Resettably Secure Computation

Vipul Goyal* and Amit Sahai**

Department of Computer Science, UCLA

Abstract. The notion of resettable zero-knowledge (rZK) was introduced by Canetti, Goldreich, Goldwasser and Micali (FOCS'01) as a strengthening of the classical notion of zero-knowledge. A rZK protocol remains zero-knowledge even if the verifier can reset the prover back to its initial state anytime during the protocol execution and force it to use the same random tape again and again. Following this work, various extensions of this notion were considered for the zero-knowledge and witness indistinguishability functionalities.

In this paper, we initiate the study of resettability for more general functionalities. We first consider the setting of resettable two-party computation where a party (called the user) can reset the other party (called the smartcard) anytime during the protocol execution. After being reset, the smartcard comes back to its original state and thus the user has the opportunity to start interacting with it again (knowing that the smartcard will use the same set of random coins). In this setting, we show that it is possible to secure realize *all PPT computable functionalities* under the most natural (simulation based) definition. Thus our results show that in cryptographic protocols, the reliance on randomness and the ability to keep state can be made significantly weaker. Our simulator for the aforementioned resettable two-party computation protocol (inherently) makes use of non-black box techniques. Second, we provide a construction of *simultaneous* resettable multi-party computation with an honest majority (where the adversary not only controls a minority of parties but is also allowed to reset any number of parties at any point). Interestingly, all our results are in the plain model.

1 Introduction

The notion of resettable zero-knowledge (rZK) was introduced by Canetti et al [CGGM00] with a motivation towards obtaining zero-knowledge protocols for highly adversarial environments. In rZK, the verifier is given the additional power that anytime during the protocol execution, it can "reset" the prover back to its initial state thus restarting the prover with the same configuration and coin

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-3-642-01001-9_35

* Research supported in part from Amit Sahai's grants and a Microsoft Graduate Research Fellowship.

** Research supported in part from NSF grants 0627781, 0716389, 0456717, and 0205594, a subgrant from SRI as part of the Army Cyber-TA program, an equipment grant from Intel, an Alfred P. Sloan Foundation Fellowship, and an Okawa Foundation Research Grant.

A. Joux (Ed.): EUROCRYPT 2009, LNCS 5479, pp. 54–71, 2009.

tosses. This notion is motivated by several natural questions. Firstly, it address the question: "is zero-knowledge possible when the prover uses the *same random coins* in more than one execution?" and (surprisingly) gives a positive answer to it. Secondly, it shows that zero-knowledge protocols can be securely implemented by devices which can neither reliably keep state nor toss coins online. An example of such a device might be a resettable stateless "smartcard" with secure hardware (which can be reset, e.g., by switching off its power) [CGGM00]. Thus, rZK can be viewed as "zero-knowledge protocols for stateless devices". Canetti et al [CGGM00] provide rZK proof system (for NP) with non-constant number of rounds and resettable witness indistinguishable (rWI) proof system with a constant number of rounds.

Canetti et al [CGGM00] observed that since the prover can be reset, an adversarial verifier can actually achieve the effect of interacting with the prover *concurrently* in several sessions, thus in particular implying concurrent zero knowledge [DNS98]. In more detail, the verifier can start a session with the prover and while the interaction is still in progress, can reset the prover to start another interaction. It could reset the prover again in the middle to that interaction and *come back to the previous interaction* by just running the protocol identically up to the point where it reset the prover before. Thus, the verifier gets the flexibility of choosing its messages in one interaction based on the messages of the prover in some other interaction. In other words, the verifier can essentially interact with the prover in various sessions and can interleave these sessions as it pleases. Thus, a rZK protocol is a concurrent ZK protocol as well [CGGM00]. Following this work, research on improving the round complexity of concurrent ZK also led to rZK with improved round complexity [KP01, PRS02].

Subsequent to the introduction of this notion, various extensions were considered. Barak, Goldreich, Goldwasser and Lindell [BGGL01] studied *resettably-sound* ZK (rsZK) where it is the verifier that can be reset by the prover. The main challenge is to design a ZK protocol which retains its soundness even when verifier uses the same random coins in multiple executions. Relying the non black-box techniques of Barak [Bar01], Barak et al were able to construct constant round rsZK arguments. These rsZK arguments were also used to construct resettable zero-knowledge arguments of knowledge [BGGL01]. An open question raised by [BGGL01] is: do there exist ZK protocols where both the prover and the verifier are resettable by each other? While this still remains an open problem, partial progress was made in [DL07]. The notion of resettability was also studied extensively in the bare public key model introduced by [CGGM00] with a goal of obtaining more efficient protocols (see [YZ07] and the references therein).

Going Beyond ZK – Our Contributions. Common to all the prior work on the notion of resettability is that they consider only the zero-knowledge (or closely related) functionality. This raises the following natural question:

"Do there exist other classes of functionalities for which resettably secure protocols can be obtained?"

In other words, do there exist protocols for other tasks such that the "security" is retained even if one of parties participating in the protocol can be reset by the other one? We initiate the study of general resettable computation and answer the above question in the affirmative.

–**Resettable Two-Party Computation.** We prove a general completeness theorem for resettable computation showing that it is possible to construct a protocol to securely realize *any PPT computable functionality*. Our results are for the setting where one party (called the user) can reset the other party (called the smartcard) during the protocol execution. We first formalize this notion of resettable two-party computation and give a natural simulation based security definition for it. We then construct a general two-party computation protocol under standard (polynomial time) cryptographic assumption. We in fact give a "resettability compiler" which can compile any semi-honest secure (in the standalone setting) protocol into one that is resettably secure. The simulator for our resettable two-party computation protocol makes use of non-black box techniques. We note that non-black box simulation is inherent since Barak et al [BGGL01] show that it is impossible to obtain rsZK arguments (for languages outside $\mathcal{BPP}$) using black-box simulation and rsZK arguments for **NP** is only a special case of a two-party functionality.

–**Resettable Multi-Party Computation.** Given the above results, two natural questions that arise then are: (a) Do there exists general secure resettable *multi-party* computation protocols (where one or more of the parties are resettable)?, and, (b) Can the construction be extended to handle cases where both parties can potentially reset each other? Towards that end, we first observe that the our construction for resettable two-party computation can be extended using standard techniques to show the existence of resettable multi-party computation (where only one of the parties can be reset) with dishonest majority. Next, we offer a construction of *simultaneous* resettable multi-party computation (where all the parties are resettable) assuming a majority of the parties behave honestly. That is, the adversary not only controls a minority of parties but is also allowed to reset any number of honest parties at any point in the protocol execution. At the core of our construction is a new construction of families of multi-party 1-round (and thus automatically resettable) zero-knowledge arguments of knowledge which are simulation sound [Sah99].

Our results show that in cryptographic protocols, the reliance on randomness and the ability to keep state can be made significantly weaker. We in fact show a simple transformation from any resettably secure protocol (as per our definitions) to a fully stateless one. By this we mean that the party which was resettable in the original protocol need not maintain any state at all in the transformed protocol.

Concurrency vs Resettability. As discussed earlier, if a party can be reset, the other party can actually achieve the effect of interacting with it *concurrently* in several sessions. This fact is the reason for the folklore that a resettably secure protocol should also be concurrently secure (i.e., concurrently self-composable).

However far reaching impossibility results have been proven showing that a large class of functionalities cannot be securely realized [Lin03, Lin04] (even in the fixed roles setting) in the plain model. This stands in sharp contrast to our general positive results for resettable two-party computation.

In resettable two-party computation, an adversarial user already has the power to reset the smartcard and interact with it as many times as it likes. This fact that the number of interactions cannot be controlled is precisely what makes resettable two-party computation possible. In the formulation of our ideal model for defining security, we give the adversarial user the power to interact with the smartcard as many times it likes. Given that an adversarial user is allowed to reset the smartcard in the real model (thus creating new problems), emulating the view of such an adversary in the ideal world is only possible if several interactions with the smartcard are allowed. In other words, we are only allowing the user to do in the ideal world what he is already allowed to do in reality. By giving our ideal adversary such extra power, we are able to construct protocols satisfying our definition in the plain model. In our construction, the number of times the simulator sends the reset request in the ideal world is polynomially related to the number of times the real world adversary sends the reset request. An open problem raised by the current work is to design a construction having a *precise simulation* [MP06] with respect to the number of outputs.

Combining our results with the impossibility results of Lindell [Lin03, Lin04], we get a first separation between the notions of resetability and concurrency. That is, a resettably secure protocol (as per our definitions) is not always a concurrently secure protocol (even for fixed roles) and vice versa (this direction is implicit in [CGGM00]). In fact there exists a large class of functionalities for which concurrently self-composable protocols do not exist but resettably secure protocols do (as we show that they exist for every two-party functionality).

Meaningful Resettable Functionalities. We stress that while the resettable setting is unsuitable for several traditional functionalities, such as Yao's Millionaire function, it remains meaningful and highly non-trivial for a large class of functions. For instance, consider the problem of "conditional blind signatures", where one is willing to sign unknown messages as long as they satisfy some property P. If a resettable device were to use a traditional two-party computation protocol to do this, it might leak full knowledge of the secret signing key; our protocols would ensure that only the power to sign messages satisfying P is given to the holder of the device. In general, functionalities where the output is "cryptographic" in nature, and where the input of the device is a cryptographic key, will be meaningful for resettable devices in a variety of settings. Techniques from the area of privacy-preserving data analysis (see [Dwo08] and the references therein) may also be useful in designing other classes of functions suitable for computation by resettable devices.

Stateless vs Stateful Devices. Our results have interesting implications about the power of stateless devices. Consider a stateless device (holding an input, possibly unable to toss coins online) trying to run a protocol for secure computation of some functionality with a user. Our results show that *stateless devices can run*

secure computation protocols for every task. Of course, stateless devices can only be used when one does not want to limit the number of protocol executions that the device carries out (with a particular input).

What if one does want to limit the number of interactions that the device carries out? We remark that it is also possible to obtain the following "best of both worlds" protocol in such a case. *In case* the adversary is not able to reset the device during the protocol interaction, the protocol provides a traditional security guarantee (i.e., security as per the ideal/real world simulation paradigm, see Canetti [Can00]). However if it turns out that the adversary *was successfully* able to reset the device, the *maximum* the adversary can do is to achieve the effect of running the protocol several times with the device (possibly choosing different input each time).

While "protocols for stateless devices" are naturally useful for weak and inexpensive devices (e.g., smartcard), other applications of such protocols could include making a powerful server (providing services to many clients) stateless to prevent denial of service attacks.

Universally Composable Multi-Party Computation using Tamper Proof Hardware. To show one example of the power of our results, we consider the recent work on obtaining universally composable multi-party computation using tamper proof hardware tokens [Kat07]. As noted before, broad impossibility results have been proven showing that a large class of functionalities cannot be UC securely realized in the plain model [CF01, CKL06]. These severe impossibility results motivated the study of other models involving some sort of *trusted* setup assumptions (assuming a trusted third party), where general positive results can be obtained. To avoid these trust assumptions (while still maintaining feasibility of protocols), Katz recently proposed using a *physical setup*. In his model, the physical setup phase includes the parties exchanging tamper proof hardware tokens implementing some functionality.

The security of the construction in [Kat07] relies on the ability of the tamper-resistant hardware to maintain state (even when, for example, the power supply is cut off). In particular, the parties need to execute a four-round coin flipping protocol with the tamper-resistant hardware. Using our techniques, one can immediately relax this requirement and make the token completely stateless. In particular, we can apply our compiler to the coin flipping protocol in [Kat07] and obtain a new construction where the token, when fed with an input x, only outputs $f(x)$ for a fixed f and halts. A construction having such a property was first obtained recently by Chandran et al [CGS08] by relying on techniques that are very specific to the setting of UC secure computation with tamper proof hardware. However our construction has an added advantage that a possibly adversarial token does not learn any information about the input of the honest party using it (and hence the security is retained even when the adversarial creator of the token "recaptures" it at a later point of time). This is leads to the first construction of UC secure computation with stateless and recapturable tokens. On the downside, as opposed to [CGS08], the security of this construction would

rely on the adversary "knowing" the code of the tokens which it distributes to the honest parties (see [CGS08] for more details).

Open Problems. The main question left open by the current work is: "Do there exist two-party and multi-party computation protocols in the *dishonest majority* setting where more than one party is resettable?". Eventually, one would like to construct simultaneous resettable multi-party computation where the adversary can control any number of parties and can reset any number of honest parties at any point (or show that such protocols cannot exist). The apparent obstacle to making any progress towards answering the above questions is the long-standing problem of constructing a simultaneous resettable zero-knowledge argument (first mentioned as an open problem in the work of Barak et al [BGGL01]). We recently settled this conjecture affirmatively in [GS08].

2 The Resettable Ideal Model

2.1 Resettable Two-Party Computation

Informally speaking, our model for resettable two-party computation is as follows. We consider a smartcard $\mathbb{S}$ holding several inputs $x_1^1, \ldots, x_1^{num}$ and random tapes $\omega_1^1, \ldots, \omega_1^{num}$. We denote by X_1 the full input and randomness vector (i.e., the concatenation of all the inputs and random tapes) held by $\mathbb{S}$. The ith *incarnation* of the smartcard $\mathbb{S}$ is a deterministic strategy defined by the pair (x_1^i, ω_1^i) as in [CGGM00]. We stress that while each incarnation has its own random tape, as in [CGGM00], when a particular incarnation is reset, it starts over with the same random tape. Thus, we model different smartcards as the different incarnations. We consider a user $\mathbb{U}$ holding an input x_2 interested in interacting with the ith incarnation of the smartcard $\mathbb{S}$. The user activates the ith incarnation of the smartcard and runs a protocol with it to securely compute a function $f(.,.)$. We do not explicitly define a way of activating the ith incarnation; it could either be through physical means or by sending an initial message to $\mathbb{S}$ specifying the incarnation $\mathbb{U}$ would like to interact with. At any point during the protocol execution, the user $\mathbb{U}$ can *reset* the smartcard $\mathbb{S}$ to its initial state, thus, having a chance to start interaction again with any incarnation with a fresh input. At the end of the protocol, both the parties get the output $f(x_1, x_2)$, $x_1 = x_1^i$ where i and x_2 are the incarnation and the inputs which were most recently selected by the user $\mathbb{U}$. We remark that we only consider *one side resettability*, that is, the smartcard $\mathbb{S}$ is not allowed to reset the user $\mathbb{U}$.

To formalize the above requirement and define security, we extend the standard paradigm for defining secure computation. We define an ideal model of computation and a real model of computation, and require that any adversary in the real model can be *emulated* (in the specific sense described below) by an adversary in the ideal model. In a given execution of the protocol we assume that all inputs have length κ, the security parameter. We consider a static adversary which chooses whom to corrupt before execution of the protocol. In our model, both the parties get the same output (the case where parties should get different

outputs can be easily handled using standard techniques). Finally, we consider *computational* security only and therefore restrict our attention to adversaries running in probabilistic polynomial time.

IDEAL MODEL. In the ideal model there is a trusted party which computes the desired functionality based on the inputs handed to it by the players. Then an execution in the ideal model proceeds as follows:

Select incarnation. The user $\mathbb{U}$ sends an incarnation index i to the trusted party which then passes it on to the smartcard $\mathbb{S}$.

Inputs. The smartcard $\mathbb{S}$ has input x_1 while the user $\mathbb{U}$ has input x_2.

Send inputs to trusted party. Both $\mathbb{S}$ and $\mathbb{U}$ send their inputs to the trusted party. An honest party always sends its real inputs to the trusted party. A corrupted party, on the other hand, may decide to send modified value to the trusted party.

Trusted party computes the result. The trusted party sets the result to be $f(x_1, x_2)$. It generates and uses uniform random coin if required for the computation of f.

Trusted party sends results to adversary. The trusted party sends the result $f(x_1, x_2)$ to either $\mathbb{S}$ or $\mathbb{U}$ depending upon whoever is the adversary.

Trusted party sends results to honest players. The adversary, depending on its view up to this point, does the following. It either sends the *abort* signal in which case the trusted party sends $\perp$ to the honest party. Or it could signal the trusted party to *continue* in which case the trusted party sends the result $f(x_1, x_2)$ to the honest party.

Reset ideal world at any point. In case the user $\mathbb{U}$ is the adversary, during the execution of any of the above steps, it can send the signal *reset* to the trusted party. In that case, the trusted party sends *reset* to the smartcard $\mathbb{S}$ and the ideal world comes back to the *select incarnation* stage.

Outputs. An honest party always outputs the response it received from the trusted party. The adversary outputs an arbitrary function of its entire view throughout the execution of the protocol.

For a given adversary $\mathcal{A}$, the *execution of f in the ideal model* on X_1, x_2 is defined as the output of the honest parties along with the output of the adversary resulting from the process above. It is denoted by $\text{IDEAL}_{f,\mathcal{A}}(X_1, x_2)$.

REAL MODEL. An honest party follows all instructions of the prescribed protocol, while a adversarial party may behave arbitrarily. If the user $\mathbb{U}$ is the adversarial party, it can *reset* the smartcard $\mathbb{S}$ at any point during the protocol execution. After getting reset, $\mathbb{S}$ comes back to its original state which it was in when starting the protocol execution thus allowing $\mathbb{U}$ to choose a fresh input and start interaction again with any incarnation of the smartcard $\mathbb{S}$. At the conclusion of the protocol, an honest party computes its output as prescribed by the protocol. Without loss of generality, we assume the adversary outputs exactly its entire view of the execution of the protocol.

For a given adversary $\mathcal{B}$ and protocol Σ for resettably computing f, the *execution of Σ in the real model* on X_1, x_2 (denoted $\text{REAL}_{\Sigma,\mathcal{B}}(X_1, x_2)$) is defined as

the output of the honest parties along with the output of the adversary resulting from the above process.

Having defined these models, we now define what is meant by a resettable two-party computation protocol. By *probabilistic polynomial time* (PPT), we mean a probabilistic Turing machine with non-uniform advice whose running time is bounded by a polynomial in the security parameter κ. By *expected probabilistic polynomial time* (EPPT), we mean a Turing machine whose *expected* running time is bounded by some polynomial, for *all* inputs.

Definition 1. *Let f and Σ be as above. Protocol Σ is a* secure protocol for computing *f if for every* PPT *adversary $\mathcal{A}$ corrupting either of the two players in the real model, there exists an* EPPT *adversary $\mathcal{S}$ corrupting that player in the ideal model, such that:*

$$\{\text{IDEAL}_{f,\mathcal{S}}(X_1, x_2)\}_{(X_1,x_2)\in(\{0,1\}^*)^2} \stackrel{c}{\equiv} \{\text{REAL}_{\Sigma,\mathcal{A}}(X_1, x_2)\}_{(X_1,x_2)\in(\{0,1\}^*)^2}.$$

Our real model translates to the so called *multiple incarnation non-interleaving* setting in the terminology of [CGGM00]. This setting was shown to be equivalent to the *multiple incarnation interleaving* setting for the case of zero-knowledge and their proof can be extended to the general case as well. In other words, a protocol Σ which is secure when the user $\mathbb{U}$ is allowed only to interact with one incarnation at a time remain secure even if $\mathbb{U}$ is allowed to *concurrently* interact with any number of incarnation simultaneously. For simplicity of exposition, we only consider the setting when the inputs of smartcard $\mathbb{S}$ are all fixed in advance (while $\mathbb{S}$ is acting as honest party in the protocol). However we remark that our protocols also work for the more general case when the inputs of $\mathbb{S}$ are adaptively chosen possibly based on the outputs in the previous protocol executions. More details regarding these issues will be provided in the full version.

2.2 Simultaneous Resettable Multi-party Computation

For lack of space, we defer the model for this case to the full version of this paper. The main changes from the two-party case is that we consider an adversary who controls a minority of the parties and can reset any number of honest parties at any point during the protocol.

2.3 Extensions

In this section, we informally describe two extensions which can be applied to our constructions proven secure as per our definitions. More formal details will be provided in the full version.

Going from Resettable to Stateless. Any protocol which is resettably secure can be transformed to a stateless protocol using relatively standard techniques. In other words, the parties which were allowed to be resettable in the original protocol need not maintain any state at all in the transformed protocol. By a

stateless device we mean that the device only supports a "request-reply" interaction (i.e., the device just outputs $f(x)$ when fed with x for some fixed f). We describe the case of two party first assuming both the parties are resettable (the case where one party is resettable is only simpler). Let we have parties P_1 and P_2 participating in the original resettably secure protocol Σ. Now we define a protocol Σ' having parties P_1' and P_2'. Each of these parties will have a secret key of a CCA-2 secure encryption scheme and a secret MAC key. The party P_1' computes the first message to be sent in the protocol Σ' by running P_1 internally. However it then sends to P_2' not only the computed message but also an encryption of the *current state* of P_1 and a MAC on it. Party P_2' similarly computes the reply by feeding the received message to P_2 and sends to P_1' not only the computed reply but also (a) the received encrypted state of P_1 and the MAC, and, (b) an encryption of the current state of P_2 and a MAC on it using its own keys. Thus for the next round, P_1' can decrypt, verify and *load* the received state into P_1, feed it the incoming reply and then compute the next outgoing message. P_2 can similarly continue with the protocol. The case of multi-party protocols can also be handled by first transforming the given protocol into one where only *one party* sends a message in any given round and then applying ideas similar to the one for the two party case to this resulting protocol.

Getting the Best of Both Worlds. One might ask the question: is it possible to have a single protocol such that *in case* the adversary is not able to reset the device, the protocol provides a traditional security guarantee (i.e., security as per the ideal/real world simulation paradigm, see Canetti [Can00]). However if it turns out that the adversary *is successfully* able to reset the device, the protocol still provides security as per the resettable ideal model definition presented above. We remark that it is easy to transform both our constructions into ones which provide such a best of both worlds guarantee (however we do not know if our transformation works for all constructions). We build a counter into the device which gets incremented with every protocol execution (whether successfully completed or not). The randomness used by the device for a protocol execution comes from the application of a PRF to the current counter value. This guarantees that in case the device *is* able to keep such a counter successfully, the randomness used in each execution is fresh and independent of others. Thus, it is easy to show that one can use a significantly simpler simulator (which can only handle standalone executions) to prove security of our constructions in such a setting.

3 Building Blocks

Our protocols make use of the following building blocks: a commitment scheme COM based on one way permutations, computational zero-knowledge proofs and proofs of knowledge, zaps [DN00], resettable sound zero-knowledge arguments [BGGL01] and the PRS concurrent zero-knowledge preamble [PRS02].

4 Resettable Two-Party Computation

4.1 The Construction

We now describe how to transform any given two party protocol Π (which is only semi-honest secure) into a resettably secure protocol Σ. Prior to the beginning of the protocol, we assume that the smartcard $\mathbb{S}$ and the user $\mathbb{U}$ have agreed upon the incarnation of $\mathbb{S}$ for the protocol. Each incarnation of the smartcard $\mathbb{S}$ has its own independent random tape. We assume that the private inputs to $\mathbb{S}$ and $\mathbb{U}$ in the protocol Π are x_1 and x_2 respectively. The smartcard $\mathbb{S}$ denotes the party which can be reset by the other party $\mathbb{U}$ in the protocol Σ. We view the random tape of the smartcard as a tuple (G, R_{rs}). Here G denotes the description of a function $G : \{0,1\}^{\leq \text{poly}(\kappa)} \rightarrow \{0,1\}^{\text{poly}(\kappa)}$ taken from an ensemble of pseudorandom functions and R_{rs} denotes a random string which $\mathbb{S}$ will use while acting as a verifier of a *resettable sound zero-knowledge* argument. Let R denote the uniform distribution. The protocol proceeds as follows.

PRS Preamble Phase

1. $\mathbb{U} \rightarrow \mathbb{S}$: Generate $r_2 \stackrel{\$}{\leftarrow} R$ and let $\beta = (x_2, r_2)$. Here r_2 is the randomness to be used (after coin flipping with $\mathbb{S}$) by the user $\mathbb{U}$ at various stages of the protocol Σ (including to carry out the protocol Π) as explained later on. We assume that r_2 is of sufficient size to allow $\mathbb{U}$ to execute all such stages. Generate random shares $\{\beta^0_{i,\ell}\}^k_{i,\ell=1}, \{\beta^1_{i,\ell}\}^k_{i,\ell=1}$ such that $\beta^0_{i,\ell} \oplus \beta^1_{i,\ell} = \beta$ for every i, ℓ. Using the commitment scheme COM, commit to all these shares. Denote these commitments by $\{B^0_{i,\ell}\}^k_{i,\ell=1}, \{B^1_{i,\ell}\}^k_{i,\ell=1}$.
   Let msg be the concatenation of all these commitment strings, i.e., $msg = B^0_{1,1}||\ldots||B^0_{k,k}\,||B^1_{1,1}||\ldots||B^1_{k,k}$. We call msg to be the *determining message* of this session (since it commits the user $\mathbb{U}$ to its input and randomness). The random tape used by the smartcard $\mathbb{S}$ to carry out rest of the protocol Σ (except when acting as a verifier of a resettable sound ZK argument) will be determined by the application of the pseudorandom function G to the determining message msg. Again, we assume that $G(msg)$ is of sufficient size to allow the execution of all the steps.
2. $\mathbb{U} \leftrightarrow \mathbb{S}$: The user $\mathbb{U}$ and the smartcard $\mathbb{S}$ will now use a resettable-sound zero-knowledge argument system (rsP, rsV) (relying a non-black box simulator [BGGL01]). $\mathbb{U}$ emulates the prover rsP and proves the following statement to the resettable verifier rsV (emulated by $\mathbb{S}$): the above PRS commit phase is a *valid commit phase* In other words, there exist values $\hat{\beta}$, $\{\hat{\beta}^0_{i,\ell}\}^k_{i,\ell=1}$, $\{\hat{\beta}^1_{i,\ell}\}^k_{i,\ell=1}$ such that (a) $\hat{\beta}^0_{i,\ell} \oplus \hat{\beta}^1_{i,\ell} = \hat{\beta}$ for every i, ℓ, and, (b) Commitments $\{B^0_{i,\ell}\}^k_{i,\ell=1}, \{B^1_{i,\ell}\}^k_{i,\ell=1}$ can be decommitted to $\{\hat{\beta}^0_{i,\ell}\}^k_{i,\ell=1}, \{\hat{\beta}^1_{i,\ell}\}^k_{i,\ell=1}$.
   The user $\mathbb{U}$ uses a fresh (uncommitted) random tape for emulation of the prover rsP. The random tape used by $\mathbb{S}$ to emulate the resettable verifier rsV comes from R_{rs}. Going forward, this will be the case with all the resettable sound zero-knowledge arguments in our protocol Σ.

3. For $\ell = 1, \ldots k$:
   (a) $\mathbb{S} \rightarrow \mathbb{U}$: Send challenge bits $b_{1,\ell}, \ldots, b_{k,\ell} \stackrel{\$}{\leftarrow} \{0,1\}^k$.
   (b) $\mathbb{U} \rightarrow \mathbb{S}$: Decommit to $B_{1,\ell}^{b_{1,\ell}}, \ldots, B_{k,\ell}^{b_{k,\ell}}$.
   The random tape required by $\mathbb{S}$ to generate the above challenge bits comes from $G(msg)$.
4. $\mathbb{S} \rightarrow \mathbb{U}$: Since the underlying protocol Π is secure only against semi-honest adversaries, the random coins used by each party are required to be unbiased. Hence $\mathbb{S}$ generates $r_2' \stackrel{\$}{\leftarrow} R$ (using the random tape from $G(msg)$) and sends it to $\mathbb{U}$. Define $r_2'' = r_2 \oplus r_2'$. Now r_2'' is the randomness which will be used by $\mathbb{U}$ for carrying out the protocol Π (among other things).

Smartcard Input Commitment Phase

1. $\mathbb{S} \rightarrow \mathbb{U}$: Generate a string $r_1 \stackrel{\$}{\leftarrow} R$ and let $\alpha = (x_1, r_1)$. Commit to α using the commitment scheme COM and denote the commitment string by A. The random tape required to generate the string r_1 and to compute the commitment A comes from $G(msg)$.
2. $\mathbb{S} \leftrightarrow \mathbb{U}$: Now $\mathbb{S}$ has to give a proof of knowledge of the opening of the commitment A to $\mathbb{U}$. In other words, $\mathbb{S}$ has to prove that it knows a value $\hat{\alpha} = (\hat{x_1}, \hat{r_1})$ such that the commitment A can be decommitted to $\hat{\alpha}$. This proof is given as follows. $\mathbb{S}$ and $\mathbb{U}$ use an ordinary computational zero-knowledge proof of knowledge system (P_{pok}, V_{pok}) where $\mathbb{S}$ and $\mathbb{U}$ emulates the prover P_{pok} (proving the above statement) and the verifier V_{pok} respectively. However the random tape used by $\mathbb{U}$ to emulate the verifier V_{pok} is required to come from r_2''. To achieve this, the interaction proceeds as follows. Let t'_{pok} be the number of rounds in (P_{pok}, V_{pok}) where a round is defined to have a message from P_{pok} to V_{pok} followed by a reply from V_{pok} to P_{pok}. For $j = 1, \ldots t'_{pok}$:
   - $\mathbb{S} \rightarrow \mathbb{U}$: $\mathbb{S}$ sends the next prover message computed as per the system (P_{pok}, V_{pok}). The random tape used by $\mathbb{S}$ to emulate P_{pok} comes from $G(msg)$.
   - $\mathbb{U} \rightarrow \mathbb{S}$: $\mathbb{U}$ sends the next verifier message computed as per the system (P_{pok}, V_{pok}) *using randomness* r_2''.
   - $\mathbb{U} \leftrightarrow \mathbb{S}$: $\mathbb{U}$ and $\mathbb{S}$ now execute a resettable-sound zero-knowledge argument where $\mathbb{U}$ emulates the prover rsP and proves the following statement to rsV emulated by $\mathbb{S}$: the PRS commit phase has a major decommitment $\hat{\beta} = (\hat{x_2}, \hat{r_2})$ such that the sent verifier message is consistent with the randomness $\hat{r_2} \oplus r_2'$ (where r_2' was as sent by $\mathbb{S}$ in the PRS preamble phase).

   The above system can be seen as a *resettable zero-knowledge argument of knowledge* system [BGGL01]. However when used in our context as above, the simulator of this system will be straightline.
3. $\mathbb{U} \rightarrow \mathbb{S}$: $\mathbb{U}$ generates $r_1' \stackrel{\$}{\leftarrow} R$ using the random tape r_2'' and sends it to $\mathbb{S}$. Define $r_1'' = r_1 \oplus r_1'$. Now r_1'' is the randomness which will be used by $\mathbb{S}$ for carrying out the protocol Π.

4. $\mathbb{U} \leftrightarrow \mathbb{S}$: $\mathbb{U}$ and $\mathbb{S}$ now execute a resettable-sound zero-knowledge argument. $\mathbb{U}$ emulates the prover rsP and proves the following statement to rsV emulated by $\mathbb{S}$: the PRS commit phase has a major decommitment $\hat{\beta} = (\hat{x_2}, \hat{r_2})$ such that message r'_1 is consistent with the randomness $\hat{r_2} \oplus r'_2$.

Secure Computation Phase

Let the underlying protocol Π have t rounds[1] where one round is defined to have a message from $\mathbb{S}$ to $\mathbb{U}$ followed by a reply from $\mathbb{U}$ to $\mathbb{S}$. Let transcript T_1^j (resp. T_2^j) be defined to contain all the messages exchanged between $\mathbb{S}$ and $\mathbb{U}$ before the point party $\mathbb{S}$ (resp. $\mathbb{U}$) is supposed to send a message in round j. Now, each message sent by either party in the protocol Π is compiled into a *message block* in Σ. For $j = 1, \ldots t$:

1. $\mathbb{S} \rightarrow \mathbb{U}$: $\mathbb{S}$ sends the next message m_1^j $(= \Pi(T_1^j, x_1, r''_1))$ as per the protocol Π. Now $\mathbb{S}$ has to prove to $\mathbb{U}$ that the sent message m_1^j was honestly generated using input x_1 and randomness r''_1. In other words, $\mathbb{S}$ has to prove the following statement: there exist a value $\hat{\alpha} = (\hat{x_1}, \hat{r_1})$ such that: (a) the message m_1^j is consistent with the input $\hat{x_1}$ and the randomness $\hat{r_1} \oplus r'_1$ (i.e., $m_1^j = \Pi(T_1^j, \hat{x}, \hat{r_1} \oplus r'_1)$), and, (b) commitment A can be decommitted to $\hat{\alpha}$. This proof is given as follows. $\mathbb{S}$ and $\mathbb{U}$ use an ordinary computational zero-knowledge proof system (P, V) where $\mathbb{S}$ and $\mathbb{U}$ emulates the prover P (proving the above statement) and the verifier V respectively. However the random tape used by $\mathbb{U}$ to emulate the verifier V is required to come from r''_2. To achieve this, the interaction proceeds as follows. Let t' be the number of rounds in (P, V) where a round is defined to have a message from P to V followed by a reply from V to P. For $j' = 1, \ldots t'$:
   - $\mathbb{S} \rightarrow \mathbb{U}$: $\mathbb{S}$ sends the next prover message computed as per the system (P, V). The random tape used by $\mathbb{S}$ to emulate P comes from $G(msg)$.
   - $\mathbb{U} \rightarrow \mathbb{S}$: $\mathbb{U}$ sends the next verifier message computed as per the system (P, V) *using randomness* r''_2.
   - $\mathbb{U} \leftrightarrow \mathbb{S}$: $\mathbb{U}$ and $\mathbb{S}$ now execute a resettable-sound zero-knowledge argument where $\mathbb{U}$ emulates the prover rsP and proves the following statement to rsV emulated by $\mathbb{S}$: the PRS commit phase has a major decommitment $\hat{\beta} = (\hat{x_2}, \hat{r_2})$ such that the sent verifier message is consistent with the randomness $\hat{r_2} \oplus r'_2$.

   To summarize, the random tape used by $\mathbb{U}$ to emulate the verifier V is required to be committed in advance. However $\mathbb{S}$ is free to use any random tape while emulating the prover P (although in the interest of its own security, $\mathbb{S}$ is instructed to use randomness from $G(msg)$.
2. $\mathbb{U}$ sends the next message m_2^j $(= \Pi(T_2^j, x_2, r''_2))$ as per the protocol Π. $\mathbb{U}$ and $\mathbb{S}$ now execute a resettable-sound zero-knowledge argument where $\mathbb{U}$

[1] This assumption is only made for simplicity of exposition. It is easy to extend our construction to handle protocols whose round complexity is not upper bounded by a fixed polynomial.

emulates the prover rsP and proves to $\mathbb{S}$ that m_2^j was honestly generated using input x_2 and randomness r_2''. More precisely, $\mathbb{U}$ proves the following statement: the PRS commit phase has a major decommitment $\hat{\beta} = (\hat{x_2}, \hat{r_2})$ such that m_2^j is consistent with the input $\hat{x_2}$ and the randomness $\hat{r_2} \oplus r_2'$.

This completes the description of the protocol Σ. The usage of randomness in the above protocol can be summarized as follows. After sending the very first message (i.e., the determining message msg), the only fresh randomness that can be used by the user $\mathbb{U}$ is while emulating the prover rsP of resettable sound zero-knowledge arguments. The smartcard $\mathbb{S}$ is essentially "free" to use any randomness it wants (except while computing messages of the underlying protocol Π). An honest $\mathbb{S}$ always sets its random tape to $G(msg)$ to carry out the protocol Σ .

At the end of above protocol Σ, both the parties will hold the desired output. We stress that we require only standard (standalone) semi-honest security from the underlying protocol Π. Thus, when we set the underlying protocol Π to be the constant round two-party computation protocol of Yao [Yao86], the resulting protocol Σ has k $(= \omega(\log \kappa))$ rounds. To obtain a constant round protocol, the first step would be the construction of a concurrent zero-knowledge argument system in a constant number of rounds. We also remark that the above resettable two-party computation also implies (under standard assumptions) resettable multi-party computation (where only one of the parties can be reset) with dishonest majority. The construction for resettable multi-party computation can be obtained using standard techniques from the two-party one (i.e., the $n-1$ "non-resettable" parties will use a regular multi-party computation protocol [GMW87] among them to emulate a single party holding $n-1$ inputs).

We prove that the protocol Σ is a resettable two-party computation protocol by proving the following two theorems; the proofs are deferred to the full version of this paper.

Theorem 1 (Security Against a Malicious $\mathbb{U}^*$). *The compiled protocol Σ is secure against a malicious $\mathbb{U}^*$.*

Theorem 2 (Security Against a Malicious $\mathbb{S}^*$). *The compiled protocol Σ is secure against a malicious $\mathbb{S}^*$.*

5 Simultaneous Resettable Multi-party Computation with Honest Majority

5.1 The Construction

We now describe how to transform any given protocol Π (which is only semi-honest secure) into a simultaneous resettably secure protocol Σ with honest majority. We assume n parties $P_1, \ldots, P_n$ where a majority of the parties behave honestly. All the parties are "resettable", or in other words, the adversarial parties can reset any number of honest parties at any time during the protocol

execution. We assume that before the protocol starts, the parties have agreed upon which incarnation will be used by which party. The private inputs of parties $P_1, \ldots, P_n$ are denoted by $x_1, \ldots, x_n$ respectively. Let R denote the uniform distribution. The protocol Σ proceeds as follows.

Input Commitment Phase

Each party P_i does the following computations. Any randomness required for these computations comes from the random tape of (appropriate incarnation of) P_i which is potentially reusable in other sessions.

- Generate a function $G_i : \{0,1\}^{\leq \text{poly}(\kappa)} \rightarrow \{0,1\}^{\text{poly}(\kappa)}$ randomly from an ensemble of pseudorandom functions and let $\alpha_i = (x_i, G_i)$. Compute a commitment to α_i using the commitment scheme COM and denote it by A_i.
- Generate the first verifier message $Z_i \leftarrow f_{zaps}(\kappa)$ of a zap system.
- Generate a pair (PK_i, SK_i) of public and secret keys using the key generation algorithm of a semantically secure public key encryption system having perfect completeness.
- Generate n strings $a_i^1, \ldots, a_i^n$ from the domain of a one way function F. For all j, compute $b_i^j = F(a_i^j)$.

Additionally, we assume that a party P_i has a random tape $R_{i,zkv}$ which it uses for the verification of messages of a 1 round ZKAOK system as we explain later on. P_i now broadcasts the values $A_i, Z_i, PK_i, b_i^1, \ldots, b_i^n$. Let the string broadcast (i.e., $A_i||Z_i||PK_i||\ b_i^1||\ldots||b_i^n$) be denoted by msg_i. The string msg_i is called the *determining message* of party P_i for this session. Note that since a party P_i may have to reuse its random tape, the determining message msg_i may be identical across various protocol executions.

The random tape used by P_i to carry out rest of the protocol Σ (except for the verification of the messages of the 1 round ZKAOK system) will be determined by the application of the pseudorandom function G_i to the (concatenation of) determining messages of all other parties. That is, denote $R_i = G_i(msg_1||\ldots||msg_{i-1}||msg_{i+1}||\ldots||msg_n)$. Now R_i serves as the random tape of P_i for the rest of the protocol. We assume that R_i is of sufficient size to allow the execution of all the steps.

Construction of a 1-round zero-knowledge argument of knowledge. We now describe the construction of a family of 1-round zero-knowledge argument of knowledge (ZKAOK) systems. The (i,j)th argument system is used by party P_i to prove statements to party P_j. Additionally, the argument systems (i,j) and (k,ℓ), where $i \neq k$, are simulation sound w.r.t. each other. To prove a statement $x \in L$ to party P_j, a party P_i holding the witness w (for the given witness relation) proceeds as follows:

- P_i breaks the witness w into n shares $w_1, \ldots, w_n$ using the Shamir threshold secret sharing scheme [Sha79] such that a majority of the shares are sufficient to reconstruct the witness w. For all k, P_i encrypts w_k under the public key PK_k. Let the ciphertext be denoted by C_k. P_i now broadcasts all the n ciphertexts so generated.

- P_i finally generates and sends the prover message of the zap system (acting on the verifier message Z_j) proving that one of the following statements is true:
  1. The ciphertexts $C_1, \ldots, C_n$ represent the encryption of shares of a valid witness. More precisely, there exists strings $\hat{w_1}, \ldots, \hat{w_n}$ such that: (a) for all k, C_k is a valid encryption of $\hat{w_k}$, and, (b) $\hat{w_1}, \ldots, \hat{w_n}$ are valid shares of a single string $\hat{w}$ as per the Shamir secret sharing scheme, and, (c) $\hat{w}$ is a valid witness for the witness relation (i.e., $x \in L$).
  2. The ciphertexts $C_1, \ldots, C_n$ represent the encryption of shares of a majority of preimages of the strings $b_1^i, \ldots, b_n^i$ under the one way function F. More precisely, there exists strings $\hat{s_1}, \ldots, \hat{s_n}$ such that: (a) for all k, C_k is a valid encryption of $\hat{s_k}$, and, (b) $\hat{s_1}, \ldots, \hat{s_n}$ are valid shares of a single string $\hat{s}$ as per the Shamir secret sharing scheme, and, (c) $\hat{s} = (\hat{a_1}^i, \ldots, \hat{a_n}^i)$ and there exists a set S_{maj} of indices such that $|S_{maj}| > n/2$ and for all $\ell \in S_{maj}$, $b_\ell^i = F(\hat{a_\ell}^i)$.

Thus, we have a *trapdoor condition* which allows a party P_i to give a simulated argument using the preimages of $b_1^i, \ldots, b_n^i$. Note that the "trapdoor" for each party is "independent". That is, informally speaking, even given the preimages of strings $b_1^i, \ldots, b_n^i$, a party P_j with $j \neq i$ will be unable to give a simulated argument.

Coin Flipping Phase

Since the underlying protocol Π is secure only against semi-honest adversaries, the random coins used by each party in Π are required to be unbiased. Hence the parties run a 2 round coin flipping phase to generate a long unbiased public random string as given below. As noted before, the random tape that a party P_i uses to execute this stage (i.e., generate of random strings, commitment and messages of the 1-round ZKAOK system) comes from R_i. However the random tape (if needed) used by P_i to verify the messages of the ZKAOK system comes from $R_{i,zkv}$.

- In the first round, each party P_i generates $R_i' \stackrel{\$}{\leftarrow} R$ and broadcasts a commitment B_i to R_i' using the commitment scheme COM. For all $j \neq i$, party P_i additionally broadcasts a ZKAOK (using the (i,j)th 1-round ZKAOK system) proving that the commitment B_i was correctly computed using randomness R_i. More precisely, P_i proves that there exists a string $\hat{\alpha_i} = (\hat{x_i}, \hat{G_i})$ such that: (a) the commitment A_i can be decommitted to $\hat{\alpha_i}$, and, (b) the commitment B_i to a random string (and the random string itself) was computed using randomness $\hat{G_i}(msg_1||\ldots||msg_{i-1}||msg_{i+1}||\ldots||msg_n)$. Note that this ZKAOK is given for a specific witness relation such that the witness allows extraction of such a $\hat{\alpha_i}$.
- In the second round, each party P_i broadcasts the committed string R_i' (without providing any decommitment information). For all $j \neq i$, party P_i additionally broadcasts a ZKAOK (using the (i,j)th 1-round ZKAOK system) proving that the commitment B_i can be decommitted to the string

R'_i. Denote $r'_1||\ldots||r'_n = R'_1 \oplus \cdots \oplus R'_n$. At this point, the strings $r'_1, \ldots, r'_n$ are all guaranteed to be random.

Each P_i further privately generates a random r_i. Define $r''_i = r_i \oplus r'_i$. Now r''_i is the randomness that will be used by party P_i to carry out the underlying protocol Π in the next stage.

Secure Computation Phase

Let the underlying protocol Π have t rounds[2] where any number of parties can send a message in any given round. Let transcript T^j be defined to contain all the messages broadcast *before* the round j. Now, for $j = 1, \ldots t$:

1. P_i sends the next message m_i^j $(= \Pi(T^j, x_i, r''_i))$ as per the protocol Π. Note that m_i^j could potentially be $\perp$.
2. For all $k \neq i$, party P_i additionally broadcasts a ZKAOK (using the (i,k)th 1-round ZKAOK system) proving that the sent message m_i^j was honestly generated using input x_i and randomness r''_i. In other words, P_i proves the following statement: there exist a value $\hat{\alpha_i} = (\hat{x_i}, \hat{G_i})$ such that: (a) the message m_i^j is consistent with the input $\hat{x_i}$ and the randomness $\hat{r_i} \oplus r'_i$ (i.e., $m_i^j = \Pi(T^j, \hat{x_i}, \hat{r_i} \oplus r'_i)$) where $\hat{r_i}$ is generated from $\hat{G_i}$, and, (b) commitment A_i can be decommitted to $\hat{\alpha_i}$. As before, the random tape used by P_i for generation and verification of the messages of ZKAOK system comes from R_i and $R_{i,zkv}$ respectively.

This completes the description of the protocol Σ. The usage of randomness in the above protocol can be summarized as follows. After sending the very first message (i.e., the determining message msg_i), the only fresh and uncommitted random tape that *can potentially* be used by a malicious party P_i is for the generation and verification of the 1-round ZKAOK messages (although the honest parties are instructed to use the committed random tape for the *generation* of 1-round ZKAOK messages).

Applying the above transformation to the constant round protocol of Beaver et al [BMR90], we obtain a constant round protocol Σ (secure against a minority of malicious parties). Our protocol is based on computational assumptions; the existence of NIZK (or equivalently, two round zaps [DN00]) to be precise. We note that it is easy to rule out information theoretically secure protocols in our setting very similar to how Barak et al [BGGL01] ruled out resettably-sound zero-knowledge *proofs*. The basic idea is that since an honest party has only a bounded size secret information (i.e., the input and the random tape), an unbounded dishonest party can interact with it several times (by resetting it each time) so as to "almost" learn its input/output behavior (and hence an honest party input "consistent" with that behavior). More details will be provided in the full version.

[2] As before, this assumption is only made for simplicity of exposition.

Let $\mathcal{M}$ be the list of malicious parties. Denote the list of honest parties $\mathcal{H} = \{P_1, \ldots, P_n\} - \mathcal{M}$. We defer the proof the following theorem to the full version of this paper for lack of space.

Theorem 3 (Security Against a Minority of Malicious Parties). *The compiled protocol Σ is secure as per definition 2.2 against the coalition of malicious parties represented by $\mathcal{M}$ as long as $|\mathcal{M}| < n/2$.*

References

[Bar01] Barak, B.: How to go beyond the black-box simulation barrier. In: FOCS, pp. 106–115 (2001)

[BGGL01] Barak, B., Goldreich, O., Goldwasser, S., Lindell, Y.: Resettably-sound zero-knowledge and its applications. In: FOCS, pp. 116–125 (2001)

[BMR90] Beaver, D., Micali, S., Rogaway, P.: The round complexity of secure protocols (extended abstract). In: STOC, pp. 503–513. ACM, New York (1990)

[Can00] Canetti, R.: Security and composition of multiparty cryptographic protocols. Journal of Cryptology: the journal of the International Association for Cryptologic Research 13(1), 143–202 (2000)

[CF01] Canetti, R., Fischlin, M.: Universally composable commitments. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 19–40. Springer, Heidelberg (2001)

[CGGM00] Canetti, R., Goldreich, O., Goldwasser, S., Micali, S.: Resettable zero-knowledge (extended abstract). In: STOC, pp. 235–244 (2000)

[CGS08] Chandran, N., Goyal, V., Sahai, A.: New constructions for UC secure computation using tamper-proof hardware. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 545–562. Springer, Heidelberg (2008)

[CKL06] Canetti, R., Kushilevitz, E., Lindell, Y.: On the limitations of universally composable two-party computation without set-up assumptions. J. Cryptology 19(2), 135–167 (2006)

[DL07] Deng, Y., Lin, D.: Instance-dependent verifiable random functions and their application to simultaneous resettability. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 148–168. Springer, Heidelberg (2007)

[DN00] Dwork, C., Naor, M.: Zaps and their applications. In: FOCS, pp. 283–293 (2000)

[DNS98] Dwork, C., Naor, M., Sahai, A.: Concurrent zero-knowledge. In: STOC, pp. 409–418 (1998)

[Dwo08] Dwork, C.: Differential privacy: A survey of results. In: Agrawal, M., Du, D.-Z., Duan, Z., Li, A. (eds.) TAMC 2008. LNCS, vol. 4978, pp. 1–19. Springer, Heidelberg (2008)

[GMW87] Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game. In: STOC 1987: Proceedings of the 19th annual ACM conference on Theory of computing, pp. 218–229. ACM Press, New York (1987)

[GS08] Goyal, V., Sahai, A.: Resolving the simultaneous resettability conjecture and a new non-black-box simulation strategy. Cryptology ePrint Archive, Report 2008/545 (2008), `http://eprint.iacr.org/`

[Kat07] Katz, J.: Universally composable multi-party computation using tamper-proof hardware. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 115–128. Springer, Heidelberg (2007)

[KP01] Kilian, J., Petrank, E.: Concurrent and resettable zero-knowledge in poly-loalgorithm rounds. In: STOC, pp. 560–569 (2001)

[Lin03] Lindell, Y.: Bounded-concurrent secure two-party computation without setup assumptions. In: STOC, pp. 683–692. ACM Press, New York (2003)

[Lin04] Lindell, Y.: Lower bounds for concurrent self composition. In: Naor, M. (ed.) TCC 2004. LNCS, vol. 2951, pp. 203–222. Springer, Heidelberg (2004)

[MP06] Micali, S., Pass, R.: Local zero knowledge. In: Kleinberg, J.M. (ed.) STOC, pp. 306–315. ACM, New York (2006)

[PRS02] Prabhakaran, M., Rosen, A., Sahai, A.: Concurrent zero knowledge with logarithmic round-complexity. In: FOCS, pp. 366–375 (2002)

[Sah99] Sahai, A.: Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In: FOCS, pp. 543–553 (1999)

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

[Yao86] Yao, A.C.-C.: How to generate and exchange secrets (extended abstract). In: FOCS, pp. 162–167. IEEE, Los Alamitos (1986)

[YZ07] Yung, M., Zhao, Y.: Generic and practical resettable zero-knowledge in the bare public-key model. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 129–147. Springer, Heidelberg (2007)