cf. Definition 1) by combining the resettable CRS with the non-interactive zero-knowledge proof of knowledge by Rackoff and Simon [RS92]. Please note that the *proof* of knowledge property of [RS92] is reduced to an *argument* of knowledge, since $\Pi^{\mathrm{res}}_{\mathrm{pCRS}}$ provides only computational security and an unbounded prover could therefore manipulate the CRS.

Corollary 5 (of Theorem 1 and [RS92])**.** *There exists a 6-round resettable zero-knowledge argument of knowledge.*

Since the resettable CRS relies on non-black-box simulation, so does the resulting argument system. The impossibility result of Barak et al. [BLV03] for public coin resettable zero-knowledge proofs (even from non-black-box simulation) does not apply, because the resulting protocol is neither public coin nor a proof system. As far as we are aware, this is the first construction of a constant-round resettable zero-knowledge argument of knowledge.

5.2 Constant-Round Resettable Computation

While there exists no real non-interactive (i.e. 1-round) two-party computation protocol, recently the notion of non-interactive secure computation (NISC) [IKO$^+$11] has been proposed. Overly simplified, the idea of NISC can be summarized as follows: there exist 2-round OT protocols (e.g. [PVW08], cf. Section 6.1.1) which can be combined with garbled circuits to yield a 2PC protocol with minimal interaction, namely two messages. The first message of the OT is sent by the receiver, fixing his input (possibly before a circuit has been chosen). Then, the sender constructs a garbled circuit and replies to the first message with "OT-encryptions" of the labels, and the garbled circuit. Thus, the receiver can obtain a valid set of labels, evaluate the circuit and learn the result. Note that the sender of the circuit does not obtain a result.

Corollary 6 (of Theorem 1 and [IKO$^+$11])**.** *There exists a 7-round resettably secure two-party computation protocol.*

Again we can simply combine the resettable CRS with the NISC protocol, while the randomness for the NISC protocol is derived from the transcript via a PRF. If it is necessary that both parties obtain an output, they can simply execute the NISC protocol in both directions, at the cost of one additional round. While the result of Ishai et al. [IKO$^+$11] makes only a black box use of the underlying primitive, the resettable CRS does not. Thus our plain model protocols (including the result in Section 6.4) lose this desirable property in theory, but we want to highlight that the main computation of the protocol still retains its efficiency.

6 Constant-Rate Resettable Computation

While the previously described protocols are very efficient with respect to the number of rounds, the garbled circuit technique makes the protocols inherently inefficient with respect to the communication complexity. It seems unlikely that this can be overcome without additional interaction. Additionally, the garbled circuit approach of the NISC makes the evaluation of arithmetic circuits rather inefficient, so in the following we describe a resettably secure variant of the IPS compiler [IPS08] that solves these problems.

Let us give a very brief introduction to the general structure of our compiler. It will be presented in the $\hat{\mathcal{F}}_{\text{dROT}}$-hybrid model, as defined in the next subsection. The compiler itself consists of an outer protocol Π in the client-server model, which implements the desired functionality, and an inner protocol ρ, which is used to emulate the servers in Π. In the following we describe these components and our requirements before presenting the actual compiler.

6.1 Tools

Our construction requires several tools. First and foremost, we need a special type of random OT, where the sender in essence deterministically derives its inputs from an encoding of the choice bits. We call this functionality $\mathcal{F}_{\text{dROT}}$ for *deterministic randomized OT*. We first define the ideal functionality, and then show how this functionality can be realized (in the $\mathcal{F}_{\text{CRS}}$-hybrid model). Indeed, similar to [PVW08], we will actually implement the multi-session version $\hat{\mathcal{F}}_{\text{dROT}}$ of this functionality. Additionally, we need a resettably UC-secure commitment. While there possibly are resettable constructions that can be realized in the $\mathcal{F}_{\text{CRS}}$-hybrid model, we focus on a construction that is directly built from $\hat{\mathcal{F}}_{\text{dROT}}$.

6.1.1 Deterministic Randomized OT

Figure 6 shows the functionality $\hat{\mathcal{F}}_{\text{dROT}}$, which is a special case of a deterministic randomized OT protocol in the multi-session setting. The receiver sends his choice bit to the functionality. The functionality derives two random values from the choice bit and the currently set tag and sends them to the OT sender. If the sender is malicious, he can overwrite these values. The tag is included to later allow different encodings of the choice bit in the protocol. Thus, an adversarial U is allowed to change tag arbitrarily. Eventually, the receiver obtains the random value corresponding to his choice bit.

In the following we give a brief sketch on how to realize $\hat{\mathcal{F}}_{\text{dROT}}$. We first show that a simple adaptation of the OT protocol of Peikert et al. [PVW08] realizes $\hat{\mathcal{F}}_{\text{dROT}}$ in the CRS-hybrid model. Without going too much into detail, we give a short overview of the protocol. The main idea is to use a so-called dual-mode cryptosystem that allows to encrypt messages under a public key, which is derived from a CRS. The encrypter selects a branch, i.e. one bit, and if the public key was created for the same branch, decryption is possible. All messages encrypted in the other branch are statistically hidden.

In the protocol, the receiver starts by computing a key pair for his choice bit c. He sends the corresponding public key to the sender, who derives his inputs r_0, r_1 from this public key based on a pseudorandom function. He then uses the public key and encrypts both values under a different branch, which allows the receiver to decrypt only one value correctly. A formal description is shown in Figure 7

Functionality $\hat{\mathcal{F}}_{\text{dROT}}$

Let $\mathsf{H} : \{0,1\}^* \to \{0,1\}^{2\kappa}$ be a random oracle and tag a random value of size $O(\kappa)$.

Choice phase:

1. Upon receiving a message $(\mathsf{sid}, \mathsf{ssid}, \mathtt{receiver}, c)$ from R, compute $(r_0, r_1) \leftarrow \mathsf{H}(\mathsf{tag}, c)$ and store the tuple $(\mathsf{sid}, \mathsf{ssid}, (r_0, r_1), \mathsf{tag}, c)$. Send $(\mathsf{sid}, \mathsf{ssid})$ to the adversary.
2. Upon receiving a message $(\mathsf{sid}, \mathsf{ssid}, \mathsf{ack})$ from the adversary, send $(\mathsf{sid}, \mathsf{ssid}, (r_0, r_1))$ to S and $(\mathsf{sid}, \mathsf{ssid}, r_c)$ to R.

Input overwrite (malicious S only):

3. Upon receiving a message $(\tilde{\mathsf{sid}}, \tilde{\mathsf{ssid}}, \mathtt{overwrite}, (\tilde{r_0}, \tilde{r_1}))$ from the adversary with $\tilde{r_0}, \tilde{r_1} \in \{0,1\}^{\kappa}$, check if there exists a tuple $(\mathsf{sid}, \mathsf{ssid}, (r_0, r_1), c)$ with $\tilde{\mathsf{sid}} = \mathsf{sid}$ and $\tilde{\mathsf{ssid}} = \mathsf{ssid}$, which has not been sent to the sender. If it exists, set $r_0 = \tilde{r_0}$ and $r_1 = \tilde{r_1}$.

Tag overwrite (malicious R only):

4. Upon receiving a message $(\tilde{\mathsf{sid}}, \tilde{\mathsf{ssid}}, \mathtt{tag}, \tilde{\mathsf{tag}})$ from the adversary, check if there exists a tuple $(\mathsf{sid}, \mathsf{ssid}, (r_0, r_1), \mathsf{tag}, c)$ with $\tilde{\mathsf{sid}} = \mathsf{sid}$ and $\tilde{\mathsf{ssid}} = \mathsf{ssid}$, which has not been sent to the sender. If it exists, compute $(r_0', r_1') \leftarrow \mathsf{H}(\tilde{\mathsf{tag}}, c)$ and replace the corresponding tuple.

Figure 6: Ideal functionality for deterministic randomized OT.

Protocol $\hat{\Pi}_{\text{dROT}}$

Both parties have access to $\mathcal{F}_{\text{CRS}}$. Let DM be a dual-mode cryptosystem according to [PVW08] and PRF be a pseudorandom function that outputs strings of length 2κ.

1. R: Upon receiving input $(\mathsf{sid}, \mathsf{ssid}, \mathtt{receiver}, c)$, compute $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{CRS}, c)$ and send pk to S.
2. S: Compute $(r_0, r_1) \leftarrow \mathsf{PRF}(\mathsf{pk})$ and set $y_b \leftarrow \mathsf{Enc}(\mathsf{pk}, b, r_b)$ for each $b \in \{0,1\}$. Send (y_0, y_1) to R and output (r_0, r_1).
3. R: Compute $\mathsf{Dec}(\mathsf{sk}, y_c)$ and output r_c.

Figure 7: Protocol $\hat{\Pi}_{\text{dROT}}$ that UC-realizes $\hat{\mathcal{F}}_{\text{dROT}}$ in the $\mathcal{F}_{\text{CRS}}$-hybrid model.

Lemma 7. *Protocol $\hat{\Pi}_{\text{dROT}}$ realizes $\hat{\mathcal{F}}_{\text{dROT}}$ in the $\mathcal{F}_{\text{CRS}}$-hybrid model, given that DM is a dual-mode cryptosystem and PRF is a pseudorandom function.*

Sketch. The simulation works basically identical to [PVW08], with a few minor changes. We refer the interested reader to [PVW08] for a detailed proof and only sketch the simulator.

Corrupted R. The simulator selects a messy CRS and learns a trapdoor. He uses this trapdoor to find the messy key, which in turn specifies the adversary's choice bit c^* (the choice bit is the complement of the index of the messy key). The simulator inputs $(\mathsf{sid}, \mathsf{ssid}, \mathtt{receiver}, c^*)$ into $\mathcal{F}_{\text{dROT}}$ and obtains r_{c^*}. He then just computes $y_{c^*} \leftarrow \mathsf{Enc}(\mathsf{pk}^*, c^*, r_{c^*})$ and $y_{1-c^*} \leftarrow \mathsf{Enc}(\mathsf{pk}^*, 1-c^*, 0)$ and sends them to the receiver. Due to the computational indistinguishability of the outputs of PRF and H, the ideal and the real execution are indistinguishable by $\mathcal{Z}$.

Corrupted S. The simulator selects a CRS in decryption mode and learns a trapdoor which allows to decrypt both values y_0^*, y_1^* that the adversary sent. He obtains the values r_0^* and r_1^* and waits for the $(\mathsf{sid}, \mathsf{ssid})$-message from $\hat{\mathcal{F}}_{\mathrm{dROT}}$. When he receives this message, he sends $(\mathsf{sid}, \mathsf{ssid}, \mathtt{overwrite}, (r_0^*, r_1^*))$ to $\mathcal{F}_{\mathrm{dROT}}$ and then sends $(\mathsf{sid}, \mathsf{ssid}, \mathsf{ack})$. Indistinguishability follows from the computational indistinguishability of a messy and a decryption CRS. □

This construction can be used to create a k-out-of-n OT using standard techniques. First, a 1-out-of-n OT can be constructed by e.g. replacing the normal 1-out-of-2 OT in the protocol of Naor and Pinkas [NP05] with the above construction. Then, use e.g. the approach described in [IPS08] to obtain a k-out-of-n OT from 1-out-of-n OT.

6.1.2 UC-secure commitments from $\mathcal{F}_{\mathrm{dROT}}$

There are several UC-secure commitments that are proven secure in the $\mathcal{F}_{\mathrm{OT}}$-hybrid model. Our OT variant $\mathcal{F}_{\mathrm{dROT}}$ however is a bit more restrictive, in the sense that it only allows random inputs from the sender, and no real derandomization (cf. Section 6.4). Additionally, since we require a commitment that is secure even if the receiver can be reset, we cannot simply use an arbitrary UC-secure commitment.

However, looking at protocols from the literature, we observe that e.g. the recent commitment protocol of Cascudo et al. [CDD$^+$15] can be easily modified to be used with $\mathcal{F}_{\mathrm{dROT}}$, since the protocol actually relies on random OT (and a PRF due to efficiency reasons, otherwise it is unconditionally secure). Additionally, the commitment in [CDD$^+$15] has a non-interactive commit and unveil phase, i.e. the sender just sends one message in both cases. As described above, this provides security even against a resetting receiver. The setup phase of their protocol simply consists of evaluations of an OT, which we replace with $\mathcal{F}_{\mathrm{dROT}}$.

Let us briefly describe their protocol. During the setup phase, the receiver essentially learns keys for a watchlist via the OTs. In order to commit, the sender encodes his message via an error correcting code and sends the codeword masked by the keys to the receiver. Once he wants to unveil, he sends the complete codeword, and the receiver can check that the codeword is consistent with the values he received during the commit phase. The main idea behind the construction is that the receiver does not learn enough about the codeword to decode it, thus the commitment is hiding. On the other hand, if the sender wants to decommit to a different value, he must change a lot of positions in the codeword, and he does not know which positions the receiver knows. This ensures the binding property.

6.2 The Outer Protocol Π

We now start with the description of the major building blocks of the compiler. The outer protocol Π is a protocol between $n+2$ parties , where n parties are the servers and the two clients provide the input to the servers. The servers have no inputs and produce no outputs (except to the clients). Let us briefly define a notion to "count" the rounds in the outer protocol. Basically, we only count the client-server communication and ignore any communication between the servers.

Definition 8. *We say that Π is a m-round* protocol if the clients sequentially provide m inputs for the servers. Server-to-server communication can be arbitrary.*

Many honest majority protocols realizing an SFE can be described as a 1-round* protocol: the clients provide inputs to the servers, who (interactively) compute the function and then only send the output to the clients. Of course, any reactive functionality requires more than 1 round.

Functionality: The protocol Π realizes a possibly randomized functionality $\mathcal{F}$ among the two clients.

Security: We assume UC-security against static corruption of a client and adaptive corruption of up to t servers, where $t \in \Omega(n)$. We assume active corruption of the servers and statistical or computational security.

Protocol Structure: Our protocol structure essentially follows the structure in [IPS08], except that we *do not allow* client-to-client messages. In most cases, outer protocols do not need client-to-client communication, so this restriction is not too severe. The protocol proceeds in rounds.

- Each client sends messages to the servers. Both clients and servers update their internal state according to these messages.
- Each party reads the messages received in this round and adds them to its state. Honest parties do not erase or manipulate the stored messages.

Each server S^j in Π maintains a state Σ^j. This state can be expressed as the combination of all messages μ_i^j, which represent the messages that server j received from client i (the local state), and an internal state σ^j. The function of the server π takes as input the current state and the incoming messages $\mathbf{w}^j$ from the clients and computes the updated state $\sigma^{j\prime}$ including all messages $\mathbf{m}^j$ that are output to the clients, i.e. $(\sigma^{j\prime}, \mathbf{m}^j) \leftarrow \pi(\sigma^j, \mu^j, \mathbf{w}^j)$. The local states are updated by $\mu_i^{j\prime} = \mu_i^j \circ (\mathbf{m}_i^j, \mathbf{w}_i^j)$.

6.3 The Inner Functionality $\mathcal{G}$ and the Inner Protocol ρ

We have the clients S and U play the role of the two clients in Π, but they will additionally implement the servers S^j of Π via a secure two-party computation. Compared to the approach in [IPS08], we have to be more careful both with regard to the security requirements of the inner protocol and the way the servers are implemented.

Functionality: The clients S and U implement the servers S^j of Π via a secure two-party computation protocol ρ^j. The server functionality that the clients will implement is denoted $\mathcal{G}$.

Security: In comparison to the IPS compiler, it is not sufficient in our case to rely on an adaptively semi-honest secure inner protocol ρ because of the following technical problem. In the IPS compiler, the simulator can always read all watchlists, and aborts if more than $\frac{1}{3}$ of the servers are corrupted (an honest party would also immediately observe this and abort, with overwhelming probability). However, in our case, the check of the correct execution is done later in the protocol, so in principle an adversary could corrupt all servers at one point in the protocol, and our simulator would have to corrupt these servers in the outer protocol to start a simulation of them. But this obviously does not work with the security requirement of the outer protocol, where only $\frac{1}{3}$ of the servers must be corrupted, i.e. by doing so the adversary would be able to distinguish the simulation from a real protocol run.

Instead, we require that the inner protocol remains private even against an active adversary (in the actively secure OT hybrid model), up to the point when he learns the result of the computation. We call this notion *semi-active security.* While this notion seems to be folklore, e.g. a similar requirement was previously used in [DGN10] to achieve covert security from passive security, to the best of our knowledge the adversary model was never formally defined.

The ideal functionality $\mathcal{F}_{\text{2PC}}^{\text{sa}}$ (cf. Figure 8) is based on the standard notion of passively secure two-party computation. However, we allow an adversary to select between passive corruption and a restricted form of active corruption. In the passive case, he will obtain both the input of the corrupted party and the output of the computation, which is the standard case for passive corruptions. If he chooses to actively corrupt, he will additionally obtain the input of the honest party, *but only after* the honest party allowed the delivery of the final result. This is modeled via the `deliver` message in $\mathcal{F}_{\text{2PC}}^{\text{sa}}$. Please note that this notion is implied by active security.

Functionality $\mathcal{F}_{\text{2PC}}^{\text{sa}}$

Implicitly parametrized by a function f.

Input phase:

1. Upon receiving a message $(\mathsf{sid}, \texttt{input}, \mathsf{P}, x_{\mathsf{P}})$ from party $\mathsf{P} \in \{\mathsf{P}_1, \mathsf{P}_2\}$, store $(\mathsf{sid}, \mathsf{P}, x_{\mathsf{P}})$ and send (sid) to the adversary. Ignore any further messages $(\mathsf{sid}, \texttt{input}, \mathsf{P}, \hat{x}_{\mathsf{P}})$ from P once $(\mathsf{sid}, \mathsf{P}, x_{\mathsf{P}})$ has been stored.

Output phase:

2. Once all inputs $(\mathsf{sid}, \mathsf{P}_i, x_{\mathsf{P}_i})$ are stored, compute $y = f(x_{\mathsf{P}_1}, x_{\mathsf{P}_2})$. Send $(\mathsf{sid}, \texttt{result})$ to the honest party.
3. Upon receiving a message $(\mathsf{sid}, \texttt{deliver})$ from the honest party, send (sid, y) to the adversary.
4. Upon receiving a message $(\mathsf{sid}, \texttt{ack})$ from the adversary, send (sid, y) to P_1 and (sid, y) to P_2.

Corrupt:

5. Upon receiving a message $(\mathsf{sid}, \texttt{pas\_corrupt}, \mathsf{C})$ from the adversary, send the input of the corrupted party C $(\mathsf{sid}, x_{\mathsf{C}})$ corresponding to sid to the adversary.
6. Upon receiving a message $(\mathsf{sid}, \texttt{act\_corrupt}, \mathsf{C})$ from the adversary, send the input of the corrupted party C $(\mathsf{sid}, x_{\mathsf{C}})$ corresponding to sid to the adversary. Additionally, after receiving the $(\texttt{deliver})$ message from the honest party H, send $(\mathsf{sid}, x_{\mathsf{H}})$ to the adversary.

Figure 8: Ideal functionality for semi-active two-party computation.

We shall briefly argue that most natural "passively secure" protocols actually satisfy this type of security in the *actively secure* OT hybrid model:

- Garbled circuits [Yao86] provide active security against the receiver, i.e. one direction is trivial. On the other hand, up to the point when the sender learns the label of the receiver, any active cheating of the sender (e.g. using a wrong circuit) gives no information.

- In the passively secure GMW protocol [GMW87], the computation phase ends with both parties holding a share (and having seen only random shares during the computation), which gives away nothing on the result. Only after the share reconstruction do the parties learn the effects of their cheating.

We will later provide an additional example of a passively secure protocol that satisfies our definition.

Protocol Structure: In order to chain several computation steps of a server in the outer protocol, the different instances of ρ have to be correlated. Suppose for example we use garbled circuits as an inner protocol, and want to chain two computation steps of a server in Π. Then, since the adversary must not learn the result of the first computation step, the labels of the result have to be used as input labels for the second computation step. Only after this step is performed as well can the output be announced.

This type of input-output mapping is not unique to garbled circuits and can be performed with at least the majority of passively secure protocols. In particular, this allows us to implement server-to-server communication without having to reveal intermediate results by simply routing the messages through the clients as inputs for the next round/computation step.

In order to minimize the communication between the parties, we use the following optimization of the inner functionality. If a computation by Π only relies on the local state μ_i^j, does not affect the internal state σ^j and is deterministic, we call this a *Type I* computation. All other computations are *Type II* computations. Observe that the client $\mathsf{C_P}$ has μ_{P}^j available, as well as the inputs w_{P}^j and can therefore compute perform this evaluation of π locally and send the result to the other client. Note, that this client-to-client communication is allowed, because it is completely determined by messages from the inner protocol.

6.4 The Compiled Resettable Protocol

The compiled protocol $\phi_{\Pi,\rho}^{dROT}$ will provide resettable security for a party S against a resetting party U. The compiler proceeds as follows. In a first step, U selects the k servers which it wants to have on its watchlist. U inputs the choices into a k-out-of-n OT based on $\mathcal{F}_{\mathrm{dROT}}$ and obtains the blinded keys for the watchlist channels, while S stores the keys for all servers. Both parties commit to their watchlist keys, but do not announce them yet.

Then, *all* OTs for the inner protocols are precomputed with random inputs by U. Due to the definition of $\mathcal{F}_{\mathrm{dROT}}$, S receives its own random inputs from $\mathcal{F}_{\mathrm{dROT}}$. These two steps are formally described in Figure 9.

Now, both parties engage in a simulated execution of the servers of Π. A formal description of the simulated execution of Π is shown in Figure 10. In a first step, the precomputed OTs are derandomized, using a variant of the derandomization technique of Beaver [Bea95]. We cannot directly use the normal derandomization procedure, because an adversary can learn both inputs via resets in the protocol of [Bea95]. Instead, we again apply a PRF to the correction value z

Resettability Compiler $\phi^{dROT}_{\Pi,\rho}$

Let ρ be an ℓ-round semi-actively secure protocol with parties $(\mathsf{P_S}, \mathsf{P_U})$ and Π be an honest-majority protocol with active security and clients $(\mathsf{C_S}, \mathsf{C_U})$ and servers $(S^1, \ldots, S^n)$. Let COM be a UC-secure commitment scheme.

Watchlist generation:

1. U: Draw a string $\mathbf{s} = (s_1, \ldots, s_n)$ with $HW(\mathbf{s}) = k$ and input $\mathbf{s}$ into $\mathcal{F}^{\text{k\_n}}_{\text{dROT}}$ to obtain k random values $(\alpha_1, \ldots, \alpha_k)$. Choose a random vector $\mathbf{w}_\mathsf{U}$ of sufficient length for the watchlists and compute $(c^j_\mathsf{U}, d^j_\mathsf{U}) \leftarrow \mathsf{COM.Commit}(w^j_\mathsf{U})$. Send $\mathbf{c}_\mathsf{U}$ to S.
2. S: Upon receiving $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)$ from $\mathcal{F}^{\text{k\_n}}_{\text{dROT}}$, compute $\beta_i = \mathsf{PRF}(\alpha_i)$ and set $\mathbf{w}_\mathsf{S} = \boldsymbol{\alpha} \oplus \boldsymbol{\beta}$. Use $\mathbf{w}_\mathsf{S}$ for the n watchlists. Compute $(c_{\boldsymbol{\beta}}, d_{\boldsymbol{\beta}}) \leftarrow \mathsf{COM.Commit}(\boldsymbol{\beta})$ and send $c_{\boldsymbol{\beta}}$ to U.

OT precomputation:

3. U: Sample $\mathbf{e} = (e^1, \ldots, e^t)$ with $e^i \in_\text{R} \{0,1\}$ and send for each $e^i, i \in [t]$ $(\mathsf{sid}, \mathsf{ssid}, \texttt{receiver}, e^i)$ to $\mathcal{F}_{\text{dROT}}$. Obtain t random strings $\mathbf{r}^\mathsf{U} = (r_{e^1}, \ldots, r_{e^t})$.
4. S: Upon receiving $2t$ strings $\mathbf{r}^\mathsf{S} = \big((r^1_0, r^1_1), \ldots, (r^t_0, r^t_1)\big)$ from $\mathcal{F}_{\text{dROT}}$, store $\mathbf{r}^\mathsf{S}$.

Figure 9: Watchlist setup and OT precomputation of $\phi^{dROT}_{\Pi,\rho}$.

sent by U and then use this *combined* with the random strings obtained in the watchlist selection and OT precomputation to fix the randomness of the outer protocol. Additionally, we derive the randomness for the inner protocols from the randomness of the outer protocol. This ensures that each new input in the outer protocol basically enforces a new random tape for S, which reduces reset attacks to active attacks.

Once the OTs are derandomized, the parties execute the inner protocol for each server in parallel up to the last round, which w.l.o.g. outputs the result of the computation. Due to the semi-active security of the inner protocol, even active attacks do not result in a security breach. However, if we would allow U to use the watchlists from the beginning of the protocol (as in [IPS08]), he would learn outputs of the inner protocol, although the output phase of the inner protocol has not begun. This would break the security of the inner protocol, and is the reason that both parties have to commit to their views prior to the output phase.

Both parties send their view over the watchlists, which basically serve as a commitment to their computation. Still, at this point neither party can access the watchlists, because the keys have not been announced. In the last round of the inner protocol, S applies a PRF to all watchlists of U to obtain his selection of watchlists. This challenge is sent to U, who has to unveil the keys for the challenged watchlists. S verifies the consistency of the challenged watchlists and in turn announces his own keys, so that U can check the consistency.

This approach guarantees that U cannot predict the challenge (e.g. by resetting) because each change in the execution of the inner protocols leads to a new challenge. This mechanism provides the same security as the watchlist in [IPS08], but is resettably secure. Once both parties agree that the watchlists are consistent, the inner protocol can be completed such that the parties learn the output.

We use the PRFs a bit sloppy in the formal description to keep the presentation clean. Assume that each PRF has a random seed, and is chosen such that the output length fits the respective application (if necessary by applying a PRG to the result). All seeds combined form the hard-coded random tape of S.

Resettability Compiler $\phi^{dROT}_{\Pi,\rho}$ (cont'd)

Simulated execution of Π: Upon input $(\mathsf{sid}, \mathsf{ssid}, \mathtt{input}, \mathsf{U}, x_\mathsf{U})$, U draws r^U_Π and $r^j_\mathsf{U}, j \in [n]$ uniformly at random and starts the execution of $\mathsf{C_U}$ with input $(x_\mathsf{U}, r^\mathsf{U}_\Pi)$. Let the output of $\mathsf{C_U}$ be the messages $m^1_\mathsf{U}, \ldots, m^n_\mathsf{U}$ to be sent to the servers.

1. U: Start the derandomization of the OTs for round i of $\rho^j, j \in [n]$ according to the following protocol. Let γ^j be the subset of OTs required by ρ^j in round i.
   (a) U: Let $\mathbf{e}_{|\gamma^j}$ be the subset of $\mathbf{e}$ corresponding to server j and round i, and $\mathbf{c}_{|\gamma^j}$ be the actual OT inputs (derived from m^j_U and r^j_U). Set $\mathbf{z}_{|\gamma^j} = \mathbf{e}_{|\gamma^j} \oplus \mathbf{c}_{|\gamma^j}$ and send $\mathbf{z}_i = (\mathbf{z}^1_{|\gamma^j}, \ldots, \mathbf{z}^n_{|\gamma^j})$ to S.
   (b) S: Upon receiving $\mathbf{z}_i$, compute $r^\mathsf{S}_\Pi = \mathsf{PRF}(\mathbf{z}_i, \mathbf{w}, \mathbf{r})$ and $\mathbf{r}_\rho = \mathsf{PRF}(\mathbf{z}_i, r^\mathsf{S}_\Pi)$, where $\mathbf{r}_\rho = (r^1_\rho, \ldots, r^n_\rho)$. Start $\mathsf{C_S}$ with input $(x_\mathsf{S}, r^\mathsf{S}_\Pi)$ and let $m^1_\mathsf{S}, \ldots, m^n_\mathsf{S}$ be the output. For all $j \in [n]$, start ρ^j with input $(m^j_\mathsf{S}, r^j_\rho)$. This yields γ^j tuples of OT inputs $\big((s^1_0, s^1_0), \ldots, (s^{\gamma^j}_0, s^{\gamma^j}_1)\big)$. For $l \in [\gamma^j]$, if $z^l = 0$, set $(\tilde{s}^l_0, \tilde{s}^l_1) = (s^l_0 \oplus r^l_0, s^l_1 \oplus r^l_1)$. If $z^l = 1$, set $(\tilde{s}^l_0, \tilde{s}^l_1) = (s^l_0 \oplus r^l_1, s^l_1 \oplus r^l_0)$. Send $(\tilde{\mathbf{s}}_0, \tilde{\mathbf{s}}_1)_{\gamma^j}$ for all $j \in [n]$ to U.
   (c) U: Upon receiving j tuples $(\tilde{\mathbf{s}}_0, \tilde{\mathbf{s}}_1)_{\gamma^j}$, use $\mathbf{r}^\mathsf{U}_{|\gamma^j}$ to obtain $\mathbf{s}_{\mathbf{c}_{\gamma^j}}$.
2. S: Continue the simulation of $\rho^j, j \in [n]$ and forward all messages from $\mathsf{P_S}$ to U. Send all inputs and outputs of ρ^j over watchlist W^j_S encrypted with w^j_S.
3. U: Start the execution of $\rho^j, j \in [n]$ with input $(m^j_\mathsf{U}, r^j_\mathsf{U})$ as party $\mathsf{P_U}$. Send all inputs and outputs of ρ^j over watchlist W^j_U encrypted with w^j_U and go back to Step 1 until $i = \ell$.
4. S: Compute $\mathsf{ch} = \mathsf{PRF}(\mathbf{W}_\mathsf{U})$, where $|\mathsf{ch}| = n$, $HW(\mathsf{ch}) = k$ and send ch to U.
5. U: Upon receiving ch, do for $j \in [n]$: if $\mathsf{ch}[j] = 1$, set $u^j = (d^j_\mathsf{U}, w^j_\mathsf{U})$. Send $\mathbf{u} = (u^1, \ldots, u^k)$ to S.
6. S: Upon receiving $\mathbf{u}$, check for all $j \in [k]$ if $\mathsf{COM.Open}(c^j_\mathsf{U}, d^j_\mathsf{U}, w^j_\mathsf{U}) = 1$ and check the consistency of all watchlists W^j_U, including consistency between watchlists. If the checks pass, send $(d_{\boldsymbol{\beta}}, \boldsymbol{\beta})$ to U. Resume ρ and let $\mathbf{o}_\mathsf{S}$ be the output. Simulate $\mathsf{C_S}$ with input $\mathbf{o}_\mathsf{S}$ and let y_S be the output. Output $(\mathsf{sid}, \mathsf{ssid}, y_\mathsf{S})$.
7. U: Check if $\mathsf{COM.Open}(c_{\boldsymbol{\beta}}, d_{\boldsymbol{\beta}}, \boldsymbol{\beta}) = 1$, and abort if not. Set $w^j_\mathsf{S} = \alpha^j \oplus \beta^j$ and check the consistency of the watchlists W^j_S. If no inconsistency occurred, resume ρ and let $\mathbf{o}_\mathsf{U}$ be the output. Simulate $\mathsf{C_U}$ with input $\mathbf{o}_\mathsf{U}$ and let y_U be the output. Output $(\mathsf{sid}, \mathsf{ssid}, y_\mathsf{U})$.

Figure 10: Simulation of the outer protocol in $\phi^{dROT}_{\Pi,\rho}$.

Theorem 2. *Let $\mathcal{F}^{\text{res}}_{\text{2PC}}$ be a 2-party functionality. Suppose Π is an 1-round* outer MPC protocol realizing $\mathcal{F}$ with adaptive security, with $n = \Theta(4k)$ and $t = \Theta(k)$, for a statistical security parameter k. Let $\mathcal{G}$ be the functionality as defined in Section 6.3 and ρ a protocol that securely realizes $\mathcal{G}$ in the OT-hybrid model with semi-active security. Then the compiled protocol $\phi^{dROT}_{\Pi,\rho}$ securely realizes $\mathcal{F}^{\text{res}}_{\text{2PC}}$ in the $\mathcal{F}_{\text{dROT}}$-hybrid model with resettable security. $\phi^{dROT}_{\Pi,\rho}$ has communication complexity $\mathcal{O}(C_\Pi + nrC_\rho)$, round complexity $\mathcal{O}(r_\Pi r_\rho)$, and invokes $\mathcal{F}_{\text{dROT}}$ $\mathcal{O}(nr_\Pi q_\rho)$ times, where C_Π is the communication complexity of Π, r_Π is the number of rounds in Π, C_ρ is the communication plus randomness complexity of ρ, and q_ρ is the number of invocations of $\mathcal{F}_{\text{dROT}}$ in ρ.*

Proof. Our proof is conceptually very similar to the proof of [IPS08]. We have to make some adaptations regarding the simulation, in particular we require two different simulators for the two parties due to the asymmetry of the parties.

Corrupted U. We will show that there exists a simulator $\mathcal{S}_{\mathsf{U}}$ (cf. Figure 11) for the protocol $\phi_{\Pi,\rho}^{dROT}$ that essentially mimics the behavior of an adversary in the outer protocol Π, which corrupts at most t parties. We can then use the UC simulator $\mathcal{S}_{\Pi}$ of Π to simulate the rest of the protocol. In order to handle resets by $\mathcal{A}_{\mathsf{U}}$, we simulate $\mathcal{F}$ for $\mathcal{S}_{\Pi}$ and forward its outputs to $\mathcal{F}_{\text{2PC}}^{\text{res}}$. In the following let $\mathcal{A}_{\mathsf{U}}$ be the dummy adversary.

The simulator always uses the same random values for the corresponding input from $\mathcal{A}_{\mathsf{U}}$ so as to simulate consistently if $\mathcal{A}_{\mathsf{U}}$ inputs the same value twice (e.g. after a reset). In order to maintain good readability, this is shown only in Step 2 and omitted later. The same technique can be used for every situation where $\mathcal{S}_{\mathsf{U}}$ needs to draw random values. In particular, every time that $\mathcal{S}_{\mathsf{U}}$ receives a message from $\mathcal{A}_{\mathsf{U}}$ which it has not received before, it appends that message to the view. If some time in the future a new message arrives with the same prefix, this implies an inconsistency in the execution (independent of possibly wrong inputs, just due to the fact that all messages are derived deterministically from the OT precomputation), and $\mathcal{S}_{\mathsf{U}}$ will in turn corrupt $\hat{S}^j$.

We now have to show that no environment can distinguish a simulation of $\mathcal{S}_{\mathsf{U}}$ from an actual protocol run of $\phi_{\Pi,\rho}^{dROT}$. Additionally, we have to show that $\mathcal{S}_{\mathsf{U}}$ corrupts at most t servers in $\hat{\Pi}$. We argue that the simulation of $\mathcal{S}_{\mathsf{U}}$ is computationally indistinguishable from a real protocol run. $\mathcal{S}_{\rho}$'s output is (computationally or statistically) indistinguishable from a protocol run of ρ. All randomness used by S is pseudorandom and thus indistinguishable from the random values used by $\mathcal{S}_{\mathsf{U}}$. Further, $\mathcal{A}_{\mathsf{U}}$ cannot abort depending on the inputs due to the use of random inputs into the OTs.

Experiment 0: This is the real protocol.

Experiment 1: Identical to Experiment 0, except that $\mathcal{S}_1$ uses new (but fixed) randomness in each step.

Experiment 2: Identical to Experiment 1, except that S_2 uses the extractor Ext of COM to learn all keys $\hat{\mathbf{w}}_{\mathsf{U}}$ and aborts of more than $\frac{t}{2}$ servers are corrupted.

Experiment 3: Identical to Experiment 2, except that S_3 learns all OT inputs by simulating $\mathcal{F}_{\text{dROT}}$.

Experiment $4.\{1,\ldots,n\}$**:** Identical to Experiment 3, except that $\mathcal{S}_4$ replaces all executions of ρ^j with executions of $\mathcal{S}_{\rho}^j$

Experiment 5: Identical to Experiment 4, except that $\mathcal{S}_5$ replaces all calls to $\mathcal{G}^j$ by directly relaying the messages from S^j.

- Experiments 0 and 1 are computationally indistinguishable due to the security of the pseudorandom function.
- Experiment 1 and Experiment 2 are indistinguishable due to the UC-security of COM and the consistency checks. Since S_2 succeeds with the extraction of all commitments with overwhelming probability, the only possibility to distinguish both experiments is if $\mathcal{A}_{\mathsf{U}}$ manages to corrupt more than $\frac{t}{2}$ servers without S aborting. An inconsistency occurs only if $\mathcal{A}_{\mathsf{U}}$ uses a different value in an execution of ρ than the one announced over the watchlist. All inconsistencies are fixed once $\mathcal{A}_{\mathsf{U}}$ sends the message over the corresponding watchlist. If S asks $\mathcal{A}_{\mathsf{U}}$

Simulator $\mathcal{S}_{\mathsf{U}}$

Let $\mathcal{S}_\Pi$ simulate an instance of $\hat{\Pi}$ with servers $\hat{S}^j$. Simulate $\mathcal{F}$ for $\mathcal{S}_\Pi$. All messages directed at servers are actually sent to $\mathcal{S}_\Pi$ and then forwarded to the simulated servers. Responses from the servers are relayed to $\mathcal{A}_{\mathsf{U}}$. Whenever $\mathcal{S}_\Pi$ sends a message x to $\mathcal{F}$, check if a tuple (x', y) is stored with $x' = x$. If yes, return y, otherwise set $x' = x$ and send $(\mathsf{sid}, \mathsf{ssid}, \mathtt{reset})$ to $\mathcal{F}^{\mathrm{res}}_{\mathrm{2PC}}$. Then send $(\mathsf{sid}, \mathsf{ssid}, \mathsf{U}, x)$ and obtain $(\mathsf{sid}, \mathsf{ssid}, y)$. Store (x', y) and send y to $\mathcal{S}_\Pi$.

1. Watchlist generation: Simulate $\mathcal{F}^{\mathrm{k\_n}}_{\mathrm{dROT}}$ for $\mathcal{A}_{\mathsf{U}}$ and learn $\hat{\mathbf{s}} = (\hat{s}_1, \ldots, \hat{s}_n)$. Store the values $\hat{\boldsymbol{\alpha}} = (\hat{\alpha}_1, \ldots, \hat{\alpha}_n)$. Corrupt all servers $\hat{S}^j$ in $\hat{\Pi}$ where $\hat{s}_j = 1$ and learn the corresponding inputs m^j_{H} of the honest parties. Draw a random vector $\hat{\boldsymbol{\beta}}$ uniformly at random and compute $(\hat{c}_{\boldsymbol{\beta}}, \hat{d}_{\boldsymbol{\beta}}) \leftarrow \mathsf{COM.Commit}(\hat{\boldsymbol{\beta}})$. Use $\hat{\mathbf{w}} = \hat{\boldsymbol{\alpha}} \oplus \hat{\boldsymbol{\beta}}$ to encrypt the watchlists and send $\hat{c}_{\boldsymbol{\beta}}$ to $\mathcal{A}_{\mathsf{U}}$.
2. OT precomputation: Simulate $\mathcal{F}_{\mathrm{dROT}}$ for $\mathcal{A}_{\mathsf{U}}$ and learn $\hat{\mathbf{e}}$. Store $\hat{\mathbf{r}} = \big((\hat{r}^1_0, \hat{r}^1_1), \ldots, (\hat{r}^t_0, \hat{r}^t_1)\big)$. If a tuple $(\hat{\boldsymbol{\alpha}}', \hat{\mathbf{r}}', \hat{r}'_\Pi)$ exists with $\hat{\boldsymbol{\alpha}}' = \hat{\boldsymbol{\alpha}}$ and $\hat{\mathbf{r}}' = \hat{\mathbf{r}}$, set $\hat{r}_\Pi = \hat{r}'_\Pi$, otherwise sample a new $\hat{r}_\Pi$ and store the tuple $(\hat{\boldsymbol{\alpha}}, \hat{\mathbf{r}}, \hat{r}_\Pi)$.
3. Simulated execution of Π:
   (a) Derandomization: for each server instance $j \in [n]$, learn $\mathcal{A}_{\mathsf{U}}$'s input $\hat{\mathbf{c}}_{|\gamma^j} = \mathbf{z}^*_{|\gamma^j} \oplus \mathbf{e}^*_{|\gamma^j}$. From this, derive the input $(\hat{m}^j_{\mathsf{U}}, \hat{r}^j_{\mathsf{U}})$.
   (b) Execution of ρ^j for all j:
   - If $\hat{s}_j = 1$, i.e. server j is on a watchlist of $\mathcal{A}_{\mathsf{U}}$, sample a random value r^j_{H} and execute an instance of ρ with input $(m^j_{\mathsf{H}}, r^j_{\mathsf{H}})$ honestly. Send all messages over watchlist W^j.
   - If $\hat{s}_j = 0$, start the simulator $\mathcal{S}^j_\rho$ with input $(\hat{m}^j_{\mathsf{U}}, \hat{r}^j_{\mathsf{U}})$ learned during the derandomization. Simulate $\mathcal{G}^j$ for $\mathcal{S}^j_\rho$ by relaying all messages to $\hat{S}^j$. Forward the answer $\hat{m}^j$ and simulate W^j accordingly.

   (c) Consistency check: perform a consistency check by randomly selecting $\hat{\mathsf{ch}}$, $HW(\hat{\mathsf{ch}}) = k$ and send $\hat{\mathsf{ch}}$ to $\mathcal{A}_{\mathsf{U}}$. Let $\mathbf{u}^*$ be the answer from $\mathcal{A}_{\mathsf{U}}$. Check consistency according to $\phi^{dROT}_{\Pi,\rho}$, and abort if S would abort. Use the extractor for COM on input $(\mathbf{c}^*_{\mathsf{U}})$ to learn all keys $\hat{\mathbf{w}}_{\mathsf{U}}$ and check all watchlists for consistency. If an unchecked watchlist j' is not consistent, proceed as follows:
   - Corrupt $\hat{S}^{j'}$ and learn all inputs between $\hat{S}^{j'}$ and the clients. Reconstruct the inputs of S input into the simulated $\mathcal{G}^j$ from this.
   - Let $\mathcal{S}^j_\rho$ corrupt all clients adaptively, and send all messages between $\mathcal{G}^j$ and the clients to $\mathcal{S}^j_\rho$.
   - $\mathcal{S}^j_\rho$ outputs the current state of each client in ρ^j.

   If more than $\frac{t}{2}$ views are inconsistent, abort. Send $\hat{d}_{\boldsymbol{\beta}}$ to $\mathcal{A}_{\mathsf{U}}$.

   (d) Output result: continue with the simulation of $\mathcal{S}^j_\rho$ for all $j \in [n]$ with input $\hat{m}^j$ obtained from the (corrupted and uncorrupted) servers $\hat{S}^j$.

Figure 11: Simulator against a corrupted U in $\phi^{dROT}_{\Pi,\rho}$.

to open watchlist W^j_{U} during the consistency check, it spots any inconsistency that is not an OT input or output with probability 1. $\mathcal{A}_{\mathsf{U}}$ can only cheat by using a wrong value in an OT and committing to a different value in this case. Then, $\mathcal{A}_{\mathsf{U}}$ is detected only with probability $\frac{1}{2}$.

A server can thus be corrupted at most with probability $\frac{1}{2}$, or if it is not challenged via ch. Each server is challenged with probability $\frac{k}{n}$, and thus the probability that $\mathcal{S}_{\mathsf{U}}$ has can corrupt more than $\frac{t}{2}$ servers without S aborting is bounded by $(1-\frac{k}{n})^{t/2} = 2^{\Omega(k)}$.

- Experiments 2 and 3 are identical since $\mathcal{S}_3$ simply simulates the OTs for $\mathcal{A}_{\mathsf{U}}$.
- We show the indistinguishability of Experiment 3 and Experiment 4 in a series of n hybrids. In the j-th experiment, we replace the simulations $\mathcal{S}_\rho^1, \dots, \mathcal{S}_\rho^{j-1}$ and the corresponding functionalities $\mathcal{G}^1, \dots, \mathcal{G}^{j-1}$ by sessions of ρ. Experiments j and $j+1$ differ exactly in one session of ρ. In the $(j+1)$-st experiment, $\mathcal{S}_\rho^j$ simulates the protocol as long as $\mathcal{A}_{\mathsf{U}}$ remains honest. If there is an inconsistency in the committed views that does not lead to an abort, $\mathcal{S}_\rho^j$ gets the state of S from $\mathcal{S}_{\mathsf{U}}$ and then executes ρ^j from S's state adaptively. This is always possible, because the commitment COM on the watchlist keys is hiding, thus preventing $\mathcal{A}_{\mathsf{U}}$ from learning results before the output phase of ρ commences. Thus, Experiments j and $j+1$ continue identically from this point.
- Experiment 4 and Experiment 5 are identically distributed, because the simulated $\mathcal{G}^j$ behave identical to S^j in $\hat{\Pi}$, thus we can replace all $\mathcal{G}^j$ by S^j in $\hat{\Pi}$.

Combined with the fact that $\mathcal{A}_{\mathsf{U}}$ obtains $\frac{t}{2}$ servers on the watchlist, we have that $\mathcal{S}_{\mathsf{U}}$ only corrupts t servers in Π, thus the simulation of $\mathcal{S}_\Pi$ is indistinguishable for any distinguishing environment, and will correctly output the input of $\mathcal{A}_{\mathsf{U}}$ to $\mathcal{F}$, which $\mathcal{S}_{\mathsf{U}}$ relays to $\mathcal{F}_{\text{2PC}}^{\text{res}}$. For each reset of an adversary in the real protocol, this procedure is repeated, leading to correctly extracted inputs for $\mathcal{F}_{\text{2PC}}^{\text{res}}$.

Corrupted S. We will show that there exists a simulator $\mathcal{S}_{\mathsf{S}}$ (cf. Figure 12) for the protocol $\phi_{\Pi,\rho}^{dROT}$ that essentially mimics the behavior of an adversary in the outer protocol Π, which corrupts at most t parties. We can then use the UC simulator $\mathcal{S}_\Pi$ of Π to simulate the rest of the protocol. In the following let $\mathcal{A}_{\mathsf{S}}$ be the dummy adversary.

We now show that no PPT environment can distinguish an execution of Π from an execution of $\phi_{\Pi,\rho}^{dROT}$. In a first step, the simulator learns the seeds $\hat{\mathbf{w}}$ for all watchlists, i.e. he can see all of $\mathcal{A}_{\mathsf{S}}$'s inputs into the inner protocols $\rho^j, j \in [n]$. Additionally, $\mathcal{S}_{\mathsf{S}}$ learns all inputs into the inner protocol by simulating $\mathcal{F}_{\text{dROT}}$ to $\mathcal{A}_{\mathsf{S}}$. We show in a series of hybrid experiments that the simulation of $\mathcal{S}_{\mathsf{S}}$ is computationally indistinguishable from a real execution.

Experiment 0: This is the real protocol.

Experiment 1: Identical to Experiment 0, except that $\mathcal{S}_1$ uses the simulator Sim of COM to equivocate all values in the commitment and sending $\hat{d}$ instead of d to $\mathcal{A}_{\mathsf{U}}$.

Experiment 2: Identical to Experiment 1, except that S_2 learns all OT inputs by simulating $\mathcal{F}_{\text{dROT}}$.

Experiment 3: Identical to Experiment 2, except that S_3 uses the extractor Ext of COM to learn all keys $\hat{\mathbf{w}}$.

Experiment 4: Identical to Experiment 3, except that S_4 aborts if more than $\frac{t}{2}$ servers are corrupted during the protocol run.

Simulator $\mathcal{S}_{\mathsf{S}}$

Let $\mathcal{S}_\Pi$ simulate an instance of $\hat{\Pi}$ with servers $\hat{S}^j$. Simulate $\mathcal{F}$ for $\mathcal{S}_\Pi$. All messages directed at servers are actually sent to $\mathcal{S}_\Pi$ and then forwarded to the simulated servers. Responses from the servers are relayed to $\mathcal{A}_{\mathsf{S}}$. Whenever $\mathcal{S}_\Pi$ sends a message x to $\mathcal{F}$, send $(\mathsf{sid}, \mathsf{ssid}, \mathsf{U}, x)$ to $\mathcal{F}_{\text{2PC}}^{\text{res}}$ and obtain $(\mathsf{sid}, \mathsf{ssid}, y)$. Relay y to $\mathcal{S}_\Pi$.

1. Watchlist generation: Simulate $\mathcal{F}_{\text{dROT}}^{\text{k\_n}}$ for $\mathcal{A}_{\mathsf{S}}$ and learn $\hat{\boldsymbol{\alpha}} = (\hat{\alpha}_1, \ldots, \hat{\alpha}_n)$. Compute $(\hat{c}_{\mathsf{U}}^j, \hat{d}_{\mathsf{U}}^j) \leftarrow \mathsf{COM.Commit}(\hat{\mathbf{w}}_{\mathsf{U}})$ and send $\hat{\mathbf{c}}_{\mathsf{U}}$ to $\mathcal{A}_{\mathsf{S}}$. Start the extractor Ext of COM on $c_{\boldsymbol{\beta}}^*$ to learn all values $\hat{\beta}$ and set $\hat{w}^j = \hat{\alpha_j} \oplus \hat{\beta}_j$. During the simulation of Π, check the consistency of *all* watchlists.
2. OT precomputation: Simulate $\mathcal{F}_{\text{dROT}}$ for $\mathcal{A}_{\mathsf{S}}$ and learn $\hat{\mathbf{r}}_{\mathsf{S}}$.
3. Simulated execution of Π:
   (a) Derandomization: for each server instance $j \in [n]$, draw a random string $\hat{\mathbf{z}}_{\gamma^j}$ and send it to $\mathcal{A}_{\mathsf{S}}$. From $(\tilde{\mathbf{s}}_0^*, \tilde{\mathbf{s}}_1^*)_{\gamma^j}$, learn $\mathcal{A}_{\mathsf{S}}$'s input $(\hat{\mathbf{s}}_0, \hat{\mathbf{s}}_1)_{\gamma^j}$.
   (b) Execution of ρ^j for all j: start the simulator $\mathcal{S}_\rho^j$ with inputs $(\hat{m}_{\mathsf{S}}^j, \hat{r}_{\mathsf{S}}^j)$ from the watchlist W_j. Simulate $\mathcal{G}^j$ for $\mathcal{S}_\rho^j$ by relaying all messages to $\hat{S}^j$ and forwarding the answers $\hat{m}^j$.
   (c) Consistency checks: Simulate all consistency checks that U would do, and abort if U aborts. If an inconsistency occurs in any watchlist, and less than $\frac{t}{2}$ servers are corrupted, proceed as follows, otherwise abort:
      - Corrupt $\hat{S}^{j\prime}$ and learn all inputs between $\hat{S}^{j\prime}$ and the clients. Reconstruct the inputs of U input into the simulated $\mathcal{G}^j$ from this.
      - Let $\mathcal{S}_\rho^j$ corrupt all clients adaptively, and send all messages between $\mathcal{G}^j$ and the clients to $\mathcal{S}_\rho^j$.
      - $\mathcal{S}_\rho^j$ outputs the current state of each client in ρ^j.
      - Continue the simulation of $\mathcal{S}_\rho^j$ from this state.
   (d) Output result: In the last round let ch^* be the challenge of $\mathcal{A}_{\mathsf{S}}$.
      - Corrupt all servers $\hat{S}^j$ for which $\mathsf{ch}[l]^* = 1, l \in [n]$. For each such server j, learn the inputs $\hat{m}_{\mathsf{H}}^j$ of the honest party, sample a random $\hat{r}_{\mathsf{H}}^j$ and perform a correct execution of ρ^j with inputs $(\hat{m}_{\mathsf{H}}^j, \hat{r}_{\mathsf{H}}^j)$ and $(\hat{m}_{\mathsf{S}}^j, \hat{r}_{\mathsf{S}}^j)$. Let τ^j be the result.
      - For each $\hat{c}_{\mathsf{U}}^j$, start the simulator for COM with input τ^j to equivocate the respective commitments to values $\tilde{d}_{\mathsf{U}}^j$. Let $\hat{u}_i^j = (\tilde{d}_{\mathsf{S}}^j, \tau^j)$ be the decommitment to $\hat{c}_{\mathsf{S}}^j$.

   Send $\hat{\mathbf{u}}$ to $\mathcal{A}_{\mathsf{S}}$. Continue with the simulation of $\mathcal{S}_\rho^j$ for all $j \in [n]$ from their current states.

Figure 12: Simulator against a corrupted S.

Experiment 5.$\{1, \ldots, n\}$**:** Identical to Experiment 4, except that $\mathcal{S}_5$ replaces all executions of ρ^j with executions of $\mathcal{S}^j$

Experiment 6: Identical to Experiment 5, except that $\mathcal{S}_6$ replaces all calls to $\mathcal{G}^j$ by directly relaying the messages from S^j.

- Experiments 0 and 1 are indistinguishable due to the UC-security of COM, i.e. Sim will succeed with overwhelming probability to equivocate the commitments.
- Experiment 1 and Experiment 2 are identically distributed since $\mathcal{S}_2$ simply simulates $\mathcal{F}_{\text{dROT}}$.

- Experiment 3 and Experiment 4 are indistinguishable by the same argumentation as above, we can therefore bound the probability that $\mathcal{S}_{\mathsf{S}}$ has can corrupt more than $\frac{t}{2}$ servers without U aborting by $(1 - \frac{k}{n})^{t/2} = 2^{\Omega(k)}$.

- We show the indistinguishability of Experiment 4 and Experiment 5 in a series of n hybrids. In the j-th experiment, we replace the simulations $\mathcal{S}_{\rho}^{1}, \dots, \mathcal{S}_{\rho}^{j-1}$ and the corresponding functionalities $\mathcal{G}^{1}, \dots, \mathcal{G}^{j-1}$ by sessions of ρ. Experiments $5.j$ and $5.j+1$ differ exactly in one session of ρ. In the $(j+1)$-st experiment, $\mathcal{S}_{\rho}^{j}$ simulates the protocol as long as $\mathcal{A}_{\mathsf{U}}$ remains honest. If there is an inconsistency in the committed views that does not lead to an abort, $\mathcal{S}_{\rho}^{j}$ gets the state of S from $\mathcal{S}_{\mathsf{U}}$ and then executes ρ^{j} from S's state adaptively. Thus, Experiments $5.j$ and $5.j+1$ continue identically from this point. An environment distinguishing between both experiments can thus be used to break the semi-active (adaptive) security of ρ.

- Experiment 4 and Experiment 5 are identically distributed, because the simulated $\mathcal{G}^{j}$ behave identical to S^{j} in $\hat{\Pi}$, thus we can replace all $\mathcal{G}^{j}$ by S^{j} in $\hat{\Pi}$.

It remains to show that the amount of corrupted servers is upper bounded by t. By the same argumentation as above, since the challenge w^* only requires k unveils, it follows by our choice of parameters that at most t servers will be corrupted. Thus $\mathcal{S}_{\Pi}$ will send the correct value x to $\mathcal{F}$, which means $\mathcal{S}_{\mathsf{S}}$ will input the correct value input into $\mathcal{F}_{\mathrm{2PC}}^{\mathrm{res}}$. □

Remark. In the above described compiler, we assumed that ρ remains secure even if the parties choose the randomness $r_{\mathsf{S}}^{j}, r_{\mathsf{U}}^{j}$ maliciously. Let us briefly sketch how we can execute a "coin-toss into the well" in the compiler to remove this assumption. For each server j, both parties draw a random string s_{S}^{j} and s_{U}^{j}, respectively, and send it to the other party. Now the party $\mathsf{P} \in \{\mathsf{S}, \mathsf{U}\}$ is supposed to use $r_{\mathsf{P}}^{j} \oplus s_{\mathsf{P}}^{j}$. This step can be verified during the consistency checks of the parties.

6.5 Applications

6.5.1 Constant-Rate Resettably Secure Computation in the $\mathcal{F}_{\mathrm{dROT}}$-hybrid model

By applying our compiler to results from the literature, we obtain constant-rate resettably secure computation. Both of the following results are based on the same outer protocol (up to some minor changes regarding the circuit type). As it is, we cannot directly use their outer protocols: in both cases, the clients perform operations for each layer of the circuit. This means that the outer protocol is not a 1-round* protocol and thus cannot be used with the compiler.

However, we argue that a simple transformation of the protocol leads to the correct structure for our compiler. As indicated above, both clients perform verification steps for each layer of the circuit. We propose that this verification be outsourced to two servers, and only the input sharing is done by the clients. Since the original protocol is secure in the presence of a malicious client, it will remain secure in case that the servers are corrupted. This transformed protocol is now a 1-round* outer protocol and can thus be used with our compiler.

Corollary 9 (of Theorem 2 and [IPS08])**.** *Let C be a boolean circuit of size s and depth d for a two-party functionality f. There exists a two-party protocol Π in the $\mathcal{F}_{\mathrm{dROT}}$-hybrid model, whose total communication complexity is $O(s + \mathsf{poly}(k, d, \log s))$ and which computationally UC-realizes $\mathcal{F}_{\mathrm{2PC}}^{\mathrm{res}}$.*

For this result, Ishai et al. [IPS08] propose the standard passively secure GMW protocol as the inner protocol. As we sketched in Section 6.3, this protocol is also semi-actively secure and thus the above corollary follows immediately. For our second result, we have to argue that the inner protocol, which is proposed in [IPS09], is semi-actively secure given actively secure OT. We will sketch this briefly here: in the protocol, two parties jointly compute a secret sharing of a multiplication. One party creates a noisy encoding of its input and sends it to the other party. This party now performs a multiplication with its own input, but does not directly send the result to the first party. Instead, both parties engage in an OT where the first party chooses the noiseless position of the codeword. From this, both parties can compute the shares.

Now, if the first party is corrupted, it can only choose the noisy encoding maliciously (remember that we have an actively secure OT, in contrast to the original protocol). Thus, up to the point when the OT outputs are delivered, the passive security of the protocol prevents the first party from learning anything (it gets no feedback at all). Similarly, if the second party is corrupted, it might choose inconsistent values for the multiplication. This leads to inconsistent shares, but until a share is announced by the first party, there is also no feedback and due to the passive security the second party learns nothing.

Corollary 10 (of Theorem 2 and [IPS09])**.** *Let C be an arithmetic circuit of size s and depth d for a two-party functionality f. There exists a two-party protocol Π in the $\mathcal{F}_{\mathrm{dROT}}$-hybrid model, whose total communication complexity is $O(s + kd)$ and which computationally UC-realizes $\mathcal{F}_{\mathrm{2PC}}^{\mathrm{res}}$.*

6.5.2 Constant-Round Resettably Secure Computation

For completeness, we show how to obtain constant-round resettable two-party computation protocol in the $\mathcal{F}_{\mathrm{dROT}}$-hybrid model, which only requires black-box use of a one-way function. The result directly follows from a modification of the constant-round protocol of Damgård and Ishai [DI05], which is presented in [IPS08]. In the original protocol of [DI05], the server uses a pseudorandom generator to encrypt a message. Using this protocol directly as the outer protocol Π is not possible, because this would require non-black-box use of the PRG in the inner protocol. Instead, Ishai et al. [IPS08] propose a modification of the protocol where the computation of the PRG is delegated to the clients. The resulting protocol is a 1-round* outer protocol and can therefore be used with $\phi_{\Pi,\rho}^{dROT}$.

Our compiler itself makes black-box use of a PRF, but both PRF and PRG can be constructed in a black-box way from one-way functions. Combined, this yields:

Corollary 11 (of Theorem 2 and [DI05, IPS08])**.** *There exists a constant-round two-party protocol in the $\mathcal{F}_{\mathrm{dROT}}$-hybrid model which makes black-box use of a one-way function and computationally UC-realizes $\mathcal{F}_{\mathrm{2PC}}^{\mathrm{res}}$.*

6.5.3 Non-Interactive UC-Secure Computation from Untrusted Resettable Tamper-Proof Hardware

In combination with the results of Döttling et al. [DMMQN13], the compiler can directly be used to obtain non-interactive UC-secure computation. They present a protocol based on resettable tamper-proof hardware tokens that creates a resettable UC-secure CRS with two tokens. A sender can simply store the program of S of the compiler $\phi_{\Pi,\rho}^{dROT}$ with its input on the tokens, and then send the tokens to U.

Previously, in order to use their protocol, it was necessary to modify either a UC-secure computation protocol in the $\mathcal{F}_{\text{CRS}}$-hybrid model like [CLOS02] such that it becomes resettable, or the resettability compiler of [GS09] such that it can be based on a CRS.

Acknowledgement. We would like to thank Ivan Damgård, Antonio Faonio, Jörn Müller-Quade and Jesper Buus Nielsen for many helpful discussions. We also thank the anonymous reviewers for their comments on an earlier draft of this paper.

References

[Bea95] Donald Beaver. Precomputing oblivious transfer. In Don Coppersmith, editor, *CRYPTO'95*, volume 963 of *LNCS*, pages 97–109. Springer, Heidelberg, August 1995.

[BGGL01] Boaz Barak, Oded Goldreich, Shafi Goldwasser, and Yehuda Lindell. Resettably-sound zero-knowledge and its applications. In *42nd FOCS*, pages 116–125. IEEE Computer Society Press, October 2001.

[Blu81] Manuel Blum. Coin flipping by telephone. In Allen Gersho, editor, *CRYPTO'81*, volume ECE Report 82-04, pages 11–15. U.C. Santa Barbara, Dept. of Elec. and Computer Eng., 1981.

[BLV03] Boaz Barak, Yehuda Lindell, and Salil P. Vadhan. Lower bounds for non-black-box zero knowledge. In *44th FOCS*, pages 384–393. IEEE Computer Society Press, October 2003.

[BOGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *20th ACM STOC*, pages 1–10. ACM Press, May 1988.

[BP13] Nir Bitansky and Omer Paneth. On the impossibility of approximate obfuscation and applications to resettable cryptography. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 241–250. ACM Press, June 2013.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In *20th ACM STOC*, pages 11–19. ACM Press, May 1988.

[CDD$^+$15] Ignacio Cascudo, Ivan Damgård, Bernardo Machado David, Irene Giacomelli, Jesper Buus Nielsen, and Roberto Trifiletti. Additively homomorphic UC commitments with optimal amortized overhead. In Jonathan Katz, editor, *PKC 2015*, volume 9020 of *LNCS*, pages 495–515. Springer, Heidelberg, March / April 2015.

[CGGM00] Ran Canetti, Oded Goldreich, Shafi Goldwasser, and Silvio Micali. Resettable zero-knowledge (extended abstract). In *32nd ACM STOC*, pages 235–244. ACM Press, May 2000.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *34th ACM STOC*, pages 494–503. ACM Press, May 2002.

[COP$^+$14] Kai-Min Chung, Rafail Ostrovsky, Rafael Pass, Muthuramakrishnan Venkitasubramaniam, and Ivan Visconti. 4-round resettably-sound zero knowledge. In Yehuda Lindell, editor, *TCC 2014*, volume 8349 of *LNCS*, pages 192–216. Springer, Heidelberg, February 2014.

[CPS13] Kai-Min Chung, Rafael Pass, and Karn Seth. Non-black-box simulation from one-way functions and applications to resettable security. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 231–240. ACM Press, June 2013.

[CR03] Ran Canetti and Tal Rabin. Universal composition with joint state. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 265–281. Springer, Heidelberg, August 2003.

[DFG$^+$11] Yi Deng, Dengguo Feng, Vipul Goyal, Dongdai Lin, Amit Sahai, and Moti Yung. Resettable cryptography in constant rounds - the case of zero knowledge. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 390–406. Springer, Heidelberg, December 2011.

[DGN10] Ivan Damgård, Martin Geisler, and Jesper Buus Nielsen. From passive to covert security at low cost. In Daniele Micciancio, editor, *TCC 2010*, volume 5978 of *LNCS*, pages 128–145. Springer, Heidelberg, February 2010.

[DGS09] Yi Deng, Vipul Goyal, and Amit Sahai. Resolving the simultaneous resettability conjecture and a new non-black-box simulation strategy. In *50th FOCS*, pages 251–260. IEEE Computer Society Press, October 2009.

[DI05] Ivan Damgård and Yuval Ishai. Constant-round multiparty computation using a black-box pseudorandom generator. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 378–394. Springer, Heidelberg, August 2005.

[DMMQN13] Nico Döttling, Thilo Mie, Jörn Müller-Quade, and Tobias Nilges. Implementing resettable UC-functionalities with untrusted tamper-proof hardware-tokens. In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 642–661. Springer, Heidelberg, March 2013.

[DPV04] Giovanni Di Crescenzo, Giuseppe Persiano, and Ivan Visconti. Constant-round resettable zero knowledge with concurrent soundness in the bare public-key model. In Matthew Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 237–253. Springer, Heidelberg, August 2004.

[GIP15] Daniel Genkin, Yuval Ishai, and Antigoni Polychroniadou. Efficient multi-party computation: From passive to active security via secure SIMD circuits. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 721–741. Springer, Heidelberg, August 2015.

[GKOV12] Sanjam Garg, Abishek Kumarasubramanian, Rafail Ostrovsky, and Ivan Visconti. Impossibility results for static input secure computation. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 424–442. Springer, Heidelberg, August 2012.

[GM11] Vipul Goyal and Hemanta K. Maji. Stateless cryptographic protocols. In Rafail Ostrovsky, editor, *52nd FOCS*, pages 678–687. IEEE Computer Society Press, October 2011.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987.

[Gol01] Oded Goldreich. *Foundations of Cryptography: Basic Tools*, volume 1. Cambridge University Press, Cambridge, UK, 2001.

[GS09] Vipul Goyal and Amit Sahai. Resettably secure computation. In Antoine Joux, editor, *EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 54–71. Springer, Heidelberg, April 2009.

[IKO$^+$11] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, Manoj Prabhakaran, and Amit Sahai. Efficient non-interactive secure computation. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 406–425. Springer, Heidelberg, May 2011.

[IKOS07] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In David S. Johnson and Uriel Feige, editors, *39th ACM STOC*, pages 21–30. ACM Press, June 2007.

[IPS08] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer - efficiently. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 572–591. Springer, Heidelberg, August 2008.

[IPS09] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Secure arithmetic computation with no honest majority. In Omer Reingold, editor, *TCC 2009*, volume 5444 of *LNCS*, pages 294–314. Springer, Heidelberg, March 2009.

[LP07] Yehuda Lindell and Benny Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In Moni Naor, editor, *EUROCRYPT 2007*, volume 4515 of *LNCS*, pages 52–78. Springer, Heidelberg, May 2007.

[LS91] Dror Lapidot and Adi Shamir. Publicly verifiable non-interactive zero-knowledge proofs. In Alfred J. Menezes and Scott A. Vanstone, editors, *CRYPTO'90*, volume 537 of *LNCS*, pages 353–365. Springer, Heidelberg, August 1991.

[MP06] Silvio Micali and Rafael Pass. Local zero knowledge. In Jon M. Kleinberg, editor, *38th ACM STOC*, pages 306–315. ACM Press, May 2006.

[MR01] Silvio Micali and Leonid Reyzin. Min-round resettable zero-knowledge in the public-key model. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 373–393. Springer, Heidelberg, May 2001.

[NP05] Moni Naor and Benny Pinkas. Computationally secure oblivious transfer. *Journal of Cryptology*, 18(1):1–35, January 2005.

[PVW08] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 554–571. Springer, Heidelberg, August 2008.

[PW09] Rafael Pass and Hoeteck Wee. Black-box constructions of two-party protocols from one-way functions. In Omer Reingold, editor, *TCC 2009*, volume 5444 of *LNCS*, pages 403–418. Springer, Heidelberg, March 2009.

[RS92] Charles Rackoff and Daniel R. Simon. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 433–444. Springer, Heidelberg, August 1992.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986.

[YZ07] Moti Yung and Yunlei Zhao. Generic and practical resettable zero-knowledge in the bare public-key model. In Moni Naor, editor, *EUROCRYPT 2007*, volume 4515 of *LNCS*, pages 129–147. Springer, Heidelberg, May 2007.

[ZDLZ03] Yunlei Zhao, Xiaotie Deng, Chan H. Lee, and Hong Zhu. Resettable zero-knowledge in the weak public-key model. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 123–139. Springer, Heidelberg, May 2003.