Multi-Protocol UC and its Use for Building Modular and Efficient Protocols

Jan Camenisch
DFINITY
jan@dfinity.org

Manu Drijvers
DFINITY
manu@dfinity.org

Björn Tackmann*
IBM Research – Zurich
bta@zurich.ibm.com

January 19, 2019

Abstract

We want to design and analyze protocols in a modular way by combining idealized components that we realize individually. While this is in principle possible using security frameworks that provide generic composition theorems, we notice that actually applying this methodology in practical protocols is far from trivial and, worse, is sometimes not even possible. As an example, we use a natural combination of zero-knowledge proofs with signature and commitment schemes, where the goal to have a party prove in zero-knowledge that it knows a signature on a committed message, i.e., prove knowledge of a witness to a statement involving algorithms of the signature and commitment scheme. We notice that, unfortunately, the composition theorem of the widely used UC framework does allow one to modularly prove the security of this example protocol.

We then describe a new variant of the UC framework, *multi-protocol UC*, and show a composition theorem that generalizes the one from the standard framework. We use this new framework to provide a modular analysis of a practical protocol that follows the above structure and is based on discrete-logarithm-based primitives. Besides the individual security proofs of the protocol components, we also describe a new methodology for idealizing them as components that can then be composed.

1 Introduction

1.1 Motivation

Complex cryptographic protocols such as anonymous credentials, e-cash, or voting are usually built by suitably combining simpler building block protocols and schemes. Designing and proving the security of such protocols is a challenging and difficult task. Initiated by the works of Pfitzmann and Waidner [PW01] and Canetti [Can01], a series of definitional frameworks has emerged that aim to modularize such constructions and, in particular, their security proofs via the use of general *composition theorems*. The core idea here is to first prove the security of all component schemes and protocols individually, and to analyze the security of the combined protocol relative to a simplified, idealized modeling of all components. The composition theorem implies the intended reduction statement and thereby establishes the security of the combined protocol. Soon after these models were published, a series of works started that modeled different types of cryptographic schemes and protocols such as commitment schemes [CF01], multi-party computation [CLOS02], digital signatures [Can04], and many more.

Today, *universally composable (UC) security* [Can01]) has been widely accepted as a desirable security goal as it allows for the use of a protocol as a building block in other protocols. In spite of this, the security proofs of complex protocols rarely invokes the composition theorem, even though they are typically built in a

*Björn has been supported by the European Union's Horizon 2020 research and innovation programme under grant agreement No. 780477 PRIViLEDGE.

modular way from multiple sub-protocols such as commitment, signature, and encryption schemes. Instead, their security is typically inferred via a reduction proof from the various security properties of the individual building blocks. Developing such a reduction proof for a complex protocol, however, is a manual process that is tedious and error-prone. Moreover, the resulting proofs are typically very long and complex and therefore hard and tedious to verify and thus often do not get sufficiently scrutinized.

The fact that authors do not make use of the modularity offered by the UC or other frameworks to simplify their security proofs seems paradoxical and surprising. A closer look, however, shows that this is, at least in many cases, due to the fact how the zero-knowledge proof functionality is modelled in UC, as it does not allow a party to prove to another party that it has correctly applied a cryptographic scheme to a hidden input. For the sake of concreteness, consider he following natural protocol which is used in variations in the construction of protocols such as for instance anonymous credential systems, voting, or e-cash. This prototypical protocols consist of a zero-knowledge proof protocol π_{ZK}, a signature scheme π_{SIG} with signature algorithm $\text{sign}(\cdot,\cdot)$ and verification algorithm $\text{ver}(\cdot,\cdot,\cdot)$, and a commitment scheme π_{COM} with commitment algorithm $\text{commit}(\cdot,\cdot)$ and opening algorithm $\text{open}(\cdot,\cdot,\cdot)$. (Both commit and open take as first input a common reference string crs, as commitment without setup is impossible in UC [CF01].) One party, the issuer I, owns a signature key pair (sk, pk) where the public key pk is known to all participants. I issues a "credential" m to a party P by creating a signature $s \leftarrow\!\!\$\ \text{sign}(sk, m)$, and producing a commitment $c \leftarrow \text{commit}(crs, m, o)$. Party P can then present its "credential" to a verifier V by proving, in zero knowledge, that it knows m, s, and o such that s is a valid signature on m relative to pk and c is a commitment on m with opening o – more formally described by the relation $\{((crs, c, pk), (m, s, o)) : \text{ver}(pk, m, s) = 1 \ \wedge\ \text{open}(crs, c, o) = m\}$. We will use this simple protocol as a running example throughout the paper.

Now assume that zero-knowledge protocol π_{ZK} realizes $\mathcal{F}_{\text{ZK}}$ (as specified in, e.g., [Can05]), signature scheme π_{SIG} realizes $\mathcal{F}_{\text{SIG}}$ (as specified in, e.g., [Can04]), and that commitment π_{COM} realizes $\mathcal{F}_{\text{COM}}$ (as specified in, e.g., [CF01]). We think that it would be desirable to prove the security of the overall protocol sketched above by idealizing all protocols π_{ZK}, π_{SIG}, and π_{COM} individually as $\mathcal{F}_{\text{ZK}}$, $\mathcal{F}_{\text{SIG}}$, and $\mathcal{F}_{\text{COM}}$ and then analyzing the security of the combined protocol by analyzing its security relative to these ideal functionalities. We also think that such a methodology would be in the true spirit of a composable security framework, at least more so than writing one monolithic "credential" functionality and proving the protocol via game hops and reductions from game-based security notions. We have noticed, however, that a proof with this methodology does not work in the UC framework of Canetti [Can18], and also not in its generalized version GUC of Canetti et al. [CDPW07]. We explain in the next section the obstacles that we hit in the course of our proof attempts, and gradually explain how they can be overcome. In a nutshell, we had to modify the functionalities realized by the sub-protocols, we had to prove a more general composition theorem, and we had to develop a new methodology for gradually idealizing the protocol components.

1.2 Towards modular proofs

The zero-knowledge functionality $\mathcal{F}_{\text{ZK}}$ described by Canetti [Can05] is parametrized by a relation R that specifies the language for which membership is to be proved, along with the witnesses. The prover inputs a pair (y, w) of statement y and witness w, and if $(y, w) \in R$, then $\mathcal{F}_{\text{ZK}}$ outputs to the verifier the element y. In our example protocol, this relation R will be $\{((crs, c, pk), (m, s, o)) : \text{ver}(pk, m, s) = 1 \wedge \text{open}(crs, c, o) = m\}$ as specified above; we will realize the functionality $\mathcal{F}_{\text{ZK}}^{R}$ for this relation R. The first obstacle is that, while the zero-knowledge functionality $\mathcal{F}_{\text{ZK}}^{R}$ depends on the algorithms $\text{ver}(\cdot,\cdot,\cdot)$ and $\text{open}(\cdot,\cdot,\cdot)$, both $\mathcal{F}_{\text{SIG}}$ and $\mathcal{F}_{\text{COM}}$ abstract away the algorithms used in the protocols π_{SIG} and π_{COM}. It is unclear how such a proof via $\mathcal{F}_{\text{ZK}}$ can be meaningfully related to $\mathcal{F}_{\text{SIG}}$ and $\mathcal{F}_{\text{COM}}$, in concluding the security of the overall protocol.

Towards resolving the above issue, assume that one abstracts a little less, say by parametrizing the functionalities $\mathcal{F}_{\text{SIG}}$ and $\mathcal{F}_{\text{COM}}$ with their algorithms, as $\mathcal{F}_{\text{SIG}}^{(\text{sign},\text{ver})}$ and $\mathcal{F}_{\text{COM}}^{(\text{commit},\text{open})}$, instead of having the algorithms be determined by the simulator. This, however, still does not suffice. In the $\mathcal{F}_{\text{SIG}}$-hybrid world, the validity of a signature s for a message m relative to public key pk is determined by asking $\mathcal{F}_{\text{SIG}}$ for verification. $\mathcal{F}_{\text{SIG}}$ determines this not (only) based on the output of $\text{ver}(pk, m, s)$, but instead uses a process that guarantees ideal unforgeability, whereas $\mathcal{F}_{\text{ZK}}$ with the above relation uses $\text{ver}(\cdot,\cdot,\cdot)$. Of course, the

results of $\mathrm{ver}(\cdot,\cdot,\cdot)$ and $\mathcal{F}_{\mathrm{SIG}}$ will deviate only with negligible probability as otherwise π_{SIG} would not even realize $\mathcal{F}_{\mathrm{SIG}}$, but this still means that the proof of the protocol composing $\mathcal{F}_{\mathrm{ZK}}$ and $\mathcal{F}_{\mathrm{SIG}}$ would need to bound the probability of the deviation and therefore reduce to the unforgeability of π_{SIG}, which is exactly what we wanted to prevent in a modular proof.

The above deviation can be prevented by having $\mathcal{F}_{\mathrm{ZK}}$, when verifying the relation $(y,w) \in R$, query the functionality $\mathcal{F}_{\mathrm{SIG}}$ instead of evaluating $\mathrm{ver}(\cdot,\cdot,\cdot)$ internally, and $\mathcal{F}_{\mathrm{COM}}$ instead of $\mathrm{open}(\cdot,\cdot,\cdot)$ analogously. To achieve this, one needs to modify how $\mathcal{F}_{\mathrm{ZK}}$ works, but that can be done; it is in fact one part of our solution and we explain it more in the next section. More formally, $\mathcal{F}_{\mathrm{ZK}}$ calls $\mathcal{F}_{\mathrm{SIG}}$ and $\mathcal{F}_{\mathrm{COM}}$ via their respective dummy protocols ϕ_{SIG} and ϕ_{COM}, so this method will then lead to a structure as depicted in Figure 1. $\mathcal{F}_{\mathrm{SIG}}$ and $\mathcal{F}_{\mathrm{COM}}$ become subroutines of $\mathcal{F}_{\mathrm{ZK}}$.

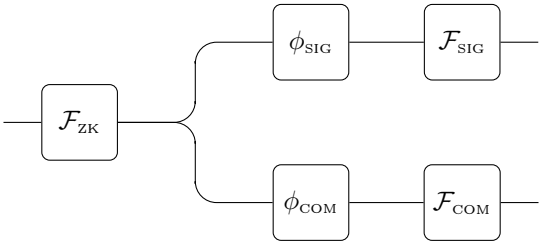

Figure 1: Functionalities $\mathcal{F}_{\mathrm{ZK}}$, $\mathcal{F}_{\mathrm{SIG}}$, and $\mathcal{F}_{\mathrm{COM}}$ interacting.

In such a setup, one also needs to change $\mathcal{F}_{\mathrm{COM}}$ because the standard formulation of [CF01] does not explicitly model non-interactive commitment and expose the $\mathrm{open}(\cdot,\cdot,\cdot)$ algorithm; we use a variant of the non-interactive commitment functionality $\mathcal{F}_{\mathrm{NCOM}}$ proposed by Camenisch et al. [CDR16] instead. But even that does not suffice. Since $\mathcal{F}_{\mathrm{SIG}}$ and $\mathcal{F}_{\mathrm{NCOM}}$ are queried from $\mathcal{F}_{\mathrm{ZK}}$, they become subroutines of $\mathcal{F}_{\mathrm{ZK}}$, which is formally invoked by the dummy protocol ϕ_{ZK}. When realizing $\mathcal{F}_{\mathrm{SIG}}$ by π_{SIG} and $\mathcal{F}_{\mathrm{NCOM}}$ by π_{NCOM}, then invoking the UC composition theorem would lead to the use of the protocol $\phi_{\mathrm{ZK}}^{\phi_{\mathrm{SIG}}\to\pi_{\mathrm{SIG}},\phi_{\mathrm{NCOM}}\to\pi_{\mathrm{NCOM}}}$. By virtue of the UC composition operation $\rho^{\phi\to\pi}$ as defined in [Can18], however, the invocations are only replaced within the protocol ϕ_{ZK}, not within its subroutine $\mathcal{F}_{\mathrm{ZK}}$. We resolve this by introducing a *recursive* composition operation $\rho^{\phi\twoheadrightarrow\pi}$ which replaces such invocations within all sub-protocols. This also requires re-proving the composition theorem.

While the above route, with many details filled in, would in principle work, it leads to a security statement that is much weaker than intended and effectively useless in applications. As $\mathcal{F}_{\mathrm{ZK}}$ now uses the functionalities $\mathcal{F}_{\mathrm{SIG}}$ and $\mathcal{F}_{\mathrm{COM}}$ to verify the relation, $\mathcal{F}_{\mathrm{SIG}}$ and $\mathcal{F}_{\mathrm{COM}}$ become subroutines of $\mathcal{F}_{\mathrm{ZK}}$. The UC composition theorem [Can18] (and also the GUC one [CDPW07], although in a different sense) require that a protocol be *subroutine respecting*, which essentially means that subroutines of $\mathcal{F}_{\mathrm{ZK}}$ (which includes $\mathcal{F}_{\mathrm{SIG}}$ and $\mathcal{F}_{\mathrm{NCOM}}$) must not communicate with any protocol that is not a sub-protocol of $\mathcal{F}_{\mathrm{ZK}}$. The only viable solution here is to have all outside protocols communicate with $\mathcal{F}_{\mathrm{SIG}}$ and $\mathcal{F}_{\mathrm{NCOM}}$ via $\mathcal{F}_{\mathrm{ZK}}$, but this results in one monolithic functionality that models all three protocols. While technically feasible, it is something that we wanted to prevent because it leads to encoding the complete complexity of the protocol in one monolithic functionality.

Our solution to this is to generalize the UC experiment such that it allows the environment to instantiate multiple different protocols. This has already been done in the GUC framework [CDPW07], but as already explained above this framework also does not resolve the problem since the composition theorem therein requires protocols to be subroutine respecting in a sense that is too narrow for our case. We define a new version of this definition, which we call *jointly subroutine respecting*, and prove a composition theorem that generalizes the one in [Can18].

1.3 Our proof methodology

As already mentioned above, our approach is to idealize the protocols π_{NIZK}, π_{SIG}, and π_{COM} individually. We already discussed some core ideas in the previous section, namely that we parametrize $\mathcal{F}_{\mathrm{NIZK}}$, $\mathcal{F}_{\mathrm{SIG}}$, and

$\mathcal{F}_{\text{NCOM}}$ by the concrete algorithms to make them compatible. We further explain in this section how we had to adapt several formal components to make this approach work.

Non-interactive zero-knowledge. We first use a non-interactive zero-knowledge protocol π_{NIZK} to instantiate a non-interactive zero-knowledge functionality $\mathcal{F}_{\text{NIZK}}$, which is adapted from the respective functionality of Groth et al. [GOS12] and the version of $\mathcal{F}_{\text{ZK}}$ of Canetti [Can05] and Camenisch at al [CKS11]. The functionality is, in particular, parametrized by a relation R that describes the statements and witnesses. This step follows as usual by the security of the non-interactive zero-knowledge protocol.

Re-writing the zero-knowledge functionality. The first step of our proof is simple and actually just a reformulation of the above functionality $\mathcal{F}_{\text{NIZK}}$. Instead of parametrizing $\mathcal{F}_{\text{NIZK}}$ by a relation R defined as $((crs, c, pk), (m, s, o)) : \text{ver}(pk, m, s) = 1 \ \wedge \ \text{open}(crs, c, o) = m$, we write $\mathcal{F}_{\text{NIZK}}$ as directly evaluating the protocols π_{SIG} and π_{COM} within the functionality in order to check whether the instance input by the prover is correct. As the commitment formalization in the work of Canetti and Fischlin [CF01] does not model non-interactive commitments and in particular does not allow to evaluate the algorithm $\text{open}(\cdot, \cdot, \cdot)$ via the protocol π_{COM}, we use a variant of the non-interactive commitment formalization $\mathcal{F}_{\text{NCOM}}$ of Camenisch et al. [CDR16].

Externalizing the algorithms. The second step of our proof is another reformulation of the zero-knowledge functionality $\mathcal{F}_{\text{NIZK}}$. Instead of emulating π_{SIG} and π_{NCOM} within $\mathcal{F}_{\text{NIZK}}$, the functionality creates Turing machine instances of π_{SIG} and π_{NCOM} that are external to $\mathcal{F}_{\text{NIZK}}$. It then verifies the relation using those external instances. This modification, which is depicted in Figure 2 on the left, works almost as expected, except for two subtleties: First, due to the UC intricacies, the adversary could have created Turing machine instances with the same identities but different code before. This will make the proof process in $\mathcal{F}_{\text{NIZK}}$ abort. Second, to provide the same meaningful guarantees as the previous functionality, the instances of π_{SIG} and π_{NCOM} must be uncorruptible, just like $\mathcal{F}_{\text{NIZK}}$ itself. We explain in Section 4 how these issues are resolved technically.

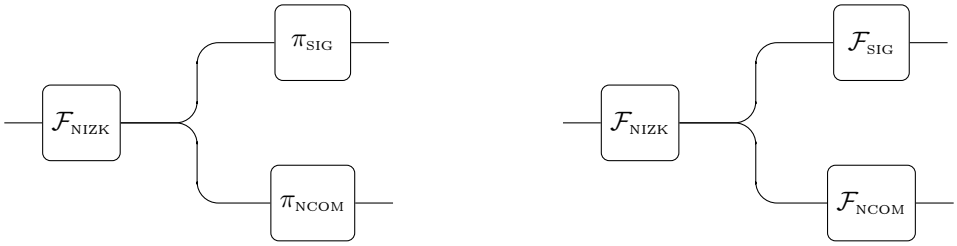

Figure 2: Functionalities $\mathcal{F}_{\text{NIZK}}$ using as subroutines π_{SIG} and π_{NCOM} (on the left) respectively $\mathcal{F}_{\text{SIG}}$ and $\mathcal{F}_{\text{NCOM}}$ interacting (on the right). In the right hand figure, we omit the dummy protocols ϕ_{SIG} and ϕ_{NCOM} that technically appear as forwarding messages between $\mathcal{F}_{\text{NIZK}}$ and the other functionalities.

Security of signatures and composition. We then finally use the security proofs of π_{SIG} and π_{NCOM} along with the (modified) composition theorem to actually obtain the situation in Figure 2 on the right, where $\mathcal{F}_{\text{NIZK}}$ uses $\mathcal{F}_{\text{SIG}}$ and $\mathcal{F}_{\text{NCOM}}$ as subroutines. This is the result on which higher-level protocols, such as credential schemes, can then be proved secure.

1.4 Related work

The initial papers of Pfitzmann and Waidner [PW01] and Canetti [Can01] have spawned a line of research work on composable security definitions. The original UC composition theorem (and later variants, such as

GNUC [HS15]) are restricted to protocols that do not share state with any other instance of any protocol. Canetti and Rabin [CR03] then showed how state can be shared between multiple sessions of a protocol; however, this requires that the protocols use the shared state in a predetermined and "nice" way. Roughly, the joint-state theorem proven there allows to convert a single instance of a scheme into multiple instances, such as via domain separation for an instance of a signature scheme. The composition theorem proved in the later GUC model of Canetti et al. [CDPW07] allows protocols to share state via one explicit global functionality, such as a PKI. None of these composition theorems apply to the type of modular protocol we analyze in this work, because the notions of subroutine respecting used there do not apply.

Other composable security frameworks with general composition theorems exist, which are based on different formal models. Notable examples include the Rective-Systems model of Backes, Pfitzmann, and Waidner [BPW04] and the IITM model of Küsters, Tuengerthal, and Rausch [KTR18], as well as the Abstract Cryptography framework of Maurer and Renner [MR11]. What all of these models have in common (also with [HS15]) is that the topology of the systems is (at least to some level) fixed in the security statement, whereas arbitrary topologies of machines can be adaptively generated in the models based on [Can01]. On the flip side, the composition theorems in those frameworks do not constrain (cf. [CDPW07]) the environment to only interact with one session of one challenge protocol, so these frameworks do not require an analogous version of the MUC theorem we prove in this work. The remaining contributions of this paper, however, also apply to those frameworks.

1.5 Our contributions

Our contributions in this work are three-fold.

Contributions to the UC model. The first contributions are specific to the UC-line of security models [Can01, CDPW07]. We point out that a modular protocol design in the sense described in the introduction is not possible in those models, as (1) the notions of subroutine respecting are too restrictive and (2) the composition operation does not allow to replace subroutines of an ideal functionality. We propose a new notion of *jointly subroutine respecting* that covers these protocols along with a new *recursive* composition operation and prove a new composition theorem that applies to these new definitions.

Modular proof methodology. We develop a methodology to modularly prove cryptographic protocols that involve zero-knowledge proofs for statements that involve other schemes used in the same protocol. The methodology is generic; we show its application to signatures and non-interactive commitment, but it can be used for other non-interactive schemes as well if their UC-protocol version allows to evaluate the algorithm directly via the input interface. (This may require a change of how the UC functionality for such schemes is written.) Beyond zero-knowledge proofs, we expect that the same methodology will be applicable to other cases where a scheme is evaluated within another protocol, which also appears in multi-party computation.

Efficient instantiation. We then provide a set of efficient protocols that realize our functionalities: $\mathcal{F}_{\text{SIG}}$ is realized via a signature scheme by Abe et al [AGOT14], $\mathcal{F}_{\text{NCOM}}$ is realized via a scheme that is essentially an ElGamal encryption with an opening via Groth-Sahai proofs [GS08] and $\mathcal{F}_{\text{NIZK}}$ is realized based on Schnorr proofs and in the global random-oracle model.

2 Preliminaries

This section describes formal material that we need for the remainder of the paper. We start in Section 2.1 by summarizing the basic workings and notation of the UC and GUC models from [Can01, CDPW07, Can18]. We then describe ideal functionalities for non-interactive zero knowledge in Section 2.3, for digital signatures in Section 2.4, and for non-interactive commitment in Section 2.5. All of these functionalities are slight adaptations of the versions found in the literature.

2.1 UC framework basics

Simulation-based security and protocol composition. Most general security frameworks are based on the real-world/ideal-world paradigm: In the real world, the parties execute the protocol using channels as defined by the model. In the ideal world, the parties securely access an ideal functionality $\mathcal{F}$ that obtains inputs from the parties, runs the program that specifies the task to be achieved by the protocol, and returns the resulting outputs to the parties. Intuitively, a protocol *securely realizes* the functionality $\mathcal{F}$ if, for any real-world adversary $\mathcal{A}$ attacking the protocol execution, there is an ideal-world adversary $\mathcal{S}$, also called the *simulator*, that emulates $\mathcal{A}$'s attack. The simulation is good if no distinguisher $\mathcal{Z}$—often called the *environment*—which interacts, in a well defined manner, with the parties and the adversary/simulator, can distinguish between the two worlds.

The advantage of such security definitions is that they satisfy strong composability properties. Let π_1 be a protocol that securely realizes a functionality $\mathcal{F}_1$. If a protocol π_2, using the functionality $\mathcal{F}_1$ as a subroutine, securely realizes a functionality $\mathcal{F}_2$, then the protocol $\pi_2^{\mathcal{F}_1 \to \pi_1}$, where the calls to $\mathcal{F}_1$ are replaced by invocations of π_1, securely realizes $\mathcal{F}_2$ (without calls to $\mathcal{F}_1$). Therefore, it suffices to analyze the security of the simpler protocol π_2 in the $\mathcal{F}_1$-*hybrid* model, where the parties run π_2 with access to the ideal functionality $\mathcal{F}_1$. A detailed treatment of protocol composition appears in, e.g., [BPW04, Can00, Can01, DM00, MR11].

The UC framework. We use the formal model of the UC framework [Can01, Can18]. The definitions are based on the simulation paradigm, and the entities taking part in the execution (protocol machines, functionalities, adversary, and environment) are described as *interactive Turing machines* (ITMs). The execution is an interaction of *ITM instances* (ITIs) and is initiated by the environment that provides input to and obtains output from the protocol machines, and also communicates with the adversary. The adversary has access to the ideal functionalities in the hybrid models, in some models it also serves as a network among the protocol machines. During the execution, the ITIs are activated one-by-one, where the exact order of the activations depends on the considered model.

Each ITI has an identity that consists of a party identifier *pid* and a session identifier *sid*. (The environment and adversary have specific, constant identifiers.) The understanding here is that all ITIs that share the same code and the same *sid* are considered a *session* of a protocol. It is natural but not required to use the same *pid* for all ITIs that are considered as the same party.

ITIs can be created adaptively during the protocol execution, with their code determined during the invocation. In order to use protocol composition, some additional restrictions are necessary. In a protocol composition $\rho^{\phi \to \pi}$, both the ideal sub-protocol ϕ and the real sub-protocol π must be *subroutine respecting*. This means, in a nutshell, that those protocols may have further subroutines, but inputs and outputs from subroutines of ϕ or π must only be given and obtained through via ϕ or π, never by directly interacting with their subroutines. (This requirement is natural, since a higher-level protocol should never directly access the internal structure of ϕ or π; this would obviously hurt composition.) Also, protocol ρ must be *compliant*. This roughly means that ρ should not be invoking instances of π with the same *sid* as instances of ϕ, as otherwise these instances of π would interact with the ones obtained by the operation $\rho^{\phi \to \pi}$. (This is again natural, as ρ should generally not invoke different subroutines with the same *sid*.)

Notation and conventions. A protocol execution involves the following types of ITIs: the environment $\mathcal{Z}$, the adversary $\mathcal{A}$, instances of the protocol machines π, and (possibly) further ITIs invoked by $\mathcal{A}$ or any instance of π (or their subroutines). The contents of the environment's output tape after the execution is denoted by the random variable $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k, z)$, where $k \in \mathbb{N}$ is the *security parameter* and $z \in \{0,1\}^*$ is the input to the environment $\mathcal{Z}$. The formal details of the execution are specified in [Can18]. We say that a protocol π UC-realizes a functionality $\mathcal{F}$ if

$$\forall \mathcal{A}\ \exists \mathcal{S}\ \forall \mathcal{Z} : \text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}} \approx \text{EXEC}_{\phi,\mathcal{S},\mathcal{Z}},$$

where "$\approx$" denotes indistinguishability of the respective distribution ensembles, and ϕ is the dummy protocol that simply relays all inputs to and outputs from the functionality $\mathcal{F}$. We also use global functionalities akin

to the EUC model in [CDPW07]. There, one additionally allows all protocols to access a global functionality $\bar{\mathcal{G}}$ that is available both in the real and in the ideal model, and that models global setup that is available to multiple protocols. The EUC execution is then described by a random variable GEXEC that is additionally parametrized by the global functionality $\bar{\mathcal{G}}$, and a protocol π realizes a functionality $\mathcal{F}$ with respect to $\bar{\mathcal{G}}$ if

$$\forall \mathcal{A}\ \exists \mathcal{S}\ \forall \mathcal{Z} : \text{GEXEC}^{\bar{\mathcal{G}}}_{\pi,\mathcal{A},\mathcal{Z}} \approx \text{GEXEC}^{\bar{\mathcal{G}}}_{\phi,\mathcal{S},\mathcal{Z}},$$

where again ϕ is the dummy protocol for $\mathcal{F}$. The "G" in GEXEC references the Global nature of $\bar{\mathcal{G}}$.

In the UC framework [Can01], the potential dishonesty of parties is modeled by the fact that the adversary $\mathcal{A}$ may corrupt protocol machines (adaptively) during the execution by sending them a special `corrupt` message. We only consider Byzantine party-corruption, which means that the party sends its entire current local state to $\mathcal{A}$ and, in all future activations, follows $\mathcal{A}$'s instructions. We do allow our protocols to securely delete state, which means the adversary does not inherently learn the complete execution history of a party upon corruption. In the formalism of [Can18], this means we are using *standard f-revealing corruption* for a function f that leaks only the current state of the protocol, not the complete history.[1] All our functionalities $\mathcal{F}$ use a corruption that is similar to *standard (adaptive) PID-wise corruption* as defined in [Can18], but adapted to the setting where parties can remove state: At any point in time, $\mathcal{F}$ will accept messages of the type $(\texttt{corrupt}, pid)$ for some party identifier *pid* from the adversary $\mathcal{A}$, in which case $\mathcal{F}$ marks the party with identifier *pid* as corrupted. In contrast to [Can18] we do not enforce that all functionalities leak the complete history of inputs and outputs of party *pid* upon this operation. The exact information that is leaked and influence that the adversary has on a functionality depends on the concrete case, but generally the adversary will learn all information and control all actions in $\mathcal{F}$ that correspond to party *pid* after corruption. Also, all functionalities allow a specific `report` upon which they respond with the list of corrupted *pid*s. For more details on party corruption, see [Can18, Section 6.1].

2.2 Strict global random oracle

The schemes we use for instantiating the NIZK and the non-interactive commitment scheme are proved secure in the random oracle model. We model the random oracle as a global shared functionality that can be used by any protocol, in more detail we use the strict global random oracle from Camenisch et al. [CDG+18]. The functionality is specified in detail in Figure 3.

Global Functionality $\bar{\mathcal{G}}_{\text{sRO}}$

Functionality $\bar{\mathcal{G}}_{\text{sRO}}$ is parametrized by the output size $\ell(\lambda)$ and keeps internally an initially empty list $\mathsf{List}_{\mathcal{H}}$ of queries and responses.

- Upon input $(\texttt{HashQuery}, m)$ from a party (P, sid), proceed as follows.
  - Find h such that $(m, h) \in \mathsf{List}_{\mathcal{H}}$. If no such h exists, let $h \leftarrow\!\!\$\ \{0,1\}^{\ell(\lambda)}$ and store (m, h) in $\mathsf{List}_{\mathcal{H}}$.
  - Output $(\texttt{HashConfirm}, h)$ to (P, sid).

Figure 3: The strict global random oracle functionality $\bar{\mathcal{G}}_{\text{sRO}}$ that does not give any extra power to any party.

In a nutshell, functionality $\bar{\mathcal{G}}_{\text{sRO}}$ provides access to the same random oracle to all parties, including the adversary. This means, in particular, that $\bar{\mathcal{G}}_{\text{sRO}}$ does not provide any additional power to the adversary, such as programming outputs for queries made in the protocols.

[1] The instantiations we provide do not achieve security if all prior states are leaked.

2.3 Non-interactive zero-knowledge functionality

The non-interactive zero-knowledge functionality $\mathcal{F}_{\text{NIZK}}$ depicted in Figure 4 is based on the functionality described by Groth et al. [GOS12], but the style of our description is more in line with the interactive zero-knowledge functionalities of Canetti [Can05] and Camenisch et al. [CKS11]. $\mathcal{F}_{\text{NIZK}}$ is parametrized by a relation $R \subseteq \{0,1\}^* \times \{0,1\}^*$ where $(y, w) \in R$ has the meaning that w is a witness for y being in a certain language. We implicitly assume that the relation R is efficiently checkable.

Functionality $\mathcal{F}_{\text{NIZK}}^{R}$

$\mathcal{F}_{\text{NIZK}}$ is parametrized by a relation R for which we can efficiently check membership. It keeps an initially empty list L of proven statements.

1. On input $(\texttt{prove}, y, w)$ from a party P, such that $(y, w) \in R$,[a] send $(\texttt{prove}, y)$ to $\mathcal{A}$.
2. Upon receiving a message $(\texttt{done}, \psi)$ from $\mathcal{A}$, with $\psi \in \{0,1\}^*$, record (y, ψ) in L and send $(\texttt{done}, \psi)$ to P.
3. Upon receiving $(\texttt{verify}, y, \psi)$ from some party P', check whether $(y, \psi) \in L$. If not, output $(\texttt{verify}, y, \psi)$ to $\mathcal{A}$ and upon receiving answer $\texttt{witness}, w$. Check $(y, w) \in R$ and if so, store (y, ψ) in L. If (y, ψ) has been stored, then output 1 to P', else output 0.

[a] Inputs that do not satisfy the respective relation are ignored.

Figure 4: Non-interactive zero-knowledge functionality based on the one described by Groth et al. [GOS12]

Functionality $\mathcal{F}_{\text{NIZK}}$ manages a list L of statements that have been proved, this is to guarantee consistent responses is repeated verification queries. The functionality accepts an input $(\texttt{prove}, y, w)$ from prover P with statement y and witness w. If $(y, w) \in R$ then the functionality sends $(\texttt{prove}, y)$ to $\mathcal{A}$, otherwise it stops and outputs nothing. As a next step, functionality $\mathcal{F}_{\text{NIZK}}$ then expects an input $(\texttt{done}, \psi)$ from $\mathcal{A}$, where $\psi \in \{0,1\}^*$ is the non-interactive proof for statement y, stores (y, ψ) in the list L and outputs the proof ψ to prover P. Any party can call $(\texttt{verify}, y, \psi)$ with statement y and proof ψ. If $(y, \psi) \in L$, then immediately output 1 to the verifier. Otherwise, output $(\texttt{verify}, y, \psi)$ to $\mathcal{A}$ in order to allow for an input of a valid witness w. If $\mathcal{A}$ indeed allows with a valid witness, then store (y, ψ) in L and output 1, otherwise (i.e., if there is no valid witness) the proof is considered incorrect and $\mathcal{F}_{\text{NIZK}}$ outputs 0 to the verifier.

As both proof generation and proof verification are local computations by the prover and the verifier in a NIZK scheme, respectively, but use interaction with the adversary in functionality $\mathcal{F}_{\text{NIZK}}$, the processes in the ideal $\mathcal{F}_{\text{NIZK}}$ are interruptible whereas those in any real protocol are not. One can use responsive environments as specified by Camenisch et al. [CEK+16] to overcome this difference and obtain an ideal model without this artificial weakness.

2.4 Signature scheme and functionality

A *signature scheme* is a triplet of algorithms $(\text{kgen}, \text{sign}, \text{ver})$. Key generation $(sk, pk) \leftarrow\!\!\$\ \text{kgen}(\lambda)$ takes as input the security parameter λ and outputs a pair or private (or secret) key sk and public key pk. Signing algorithm $s \leftarrow\!\!\$\ \text{sign}(sk, m)$ takes as input private key sk and message m, and produces a signature s. Verification algorithm $b \leftarrow \text{ver}(pk, m, s)$ takes as input public key pk, message m, and signature s, and outputs a bit b that signifies whether s is a valid signature on m relative to public key pk. Algorithms kgen and sign may be probabilistic, whereas ver is deterministic. The standard definition of signature scheme security, existential unforgeability under chosen-message attack, has been introduced by Goldwasser et al. [GMR88] and states that the probability for an efficient adversary, given an oracle for producing valid signatures, to output a valid signature on a message that has not been queried to the oracle must be negligible.

We specify the signature functionality $\mathcal{F}_{\text{SIG}}$ that we use in this work in Figure 5. Several different formalizations of signature scheme security in composable frameworks have been described in the literature [PW01, Can04, KT13, BMT18]. Our functionality follows most closely the version of Küsters and Tuengerthal [KT13] but is parametrized by fixed signature algorithms instead of allowing them to be determined adaptively. (Our functionality also runs the key generation inside the functionality, and that algorithms and keys are output, instead of input, at the adversary interface.) We furthermore change the functionality in the following way: Instead of allowing all parties access to the signing operation, we allow this only to one specified signer S. This is more in line with the functionality described by Canetti [Can04] and is required for properly handling corruptions in our setting, where we need to deal with corrupted receivers. (The functionality of [KT13] provides no guarantees in that case.)

Functionality $\mathcal{F}_{\text{SIG}}^{(\text{kgen},\text{sign},\text{ver})}$

Functionality $\mathcal{F}_{\text{SIG}}$ has sender S specified in *sid*. Set $\mathcal{C}$, initially empty, specifies the set of currently corrupted parties. The functionality keeps a set $\mathcal{L}$ of properly signed messages.

0. Upon the first activation from S, run $(sk, pk) \leftarrow\!\!\$\ \text{kgen}(\lambda)$, where λ is obtained from the security parameter tape, and store (sk, pk).
1. Upon input `pubkey` from party S, output $(\texttt{pubkey}, pk)$ to S.
2. Upon input $(\texttt{sign}, m)$ from party S with $m \in \{0,1\}^*$, compute $s \leftarrow\!\!\$\ \text{sign}(sk, m)$. Set $\mathcal{L} \leftarrow \mathcal{L} \cup \{m\}$ and output s to S.
3. Upon input $(\texttt{verify}, pk', m', s')$ from party P, compute $b \leftarrow \text{ver}(pk', m', s')$. If $S \notin \mathcal{C} \wedge pk = pk' \wedge b \wedge m' \notin \mathcal{L}$ then output $(\texttt{result}, 0)$ to P. Else output $(\texttt{result}, b)$ to P.
4. Upon input $(\texttt{corrupt}, P)$ from the adversary, set $\mathcal{C} \leftarrow \mathcal{C} \cup \{P\}$. If $P = S$, then additionally output sk to $\mathcal{A}$.

Figure 5: Signature functionality

In more details, $\mathcal{F}_{\text{SIG}}$ allows for one specific sender S which is specified in its session identifier *sid*. Upon first activation from S, functionality $\mathcal{F}_{\text{SIG}}$ runs kgen to generate a new key pair (sk, pk) for S. Functionality $\mathcal{F}_{\text{SIG}}$ then allows sender S to obtain its own public key via the `pubkey` operation and to sign a message m via the $(\texttt{sign}, m)$ operation. Verifiers can invoke $(\texttt{verify}, pk', m', s')$ in order to verify signature s' for message m' relative to public key pk'. The functionality guarantees that, as long as the sender is honest, signatures can never be forged: If pk' is the public key pk generated in the functionality and message m' has never be signed before, then s' is deemed invalid, independent of the algorithm ver. In any other case, meaning for corrupted S, or for a different pk', the actual verification algorithm is used to determine the result. Upon corruption of any party, adversary $\mathcal{A}$ from then on controls the interface of that party. If the sender is corrupted, then $\mathcal{A}$ also learns the secret key sk.

Generating the key pair (pk, sk) inside functionality $\mathcal{F}_{\text{SIG}}$ may first seem to provide unnaturally strong guarantees: corrupted parties cannot be forced to actually generate their keys with algorithm kgen. Yet, functionality $\mathcal{F}_{\text{SIG}}$ does not require a corrupt sender to ever use $\mathcal{F}_{\text{SIG}}$: As `verify` can be called with any public key pk', the corrupted party can generate this key pk' and produce signatures s' in arbitrary ways. The only guarantee that $\mathcal{F}_{\text{SIG}}$ provides in this setting is that honest verifiers actually use the correct algorithm for verification.

2.5 Non-interactive commitment scheme and functionality

A *non-interactive commitment scheme in the CRS model* specifies three algorithms, crsgen, commit, and open. CRS generator $crs \leftarrow\!\!\$\ \mathrm{crsgen}(\lambda)$ gets as input the security parameter λ and produces a common reference string crs. Commitment algorithm $(c, o) \leftarrow\!\!\$\ \mathrm{commit}(crs, x)$ gets as input the CRS crs and a message x. It produces as output the commitment c and opening information o. Opening algorithm $b \leftarrow \mathrm{open}(crs, c, x, o)$ gets as input the CRS crs, commitment c, message x, and opening information o, and outputs a bit b. Value $b = 1$ signifies that message x is contained in c, given that crs and o are correct. Value $b = 0$ signifies that commitment c does not open to x relative to crs and o. Algorithms crsgen and commit are probabilistic, whereas open is deterministic.

To instantiate our functionality $\mathcal{F}_{\text{NCOM}}$, we use extractable trapdoor commitments. That is, we additionally require trapdoor algorithms tdcom and tdopen and extraction algorithm extract, where trapdoor commitment algorithm $(\tilde{c}, \mathit{info}) \leftarrow\!\!\$\ \mathrm{tdcom}(crs, td)$ takes as input the CRS crs and a trapdoor td and outputs a commitment c and additional information info. Trapdoor opening algorithm $\tilde{o} \leftarrow\!\!\$\ \mathrm{tdopen}(x, \mathit{info})$ takes as input a message x and the value info, and produces an opening information $\tilde{o}$ that allows to open the commitment c to the value x. Extraction algorithm $x' \leftarrow \mathrm{extract}(par, td, c)$ that the value x' contained in commitment c, using the trapdoor td.

The non-interactive commitment functionality $\mathcal{F}_{\text{NCOM}}$ depicted in Figure 6 follows the version of [CDR16], but is simplified for a single session. Furthermore, the functionality is parametrized by the algorithms, including the trapdoor ones. The functionality is slightly changed from the version in [CDR16], the reason is that we want/need to keep the interfaces more along the lines of the actual algorithms, since we want to use them in the ZK proof. In particular, the functionality allows to call `commit` and `open` even with incorrect values for the CRS, it simply does not provide any guarantees in that case. This will be instrumental in our modular proof.

In more detail, functionality $\mathcal{F}_{\text{NCOM}}$ is initialized through a message $(\texttt{setup}, par, td)$ by $\mathcal{A}$. This message allows the adversary to specify an arbitrary string as the CRS used by the algorithms. Parties can validate the CRS by means of calling $(\texttt{validate}, par')$ in the functionality, which responds whether $par \stackrel{?}{=} par'$. Parties can generate a commitment on a message x by calling $(\texttt{commit}, par', x)$. For honest parties that use the correct CRS, the commitment c and the opening open are generated via the trapdoor algorithms tdcom and tdopen, and stored in a table within functionality. This guarantees hiding, since c is generated independently of the message x. For dishonest parties or parties that use an incorrect CRS, the adversary is asked to provide a commitment and opening.

The opening via $(\texttt{open}, par', c, x, \mathit{open})$ behaves analogously. If called by an honest party and with a correct CRS, functionality $\mathcal{F}_{\text{NCOM}}$ checks whether c was generated in the functionality. In that case, the commitment provides ideal guarantees, meaning that the result is 1 if and only if all parameters are the same as during commitment generation. If commitment c was not generated in the functionality, the extract algorithm is used to obtain the message contained in the commitment. Finally, if the party calling `open` is corrupt or the CRS is incorrect, then $\mathcal{A}$ is called to determine the output to that party.

The use of extraction may at first seem unnecessary, since we already obtained the purported value x in the `open` call and would simply need to verify the correctness of the opening. This, however, does not provide the expected semantics in terms of *binding*: The functionality in this case guarantees that a corrupt party only ever open a commitment to one particular value x', it does not, however, guarantee that this value is already determined at the time of commitment (which for dishonest parties may happen outside of the functionality). Such a formulation would not prevent a dishonest party from generating two pairs (x, o) and (x', o') of value and opening, and then at the time of `open` present one of the two to the honest party. Camenisch et al. [CDR16] prove that any protocol π_{NCOM} that realizes such a weaker form of $\mathcal{F}_{\text{NCOM}}$ still provides *binding* in the expected sense; however, this is a property of π_{NCOM}, not $\mathcal{F}_{\text{NCOM}}$. We therefore decide to use a stronger definition of $\mathcal{F}_{\text{NCOM}}$ based on extraction, where the fact that extract deterministically outputs the *valid* value x guarantees that every commitment c can only contain one particular value.

Functionality $\mathcal{F}_{\text{NCOM}}^{(\mathcal{M},\text{open},\text{tdcom},\text{tdopen},\text{extract})}$ – $\bar{\mathcal{G}}_{\text{sRO}}$ hybrid

Set $\mathcal{C}$, initially empty, specifies the set of currently corrupted parties. Keep a table of commitment information, containing triplets of commitment, value, and opening information. Allow algorithms $\text{open}, \text{tdcom}, \text{tdopen}, \text{extract}$ to query $\bar{\mathcal{G}}_{\text{sRO}}$, and require extract to be deterministic.

1. Upon $(\texttt{setup}, par, td)$ from $\mathcal{A}$, store par and td, output $\texttt{ok}$ to $\mathcal{A}$.
2. Upon $(\texttt{validate}, par')$ from P, check $par \stackrel{?}{=} par'$ and report the result to P.
3. Upon $(\texttt{commit}, par', x)$ with $x \in \mathcal{M}$ from P, proceed as follows.
   (a) If $par' \neq par$ or $P \in \mathcal{C}$, then output $(\texttt{commit}, P, par', x)$ to the adversary. Receive a response $(\tilde{c}, \tilde{x}, \widetilde{open})$ from the adversary, record it in the commitment table, and output $(\tilde{c}, \widetilde{open})$ to P.
   (b) Else, in case $par' = par$, compute $(c, info) \leftarrow\!\!\$\ \text{tdcom}(par, td)$, abort if c has been recorded for a message $x' \neq x$, compute $open \leftarrow\!\!\$\ \text{tdopen}(x, info)$, abort if $\text{open}(par, c, x, open) \neq 1$, record the commitment $(c, x, open)$ in the table, and return $(c, open)$ to P.
4. Upon $(\texttt{open}, par', c, x, open)$ from some P:
   (a) If $par' \neq par$ or $P \in \mathcal{C}$, then output $(\texttt{open}, P, par', c, x, open)$ to the adversary. Obtain a response b and set $f \leftarrow b$.
   (b) Else, meaning that $par' = par$, set $f \leftarrow \text{open}(par, c, x, open) = 1$.
      i. If c was generated inside this functionality, look up record $(c, \tilde{x}, \tilde{open})$ for some $\tilde{x}, \tilde{open}$. If $x \neq \tilde{x}$, set $f \leftarrow 0$.
      ii. If c was not generated inside this functionality and $x \neq \text{extract}(par, td, c)$ set $f \leftarrow 0$.

   Output f to P.

Figure 6: Non-interactive commitment functionality

3 Multi-protocol UC

In this section, we describe the modifications that must be done to the UC framework in order to provide modular proofs in the spirit described in the introduction. The standard UC experiment allows the environment to create only a single session of a single protocol; it is shown that this setting is sufficient if the challenge protocol is subroutine respecting, in the sense that it implies security also in a setting where the environment can invoke multiple sessions of arbitrary protocols in addition to the challenge protocol. The notion of subroutine respecting also enforces a tree structure on protocols with modular proofs, where different sub-protocols are fully isolated, which is more explicit in the GNUC model [HS15]. Such a tree structure, however, does not capture the situation we're interested in, with multiple functionalities that interact with each other. The same restriction as in UC also applies to the EUC experiment that is the technical core of the composition theorem in the GUC paper [CDPW07], where in addition the environment can access a global functionality. We first introduce in Section 3.1 a new recursive UC operation, before we describe in Section 3.2 the Multi-protocol UC (MUC) experiment, a new definition of subroutine respecting, and our new composition theorem with proof.

3.1 Recursive universal composition

Universal composition of protocols is defined via an operation $\rho^{\phi \to \pi}$ that replaces, in the protocol ρ, all external-write instructions targeted at ITIs that run code ϕ by ITIs that run code π [Can18]. The composition theorem then shows that for a compliant protocol ρ and subroutine respecting protocols ϕ and π such that π UC-emulates ϕ, protocol $\rho^{\phi \to \pi}$ UC-emulates protocol ρ. This composition operation works in a setting where the subroutine-respecting protocols π and ϕ will receive input from and provide subroutine output only to ρ, not from or to any other subroutine of ρ.[2] Therefore, all invocations to the protocol session in question occur in ρ.

In the setting we consider in this paper, the same restriction is not true. We want to explicitly allow for sub-protocols that are used in different levels of the hierarchy: the zero-knowledge functionality $\mathcal{F}_{\text{ZK}}$ uses $\mathcal{F}_{\text{SIG}}$ and $\mathcal{F}_{\text{NCOM}}$ as subroutines, but also other, higher-level protocols (which may also invoke $\mathcal{F}_{\text{ZK}}$) need to invoke $\mathcal{F}_{\text{SIG}}$ to, in this case, create the necessary signatures and commitments. Therefore, we define a *recursive* composition operation that applies to a protocol together with all of its sub-protocols.

Recursive universal composition operation. We define the new composition operator $\text{UC}^*()$ as follows. Like the operator $\text{UC}()$ defined by Canetti [Can18], given a protocol ϕ, a protocol ρ, and a protocol π (that presumably $\text{UC}^*()$ emulates ϕ), the composed protocol $\rho^{\phi \to\!\!\!\!\to \pi} = \text{UC}^*(\rho, \pi, \phi)$ that is identical to protocol ρ, with the following modifications:

1. Wherever ρ contains an instruction to pass input x to an ITI running ϕ with identity (sid, pid), then $\rho^{\phi \to\!\!\!\!\to \pi}$ contains instead an instruction to pass input x to an ITI running π with identity (sid, pid).

2. Whenever $\rho^{\phi \to\!\!\!\!\to \pi}$ receives an output passed from $\pi_{(sid,pid')}$ (i.e., from an ITI running π with identity (sid, pid')), it proceeds as ρ proceeds when it receives an output passed from $\phi_{(sid,pid')}$.

3. Whenever ρ contains an instruction to pass input x to an ITI running $\psi \neq \phi$ with identity (sid, pid), then $\rho^{\phi \to\!\!\!\!\to \pi}$ contains instead an instruction to pass input x to an ITI running $\psi^{\phi \to\!\!\!\!\to \pi} = \text{UC}^*(\psi, \phi, \pi)$ with identity (sid, pid).

4. Whenever $\rho^{\phi \to\!\!\!\!\to \pi}$ receives an output passed from $\psi^{\phi \to\!\!\!\!\to \pi}_{(sid,pid')}$ (i.e., from an ITI running $\psi^{\phi \to\!\!\!\!\to \pi}$ with identity (sid, pid')), it proceeds as ρ proceeds when it receives an output passed from $\psi_{(sid,pid')}$.

Overall, the difference with the composition operation from the original framework [Can18] is that protocol replacement also applies to sub-protocols. We remark that a bit more care is needed with this operator: UC composition theorems do not work for protocols ρ that create sub-protocols of both ϕ and π with the same session identifier—this is simply not captured by the main UC experiment.[3] The latest release of the UC framework resolves this issue by explicitly forbidding such a situation for the protocol ρ by the notion of *compliant* [Can18]. The situation is a bit more subtle here, where a similar property has to hold for a set of protocols with all sub-protocols.

3.2 The multi-protocol UC experiment and composition theorem

The standard UC experiment allows the environment to create a single session of a single protocol, which is usually called π in the real experiment and ϕ in the ideal experiment. In more detail, the control function specifies that the first ITI created by the environment be the adversary, and every subsequently created ITI have the code of either π or ϕ, depending on the considered experiment. The plain UC experiment therefore does not model a situation where multiple protocols are executed together and call each other in different ways, and where the environment (or higher-level protocols) can use multiple ones simultaneously. The GUC model in [CDPW07] allows this generality; however, GUC security is hard to prove for general protocols and

[2] The proof of the UC composition theorem in [Can18] also applies more generally, but the composition operation is defined in the way described here.

[3] The formal model in [CDPW07] does not exclude this case, which then leaves the composition theorem flawed.

therefore the authors in [CDPW07] resort to the notion of EUC, which again has a single challenge protocol. We therefore develop a new *multi-protocol UC (MUC)* experiment and definition that has the necessary generality.

In a nutshell, this new experiment can be understood as a UC experiment which (1) specifies a vector of challenge protocols and where (2) the environment is allowed to instantiate arbitrarily many sessions of each of the challenge protocols, which (3) can also access a global functionality $\bar{\mathcal{G}}$ as in GUC. How $\mathcal{Z}$ addresses the individual challenge protocols actually does not matter, but for the sake of concreteness we assume that $\mathcal{Z}$ uses the code field of the external-write instruction (which is otherwise ignored anyway) to specify this information. The remaining description of the UC experiment carries over from [Can18] and the EUC experiment in [CDPW07], respectively, and we write the random variable that describes the environment's output in the MUC experiment with environment $\mathcal{Z}$, adversary $\mathcal{A}$, protocol vector $(\pi_1, \ldots, \pi_n)$, and shared global functionality $\bar{\mathcal{G}}$ as $\text{MEXEC}^{\bar{\mathcal{G}}}_{(\pi_1,\ldots,\pi_n),\mathcal{A},\mathcal{Z}}$. We simply write $\text{MEXEC}_{(\pi_1,\ldots,\pi_n),\mathcal{A},\mathcal{Z}}$ if the protocols do not access a global shared functionality. Also in accordance with standard UC and EUC, the notion of MUC emulation is then defined as follows.

Definition 1. The vector $\vec{\pi} = (\pi_1, \ldots, \pi_n)$ of protocols *MUC-emulates* the vector $\vec{\phi} = (\phi_1, \ldots, \phi_n)$ of protocols *with respect to a global functionality* $\bar{\mathcal{G}}$ if there exists a simulator $\mathcal{S}$ such that

$$\text{MEXEC}^{\bar{\mathcal{G}}}_{\vec{\pi},\mathcal{A},\mathcal{Z}} \approx \text{MEXEC}^{\bar{\mathcal{G}}}_{\vec{\phi},\mathcal{S},\mathcal{Z}},$$

for all environments $\mathcal{Z}$. We simply say that the vector $\vec{\pi}$ *MUC-emulates* the vector $\vec{\phi}$ if they do not use a global functionality (i.e., if the protocols are subroutine respecting in the sense of [Can18]), and $\text{MEXEC}_{\vec{\pi},\mathcal{A},\mathcal{Z}} \approx \text{MEXEC}_{\vec{\phi},\mathcal{S},\mathcal{Z}}$ for all $\mathcal{Z}$.

The following lemma formalizes the intuition that a protocol that is subroutine respecting in the sense of [Can18] and UC-emulates another subroutine-respecting protocol ϕ if and only if it MUC-emulates ϕ. As the MUC experiment provides the environment with strictly more flexibility, it is obvious that MUC-emulation implies UC-emulation. On the flip side, the only additional restriction of the UC-experiment is that it only allows a single session of the protocol, but this difference is merely quantitative. The analogous statement holds for EUC security.

Lemma 2. *Let π, ϕ be protocols that are both subroutine respecting in the sense of [Can18]. Then π UC-emulates ϕ if and only if π MUC-emulates ϕ. Let $\bar{\mathcal{G}}$ be a global functionality and π_2, ϕ_2 be protocols that are both $\bar{\mathcal{G}}$-subroutine respecting in the sense of [CDPW07]. Then π EUC-emulates ϕ if and only if π MUC-emulates ϕ with respect to $\bar{\mathcal{G}}$.*

The following definition formalizes the version of subroutine respecting for MUC. In a nutshell, it is the same as for UC and EUC, but generalized to the setting of multiple protocols. That is the multiple protocols can share state and interact arbitrarily, but all communication with the outside world has to be performed via the any main party of any protocol.

Definition 3. Let $\vec{\pi} = (\pi_1, \ldots, \pi_n)$ be a vector of protocols. Then $\vec{\pi}$ is *jointly subroutine respecting* if all inputs and outputs of any subsidiary of any protocol in $\vec{\pi}$ happen through an instance of some party or a protocol in the vector $\vec{\pi}$. Let $\bar{\mathcal{G}}$ be a global shared functionality. Then $\vec{\pi}$ is *$\bar{\mathcal{G}}$-jointly subroutine respecting* if it is jointly subroutine respecting except that it also accesses the global shared functionality $\bar{\mathcal{G}}$.

The definition is considerably more lenient than the subroutine respecting definitions in [Can18, CDPW07]. Not only does it allow the communication between functionalities as needed, it even allows for communication between all possible sessions of those protocols! The reason we use this definition is that it is a lot simpler to write, that any restriction to a particular number of sessions sharing state would be arbitrary, and that our composition theorem does not require this restriction, as in contrast to [Can18] we do not reduce to a single session of the challenge protocol(s). Furthermore, for "reasonable" protocols this should not introduce excessive complexity in the proof beyond the inherent additional complexity of handling multiple parallel sessions.

The above definition then allows us to specify and prove the multi-protocol version of the composition theorem. We need to impose some restrictions which are necessary for the reduction proof to work, but which are anyway natural for practical protocols. We say that protocol vectors $\vec{\pi}$ and $\vec{\phi}$ are *compatible* if no execution of $\vec{\pi}$ creates a sub-protocol with code listed in $\vec{\phi}$ but not listed in $\vec{\pi}$, and vice versa.

Theorem 4 (Multi-protocol UC composition, recursive)**.** *Let $\bar{\mathcal{G}}$ be a global shared functionality, let $\vec{\pi}$ and $\vec{\phi}$ be vectors of compatible PPT protocols with $|\vec{\pi}| = |\vec{\phi}|$, such that $\vec{\pi}$ MUC-emulates $\vec{\phi}$ with respect to $\bar{\mathcal{G}}$, and both $\vec{\phi}$ and $\vec{\pi}$ are $\bar{\mathcal{G}}$-jointly subroutine respecting. Let ρ be a PPT protocol such that in no execution a party of ρ creates a sub-routine with code listed in $\vec{\pi}$ but not in $\vec{\phi}$.*[4] *Consider sub-vectors $\underset{\rightarrow}{\pi}$ and $\underset{\rightarrow}{\phi}$ that are obtained from $\vec{\pi}$ and $\vec{\phi}$ by selecting elements in the same positions. Then protocols $\rho^{\phi \twoheadrightarrow \pi}|\underset{\rightarrow}{\pi}$ MUC-emulate protocols $\rho|\underset{\rightarrow}{\phi}$ with respect to $\bar{\mathcal{G}}$.*

Proof. This proof is quite similar to the composition proof for GUC [CDPW07]. In particular, we also first observe (in the spirit of the original UC proof) that security against a dummy adversary is equivalent to security against all adversaries. Unlike UC, we also do not need a hybrid argument in the proof, since MUC security, like GUC, already models multiple sessions of the challenge protocols.

The universality of the dummy adversary follows exactly as in [Can18, CDPW07]. As $\vec{\pi}$ MUC-emulates $\vec{\phi}$ with respect to $\bar{\mathcal{G}}$, there is an adversary $\mathcal{S}_\pi$ such that

$$\text{MEXEC}^{\bar{\mathcal{G}}}_{\vec{\pi},\mathcal{D},\mathcal{Z}_\pi} \approx \text{MEXEC}^{\bar{\mathcal{G}}}_{\vec{\phi},\mathcal{S}_\pi,\mathcal{Z}_\pi}$$

for any environment $\mathcal{Z}_\pi$. We will use $\mathcal{S}_\pi$ to construct the simulator $\mathcal{S}_\rho$ for the protocol $\rho|\underset{\rightarrow}{\pi}$, which works as follows: $\mathcal{S}_\rho$ emulates a copy of $\mathcal{S}_\pi$ internally, forwarding all messages sent by the environment $\mathcal{Z}$ and addressed to instances of $\vec{\pi}$ (which includes all subroutines of those protocols) to this copy of $\mathcal{S}_\pi$, and analogously forwarding all messages sent by $\mathcal{S}_\pi$ to $\mathcal{Z}$. Additionally, $\mathcal{S}_\rho$ forwards all communication between its internal copy of $\mathcal{S}_\pi$ and the actual instances of the protocols $\vec{\phi}$ in its execution.

We prove the claimed equation $\text{MEXEC}_{\rho^{\vec{\phi} \twoheadrightarrow \vec{\pi}}|\underset{\rightarrow}{\pi},\mathcal{D},\mathcal{Z}} \approx \text{MEXEC}_{\rho|\underset{\rightarrow}{\phi},\mathcal{S}_\rho,\mathcal{Z}}$ by contradiction. Assume that there is an environment $\mathcal{Z}$ such that the equation does not hold. We construct an environment $\mathcal{Z}_\pi$ from $\mathcal{Z}$ as follows. Environment $\mathcal{Z}_\pi$ internally emulates a copy of $\mathcal{Z}$ and follows its instructions. Whenever $\mathcal{Z}$ provides input to ρ, then $\mathcal{Z}_\pi$ emulates the execution of the corresponding ITI executing ρ as well. When any of these instances of ρ or $\mathcal{Z}$ provide input to any other protocol of $\vec{\phi}$ (or $\underset{\rightarrow}{\phi}$, respectively, for $\mathcal{Z}$), then $\mathcal{Z}_\pi$ provides the corresponding input to the respective instance in its own experiment. Inputs from $\mathcal{Z}$ to $\bar{\mathcal{G}}$ and outputs from $\bar{\mathcal{G}}$ to $\mathcal{Z}$ are simply forwarded. As the protocols $\vec{\pi}$ and $\vec{\phi}$ are $\bar{\mathcal{G}}$-jointly subroutine respecting, all the input/output communication with external protocols in the experiment of $\mathcal{Z}$ is performed either via the main parties of $\vec{\pi}$ or directly with $\bar{\mathcal{G}}$, which ensures that $\mathcal{Z}_\pi$ can proceed as described. In terms of network communications, $\mathcal{Z}_\pi$ forwards the messages between $\mathcal{Z}$ and the emulated instances of ρ. Network communication from $\mathcal{Z}$ to $\vec{\pi}$ (or sub-parties thereof) is forwarded to the adversary in the experiment of $\mathcal{Z}_\pi$; likewise, outputs from that adversary are forwarded to the emulated $\mathcal{Z}$.

We observe that

$$\text{MEXEC}^{\bar{\mathcal{G}}}_{\vec{\pi},\mathcal{D},\mathcal{Z}_\pi} = \text{MEXEC}^{\bar{\mathcal{G}}}_{\rho^{\vec{\phi} \twoheadrightarrow \vec{\pi}}|\underset{\rightarrow}{\pi},\mathcal{D},\mathcal{Z}},$$

as $\mathcal{Z}_\pi$ emulates the ITIs for ρ and their interactions with other ITIs faithfully. We analogously observe that

$$\text{MEXEC}^{\bar{\mathcal{G}}}_{\vec{\phi},\mathcal{S}_\pi,\mathcal{Z}_\pi} = \text{MEXEC}^{\bar{\mathcal{G}}}_{\rho|\underset{\rightarrow}{\phi},\mathcal{S}_\rho,\mathcal{Z}},$$

as again $\mathcal{Z}_\pi$ emulates the protocol ρ and simulator $\mathcal{S}_\rho$ does the same as $\mathcal{S}_\pi$ and delivers messages to parties of ρ in the same way as $\mathcal{Z}_\pi$ does in the other experiment. Additionally, as ρ is restricted to not create subroutines in $\vec{\pi}$ except for those that already appear in $\vec{\phi}$, we make sure that ρ does not create sub-routines that could possibly interact with instances of $\vec{\phi}$ but cannot be emulated in the experiment $\text{MEXEC}^{\bar{\mathcal{G}}}_{\vec{\phi},\mathcal{S}_\pi,\mathcal{Z}_\pi}$.

[4] This is a somewhat analogous but stricter version of a condition in the definition of *compliant* in [Can18].

By our assumption, the environment $\mathcal{Z}$ succeeds in distinguishing, more formally

$$\text{MEXEC}^{\bar{\mathcal{G}}}_{\rho^{\vec{\phi}\to\vec{\pi}}|_{\vec{\pi}},\mathcal{D},\mathcal{Z}} \not\approx \text{MEXEC}^{\bar{\mathcal{G}}}_{\rho|_{\vec{\phi}},\mathcal{S}_\rho,\mathcal{Z}}.$$

By the above equalities this is equivalent to

$$\text{MEXEC}^{\bar{\mathcal{G}}}_{\vec{\pi},\mathcal{D},\mathcal{Z}_\pi} \not\approx \text{MEXEC}^{\bar{\mathcal{G}}}_{\vec{\phi},\mathcal{S}_\pi,\mathcal{Z}_\pi},$$

which contradicts the assumption that $\vec{\pi}$ MUC-emulates $\vec{\phi}$ with respect to $\bar{\mathcal{G}}$. This concludes the proof. □

As a remark, we point out two specific types of statements that are in particular implied by this composition theorem and relate the framework to standard UC.

- Any ($\bar{\mathcal{G}}$-subroutine respecting) protocol proven secure in EUC is also secure in multi-protocol UC; this is stated in Lemma 2, along with the corresponding statement for plain UC. In particular, EUC-secure of UC secure protocols that are in particular subroutine respecting (of the respective type) can immediately be used as subroutines in MUC-secure protocols.
- Let $\vec{\pi}$ and $\vec{\phi}$ be $\bar{\mathcal{G}}$-jointly subroutine respecting such that $\vec{\pi}$ emulates $\vec{\phi}$. Let ρ be a $\vec{\phi}$-hybrid protocol that is in particular subroutine respecting, meaning that its subroutines $\vec{\phi}$ are *not* exposed to the outside. Then $\rho^{\vec{\phi}\to\vec{\pi}}$ EUC-emulates ρ with respect to $\bar{\mathcal{G}}$.

4 Modular proof of a protocol

The goal of this section is to exemplify how the modular proof approach described in the introduction applies to the example protocol consisting of a non-interactive zero-knowledge proof π_{NIZK}, a non-interactive commitment scheme π_{NCOM}, and a signature scheme π_{SIG}. As described there, we idealize these protocols as functionalities $\mathcal{F}_{\text{NIZK}}$, $\mathcal{F}_{\text{NCOM}}$, and $\mathcal{F}_{\text{SIG}}$, respectively, but in a manner that these functionalities interact with each other and can all be accessed by other, higher-level protocols. We remark that the proof methodology showed in this section also applies to frameworks such as those of Backes, Pfitzmann, and Waidner [BPW04]; Küsters, Tuengerthal, and Rausch [KTR18]; and Maurer and Renner [MR11]. Of course the details of run-time and corruption handling must be adapted, and in those frameworks no additional modification of the composition theorem is required.[5]

But *how* do we realize functionalities in a way that they interact with each other? We begin by realizing functionality $\mathcal{F}_{\text{NIZK}}$—the one that depends on the code of the protocols π_{NCOM} and π_{SIG} via the relation R that parametrizes it. More concretely, for the relation R described in the introduction in terms of $\text{open}(\cdot,\cdot,\cdot)$ and $\text{ver}(\cdot,\cdot,\cdot)$, we let the protocol π^R_{NIZK} realize the functionality $\mathcal{F}^R_{\text{NIZK}}$. (A concrete instantiation is given in Section 5.3.) We then go on to re-write $\mathcal{F}_{\text{NIZK}}$ in a different way, namely using the protocols π_{NCOM} and π_{SIG} instead of the relation R, in Section 4.1. Except for technical details such as run-time, this formulation is obviously equivalent to the standard $\mathcal{F}^R_{\text{NIZK}}$. In Section 4.2, we then re-write the functionality to evaluate $\text{open}(\cdot,\cdot,\cdot)$ and $\text{ver}(\cdot,\cdot,\cdot)$ by creating *external* instances of π_{NCOM} and π_{SIG}. We explain that some care needs to be taken here with respect to corruptions and modeling details, but the issues can be overcome.

In the previous step, we did already provide the groundwork for having functionalities interact: Functionality $\mathcal{F}_{\text{NIZK}}$ invokes protocols π_{NCOM} and π_{SIG}. We can then use concrete instantiations of those protocols that, as shown in Sections 5.1 and 5.2, instantiate $\mathcal{F}_{\text{NCOM}}$ and $\mathcal{F}_{\text{SIG}}$. We discuss in Section 4.3 that the common security proofs for such schemes have to be adapted slightly in terms of corruption modeling. Using our variant of the UC theorem, we show in Section 4.4 how the results in the previous sections are put together.

[5] The methodology does not work in the framework of Hofheinz and Shoup [HS15], which inherently enforces a strict hierarchy on sub-protocols.

4.1 Writing the NIZK functionality with internal protocols

The next step is to rewrite the functionality $\mathcal{F}_{\text{NIZK}}$ into another functionality $\mathcal{F}_{\text{NIZK2}}$. In a nutshell, we specialize the relation R to $((pk, crs, c), (m, s, open)) : \text{ver}(pk, m, s) = 1 \wedge \text{open}(crs, c, open) = 1$; in other words, the goal is to prove knowledge of a message m, a signature s, and an opening information *open*, such that the commitment c, relative to the CRS *crs*, opens to the message m for which the prover knows the signature s relative to the public key pk. Needless to say, for this to work, the set of relations $\mathcal{R}$ supported by the zero-knowledge protocol π_{NIZK} must be compatible with the computations required by ver and open.

Functionality $\mathcal{F}_{\text{NIZK2}}^{(\pi_{\text{NCOM}}, \pi_{\text{SIG}})}$

$\mathcal{F}_{\text{NIZK}}$ is parametrized by algorithms π_{NCOM} and π_{SIG} used for verifying statements. It keeps an initially empty list L of proven statements.

1. Wait for an input $(\texttt{prove}, y, w)$ from P such that $y = (pid, sid_s, sid_c, pk, c)$ and $w = (m, s, o)$. Emulate ITMs π_{NCOM}, with identity (pid, sid_c), and π_{SIG}, with session identifier (pid, sid_s). Input $(\texttt{verify}, pk, m, s)$ to π_{SIG} and store the returned bit as b_{ver}. Input $(\texttt{open}, c, m, o)$ to π_{NCOM} and store the returned bit as b_{open}. Send $(\texttt{prove}, y)$ to $\mathcal{A}$.
2. Upon receiving a message $(\texttt{done}, \psi)$ from $\mathcal{A}$, with $\psi \in \{0,1\}^*$, record (y, ψ) in L and send $(\texttt{done}, \psi)$ to P.
3. Upon receiving $(\texttt{verify}, y, \psi)$ from some party P', check whether $(y, \psi) \in L$. If not, output $(\texttt{verify}, y, \psi)$ to $\mathcal{A}$ and upon receiving answer $(\texttt{witness}, w)$. Check the validity of (y, w) as in step 1. If the pair is valid, store (y, ψ) in L. If (y, ψ) has been stored, then output 1 to P', else output 0.

Figure 7: Non-interactive zero-knowledge functionality, written with emulated protocols π_{NCOM} and π_{SIG} instead of relation R.

By inspection, the functionality $\mathcal{F}_{\text{NIZK2}}^{(\pi_{\text{NCOM}}, \pi_{\text{SIG}})}$ as specified in Figure 7 has the same input/output behavior as the functionality $\mathcal{F}_{\text{NIZK}}^{R}$ for the appropriate relation R specified above. The only difference is in run-time, where the verification of $(y, w) \in R$ in $\mathcal{F}_{\text{NIZK}}$ may be implemented differently from the explicit verification via the protocols π_{NCOM} and π_{SIG} in $\mathcal{F}_{\text{NIZK2}}$. This is a minor difference that can easily be dealt with in $\mathcal{F}_{\text{NIZK2}}$ (at the cost of a higher run-time bound), which means that we obtain the following corollary to the results of Section 5.3.

Corollary 5. *Let π_{SIG} be a signature scheme and π_{NCOM} a non-interactive commitment, and let R be the relation as specified above. Let π_{NIZK} be a zero-knowledge protocol as specified in Section 2.3 which realizes the functionality $\mathcal{F}_{\text{NIZK}}^{R}$. Then π_{NIZK}^{R} UC-realizes the functionality $\mathcal{F}_{\text{NIZK2}}^{(\pi_{\text{NCOM}}, \pi_{\text{SIG}})}$.*

4.2 Writing the NIZK functionality with external protocols

In the next step, we again re-write the zero-knowledge functionality to obtain a functionality $\mathcal{F}_{\text{NIZK3}}$ that, instead of emulating the protocols π_{NCOM} and π_{SIG} internally, invokes corresponding ITIs external to $\mathcal{F}_{\text{NIZK3}}$ and uses them for the verification. Looking ahead, our goal is to replace those protocol instances by ideal functionalities, such that $\mathcal{F}_{\text{NIZK3}}$ communicates with idealized versions $\mathcal{F}_{\text{SIG}}$ and $\mathcal{F}_{\text{NCOM}}$.

There are two main obstacles we have to overcome before this approach can actually work out:

- The $(\texttt{prove}, y, w)$ operation in $\mathcal{F}_{\text{NIZK2}}$, emulating instances of π_{SIG} and π_{NCOM}, always succeeds (if the run-time requirements are satisfied). In an execution with $\mathcal{F}_{\text{NIZK3}}$, in contrast, $\mathcal{A}$ can create ITIs with identities (pid, sid_s) or (pid, sid_c) that run code that is different from π_{SIG} or π_{NCOM}, respectively. In this case, the operation $(\texttt{prove}, y, w)$ in $\mathcal{F}_{\text{NIZK3}}$ necessarily fails, since the invocation of the external

Functionality $\mathcal{F}_{\text{NIZK3}}^{(\pi_{\text{NCOM}},\pi_{\text{SIG}})}$

$\mathcal{F}_{\text{NIZK}}$ is parametrized by algorithms π_{NCOM} and π_{SIG} used for verifying statements. It keeps an initially empty list L of proven statements.

1. Wait for an input $(\texttt{prove}, y, w)$ from P such that $y = (pid, sid_s, sid_c, pk, c)$ and $w = (m, s, o)$. Check that pid is specified as uncorruptible in both sid_s and sid_c. Input $(\texttt{verify}, pk, m, s)$ to a possibly new ITM π_{SIG} with identity (pid, sid_s), obtain a return bit and store it as b_{ver}. Input $(\texttt{open}, c, m, o)$ to a possibly new ITM π_{NCOM} with identity (pid, sid_c), obtain a return bit and store it as b_{open}. If both operations return successful and $b_{\text{ver}} \wedge b_{\text{open}}$, then continue, else stop. Send $(\texttt{prove}, y)$ to $\mathcal{A}$.
2. Upon receiving a message $(\texttt{done}, \psi)$ from $\mathcal{A}$, with $\psi \in \{0,1\}^*$, record (y, ψ) in L and send $(\texttt{done}, \psi)$ to P.
3. Upon receiving $(\texttt{verify}, y, \psi)$ from some party P', check whether $(y, \psi) \in L$. If not, output $(\texttt{verify}, y, \psi)$ to $\mathcal{A}$ and upon receiving answer $(\texttt{witness}, w)$. Check the validity of (y, w) as in step 1. If the pair is valid, store (y, ψ) in L. If (y, ψ) has been stored, then output 1 to P', else output 0.

Figure 8: Non-interactive zero-knowledge functionality, calling out to external instances of π_{NCOM} and π_{SIG}.

ITIs does not yield the expected result (but instead fails). This can be checked by the simulator, which simply prevents $\mathcal{F}_{\text{NIZK3}}$ from continuing if this case occurs, and is described in Theorem 6.

- The instances of the ITMs π_{SIG} and π_{NCOM} that we create must be incorruptible, otherwise the functionality $\mathcal{F}_{\text{NIZK3}}$ is significantly weaker than its counterpart $\mathcal{F}_{\text{NIZK2}}$. On the other hand, other instances of the same ITMs must still be corruptible, as otherwise the overall security statement is not particularly interesting. This will be achieved by encoding in *sid* the *pid* of the ITI that is to be incorruptible.

Incorruptible protocols. The UC framework provides great flexibility in handling corruptions. Corruption is modeled via specific messages sent by the adversary to the targeted ITI, which then behaves according to its own instructions. This may, such as for the case of Byzantine corruption, mean that the ITI sends its entire state to the adversary and subsequently follows the adversary's instructions. Yet, other mechanisms of handling corruptions are also possible.

We use this flexibility of the UC framework to model the case where one participant cannot be corrupted. We specify variants π'_{SIG} and π'_{NCOM} of the protocols π_{SIG} and π_{NCOM}, in which the session identifier *sid* is a pair (pid, sid') such that *pid* specifies the identity of the uncorruptible participant and sid' is used as the session identifier of π_{SIG} or π_{NCOM}, respectively. The protocols π'_{SIG} and π'_{NCOM} behave exactly as π_{SIG} and π_{NCOM}, respectively, except for their behavior in the case of corruption: the participant identifier pid' of the ITI is compared to the value *pid* obtained from *sid*; if they are equal, the corruption request is ignored, otherwise the ITI proceeds as before.

For ease of notation, and where there is no risk for confusion, we denote denote π'_{SIG} and π'_{NCOM} by the symbols π_{SIG} and π_{NCOM} from here onwards.

Theorem 6. *Let ϕ_{NIZK2} be the dummy protocol for $\mathcal{F}_{\text{NIZK2}}^{(\pi_{\text{NCOM}},\pi_{\text{SIG}})}$ and ϕ_{NIZK3} be the dummy protocol for $\mathcal{F}_{\text{NIZK3}}^{(\pi_{\text{NCOM}},\pi_{\text{SIG}})}$. Protocol ϕ_{NIZK2} UC-emulates ϕ_{NIZK3}.*

Proof. The simulator $\mathcal{S}$ behaves transparently unless $\mathcal{A}$ instructs it to create an ITI with identity (pid, sid) that corresponds to one used by $\mathcal{F}_{\text{NIZK3}}$, but with code that differs from the expected one for either π_{NCOM} or π_{SIG}. In that case, $\mathcal{S}$ does not instruct $\mathcal{F}_{\text{NIZK3}}$ to output the proof, even if $\mathcal{A}$ instructs the expected instance of $\mathcal{F}_{\text{NIZK2}}$ accordingly.

It is easy to see that the simulation is perfect when no such ITI is created. If such an ITI is created, in both cases the functionality does makes no progress. This concludes the proof. □

We immediately obtain the analog of Corollary 5, namely that every protocol that UC-realizes $\mathcal{F}_{\text{NIZK}}$ and supports the relations obtained through π_{NCOM} and π_{SIG} also UC-realizes $\mathcal{F}_{\text{NIZK3}}$.

4.3 Realizing the functionalities $\mathcal{F}_{\text{SIG}}$ and $\mathcal{F}_{\text{NCOM}}$

Functionality $\mathcal{F}_{\text{NIZK3}}^{(\pi_{\text{NCOM}},\pi_{\text{SIG}})}$ calls out to protocols π_{NCOM} and π_{SIG} with specific party and session identifiers. The idea of this modeling is that other parties can also instantiate π_{NCOM} and π_{SIG} for the corresponding session: signatures and openings verified by $\mathcal{F}_{\text{NIZK3}}$ via its instances of π_{NCOM} and π_{SIG} must have been generated somewhere. The next step in our proof is then to idealize the guarantees of π_{NCOM} and π_{SIG} via functionalities $\mathcal{F}_{\text{NCOM}}$ and $\mathcal{F}_{\text{SIG}}$.

The functionality $\mathcal{F}_{\text{NIZK3}}^{(\pi_{\text{NCOM}},\pi_{\text{SIG}})}$ invokes variants of the protocols π_{NCOM} and π_{SIG} whose UC formalization is modified in a way as to make the instances generated by $\mathcal{F}_{\text{NIZK3}}$ incorruptible. Of course, this modeling only makes sense if we later idealize their guarantees in a way that the respective party in $\mathcal{F}_{\text{NCOM}}$ and $\mathcal{F}_{\text{SIG}}$ is incorruptible as well. Luckily, this is easy to do and analogous to the modeling for the protocols in Section 4.2: Before accepting a corruption request by the adversary, the functionalities check whether the intended *pid* is specified as incorruptible in the *sid*. A bit more precisely, in queries of the type $(\texttt{corrupt}, P)$, $\mathcal{F}_{\text{SIG}}$ checks whether the party identifier of P matches the value of *pid* specified in the *sid* of $\mathcal{F}_{\text{SIG}}$. If the values match, then the request is ignored. Otherwise the functionality proceeds as specified.

We write the following simple corollary to our security statements in Sections 5.1 and 5.2. The proof of the corollary is obvious and omitted.

Corollary 7. *The variants π'_{SIG} and π'_{NCOM} of the protocols in which a party can be specified as incorruptible in the sid (E)UC-realize the variants $\mathcal{F}'_{\text{SIG}}$ and $\mathcal{F}'_{\text{NCOM}}$ of the functionalities in which the respective party is incorruptible.*

In the following, we simply write $\mathcal{F}_{\text{SIG}}$ and $\mathcal{F}_{\text{NCOM}}$ instead of $\mathcal{F}'_{\text{SIG}}$ and $\mathcal{F}'_{\text{NCOM}}$ wherever it is clear from the context that we mean the variants with incorruptible parties.

4.4 Using MUC composition to combine the results

In the previous sub-sections, we showed how the zero-knowledge functionality $\mathcal{F}_{\text{NIZK}}^{R}$ realized as described in Section 5.3 can be rewritten to verify the relation by calling out to external protocols via Corollary 5 and Theorem 6. We also showed in Corollary 7 a modification of the results in Sections 5.1 and 5.2 that adapts the standard corruption handling to the one used in Theorem 6. The obvious missing step is to use the composition theorem in order to show that $\mathcal{F}_{\text{NIZK3}}^{(\pi_{\text{NCOM}},\pi_{\text{SIG}})}$ realizes $\mathcal{F}_{\text{NIZK3}}^{(\phi_{\text{NCOM}},\phi_{\text{SIG}})}$, or rather that protocol $\phi_{\text{NIZK3}}^{(\pi_{\text{NCOM}},\pi_{\text{SIG}})}$ realizes protocol $\phi_{\text{NIZK3}}^{(\phi_{\text{NCOM}},\phi_{\text{SIG}})}$. This is a priori not the case, since literally the universal composition operation $\rho^{\phi\to\pi}$ specified in [Can18] only affects the outermost protocol, which in the case of an ideal functionality is the dummy protocol, but we invoke the sub-protocols from the functionality, which is a subroutine of the dummy protocol! This is why we need the recursive composition operation defined in Section 3.1.

By the obvious adaptation of the UC composition theorem to recursive composition, which we stated as Theorem 15 in Appendix A, we can then conclude the result stated above, namely that $\phi_{\text{NIZK3}}^{(\pi_{\text{NCOM}},\pi_{\text{SIG}})}$ UC-realizes $\phi_{\text{NIZK3}}^{(\phi_{\text{NCOM}},\phi_{\text{SIG}})}$. This statement, however, turns out to be almost useless for the practical use of ϕ_{NIZK3}: In order to use ϕ_{NIZK3} as a sub-protocol, it needs to be subroutine respecting! For ϕ_{NIZK3} to be subroutine respecting means that the input/output interfaces of its subroutines, in particular $\mathcal{F}_{\text{SIG}}$ and $\mathcal{F}_{\text{NCOM}}$, must not be usable by any protocol other than ϕ_{NIZK3}. This renders the triplet of functionalities $(\mathcal{F}_{\text{NIZK}}, \mathcal{F}_{\text{NCOM}}, \mathcal{F}_{\text{SIG}})$ almost useless, since no signatures or commitments can be created.

To resolve the above conundrum, we turn to the MUC model and Theorem 4, its composition theorem. We first use Lemma 2 to adapt the security statements made in Theorem 6 and Corollary 7 for the protocols π_{NIZK}, π_{SIG}, and π_{NCOM} from the UC model to the MUC model. We then finally invoke Theorem 4 to conclude

that the protocol vector $(\phi_{\text{NIZK}}, \pi_{\text{SIG}}, \pi_{\text{NCOM}})$ MUC-realizes the protocol vector $(\phi_{\text{NIZK}}, \phi_{\text{SIG}}, \phi_{\text{NCOM}})$. This implies our final result, which is formalized in the following theorem.

Theorem 8. *Let π_{SIG}, π_{NCOM}, and π_{NIZK} be the protocols described in Section 5 and the relation R as described above. Then the protocol vector $(\pi^R_{\text{NIZK}}, \pi_{\text{SIG}}, \pi_{\text{NCOM}})$ MUC-realizes the protocol vector $(\phi^{(\phi_{\text{NCOM}}, \phi_{\text{SIG}})}_{\text{NIZK3}}, \phi_{\text{SIG}}, \phi_{\text{NCOM}})$ with respect to $\bar{\mathcal{G}}_{\text{sRO}}$. The vector of dummy protocols can also be viewed as the vector $(\mathcal{F}^{(\phi_{\text{SIG}}, \phi_{\text{NCOM}})}_{\text{NIZK3}}, \mathcal{F}_{\text{SIG}}, \mathcal{F}_{\text{NCOM}})$ of functionalities.*

We emphasize that the strength of our result, in comparison with a formalization in the standard UC framework [Can18], is that protocols such as the one described here can be developed using a set of ideal functionalities that abstract the underlying schemes and interact with each other, while still being usable by higher-level protocols in the expected way. This simplifies protocol development, since one can actually work with idealized components that abstract from the intricacies of the component schemes, and it allows to specify intermediate protocols as a combination of (few) simple and standard components instead of as a monolithic, tailored, and complex intermediate functionality.

5 Concrete Protocols

Section 4 showed how realizations of signatures, commitments, and zero-knowledge proofs can securely be combined. In this section we will propose concrete instantiations that are compatible and efficient, meaning that the zero-knowledge protocol can prove statements about signature verifications and commitment openings. To this end, we will fix a single family of bilinear groups $\mathsf{bilinearGroups}$, such that $\mathsf{bilinearGroups}(\lambda) = (q, \mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_t, e, g, \tilde{g})$, with $\mathbb{G} = \langle g \rangle$, $\tilde{\mathbb{G}} = \langle \tilde{g} \rangle$, and $\mathbb{G}_t$ of prime order q, with bilinear map e, of a security level matching λ. All our algorithms will implicitly be parametrized by $\mathsf{bilinearGroups}$ and obtain the correct groups for the security parameter.

5.1 Signatures

Before we present our concrete realization of $\mathcal{F}_{\text{SIG}}$, we show that any signature scheme secure in the property-based sense (i.e., EUF-CMA as defined in [GMR88]). Similar generic realizations of $\mathcal{F}_{\text{SIG}}$ have been shown for slightly different formulations of $\mathcal{F}_{\text{SIG}}$ [Can04, BMT18]. Then, we use this result to show that the structure-preserving signature scheme by Abe et al. [AGOT14] is a secure realization of $\mathcal{F}_{\text{SIG}}$.

5.1.1 Generic realization of $\mathcal{F}^{(\text{kgen},\text{sign},\text{ver})}_{\text{SIG}}$ from EUF-CMA

Let algorithms $\text{kgen}, \text{sign}, \text{ver}$ be an EUF-CMA signature scheme. We now describe the following generic "wrapper" algorithm $\pi^{(\text{kgen},\text{sign},\text{ver})}_{\text{SIG}}$ that realizes $\mathcal{F}^{(\text{kgen},\text{sign},\text{ver})}_{\text{SIG}}$.

Protocol $\pi^{(\text{kgen},\text{sign},\text{ver})}_{\text{SIG}}$

Upon a `pubkey` or `sign` query, if (sk, pk) not yet defined, then run $(sk, pk) \leftarrow\!\!\$\ \text{kgen}(\lambda)$, where λ is obtained from the security parameter tape, and store (sk, pk).

1. Upon input `pubkey` from party P, output $(\texttt{pubkey}, pk)$ to P.
2. Upon input $(\texttt{sign}, m)$ from party P with $m \in \{0,1\}^*$, compute $s \leftarrow\!\!\$\ \text{sign}(sk, m)$. If $s = \bot \vee \neg(\text{ver}(pk, m, s) \vee S \in \mathcal{C}$ then output $\bot$ to P. Else set $\mathcal{L} \leftarrow \mathcal{L} \cup \{m\}$ and output s to P.
3. Upon input $(\texttt{verify}, pk', m', s')$ from party P, compute $b \leftarrow \text{ver}(pk', m', s')$. If $S \notin \mathcal{C} \wedge pk = pk' \wedge b \wedge m' \notin \mathcal{L}$ then output $(\texttt{result}, \bot)$ to P. Else output $(\texttt{result}, b)$ to P.

We will assume that $\pi^{(\text{kgen},\text{sign},\text{ver})}_{\text{SIG}}$ uses *secure erasures*, meaning that any state of protocol participants that is not explicitly persisted will be securely deleted. This means that if a party later becomes corrupted,

the adversary will not be able to see protocol state other than the explicitly persisted values. For our signature protocol, the only persisted state consists of the signer's key pair.

Theorem 9. *Protocol* $\pi_{\text{SIG}}^{(\text{kgen},\text{sign},\text{ver})}$ *UC emulates* $\mathcal{F}_{\text{SIG}}^{(\text{kgen},\text{sign},\text{ver})}$ *if* $(\text{kgen}, \text{sign}, \text{ver})$ *is existentially unforgeable against an adaptive chosen message attack, and assuming secure erasures.*

Proof. Let simulator $\mathcal{S}$ internally simulate $\mathcal{A}$. As $\mathcal{F}_{\text{SIG}}^{(\text{kgen},\text{sign},\text{ver})}$ handles signing and verification queries, $\mathcal{S}$ only comes into action when it is notified by $\mathcal{F}_{\text{SIG}}^{(\text{kgen},\text{sign},\text{ver})}$ of a party P becoming corrupted, and $\mathcal{S}$ must present a convicing state for simulated party "P". If P is a verifier (i.e., not the signer S), then the simulator does not need to prepare any state, as verifiers do not maintain any state. If P is the signer, then the state should consist of the key pair, which $\mathcal{S}$ can simulate as the key is obtained from $\mathcal{F}_{\text{SIG}}^{(\text{kgen},\text{sign},\text{ver})}$.

Observe that the simulation is almost perfect: Both the real and ideal world use algorithms kgen, sign, and ver to generate keys, sign messages, and verify signatures, respectively, and adaptively corrupted parties can be simulated perfectly. The only difference are the additional checks that $\mathcal{F}_{\text{SIG}}^{(\text{kgen},\text{sign},\text{ver})}$ performs. When signing, it only outputs signatures that are valid w.r.t. the verification algorithm. By completeness of the signature algorithms, this can only occur with negligible probability. When verifying, the $\mathcal{F}_{\text{SIG}}^{(\text{kgen},\text{sign},\text{ver})}$ will reject forgeries, i.e., signatures on messages that were never signed by the honest signer. An environment that causes this case to apply with non-negligible probability directly translates to a EUF-CMA forger with non-negligible advantage, contradicting the assumption that $(\text{kgen}, \text{sign}, \text{ver})$ is EUF-CMA.

□

5.1.2 AGOT structure-preserving signature

We recall the AGOT structure-preserving signature scheme $(\text{kgen}_{\mathsf{agot}}, \text{sign}_{\mathsf{agot}}, \text{ver}_{\mathsf{agot}})$ as introduced by Abe et al [AGOT14].

$\text{kgen}_{\mathsf{agot}}()$: Choose a random $v \leftarrow\!\!\$\ \mathbb{Z}_q$, $x \leftarrow\!\!\$\ \mathbb{G}$, compute $y \leftarrow \tilde{g}^v$, and set $(sk, pk) \leftarrow (v, (y, x))$.

$\text{sign}_{\mathsf{agot}}(par, sk, m)$: Compute

$$u \leftarrow\!\!\$\ \mathbb{Z}_q^*, \quad r \leftarrow \tilde{g}^u, \quad s \leftarrow (m^v \cdot x)^{1/u}, \quad t \leftarrow (s^v \cdot g)^{1/u}$$

and output $\sigma \leftarrow (r, s, t)$.

$\text{ver}_{\mathsf{agot}}(par, pk, m, \sigma)$: Parse $\sigma = (r, s, t)$, and $pk = (y, x)$ and output 1 if $m, s, t \in \mathbb{G}, r \in \tilde{\mathbb{G}}$, $r \neq 1_{\tilde{\mathbb{G}}}$, $e(s, r) = e(m, y) \cdot e(x, \tilde{g})$, and $e(t, r) = e(s, y) \cdot e(g, \tilde{g})$. Output 0 otherwise.

Assumption 10. Define the AGOT-advantage of an adversary as follows.

$$\begin{aligned}\mathsf{Adv}_{\mathsf{agot}}(\mathcal{A}) = \mathsf{Pr}\Big[&(q, \mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_t, e, g, \tilde{g}) \leftarrow \mathsf{bilinearGroups}(\lambda), v \leftarrow\!\!\$\ \mathbb{Z}_q, x \leftarrow\!\!\$\ \mathbb{G}, y \leftarrow \tilde{g}^v,\\ &(m, r, s, t) \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{v}}(\cdot)}(q, \mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_t, e, g, \tilde{g}, x, y) : (m, s, t) \in \mathbb{G}^3 \wedge r \in \tilde{\mathbb{G}} \wedge m \notin Q \wedge\\ &e(s, r) = e(m, y) \cdot e(x, \tilde{g}) \wedge e(t, r) = e(s, y) \cdot e(g, \tilde{g})\Big],\end{aligned}$$

where $\mathcal{O}^{\mathsf{v}}$ on input $m \in \mathbb{G}$ takes $u \leftarrow\!\!\$\ \mathbb{Z}_q$, $r \leftarrow \tilde{g}^u$, $s \leftarrow (m^v \cdot x)^{1/u}$, $t \leftarrow (s^v \cdot g)^{1/u}$, adds m to initially empty set Q and returns (r, s, t). The AGOT assumption states that no PPT adversary $\mathcal{A}$ has non-negligible $\mathsf{Adv}_{\mathsf{agot}}(\mathcal{A})$.

Abe et al. [AGOT14] prove that this assumption holds in the generic group model.

Theorem 11. *Signature scheme* $(\text{kgen}_{\mathsf{agot}}, \text{sign}_{\mathsf{agot}}, \text{ver}_{\mathsf{agot}})$ *is EUF-CMA secure under Assumption 10.*

Proof. As Assumption 10 is effectively equivalent to the EUF-CMA game for the AGOT signature scheme, this theorem follows immediately. □

From Theorem 9 and Theorem 11 the following corollary follows.

Corollary 12. *Protocol* $\pi_{\text{SIG}}^{(\text{kgen}_{\mathsf{agot}},\text{sign}_{\mathsf{agot}},\text{ver}_{\mathsf{agot}})}$ *UC emulates* $\mathcal{F}_{\text{SIG}}^{(\text{kgen}_{\mathsf{agot}},\text{sign}_{\mathsf{agot}},\text{ver}_{\mathsf{agot}})}$ *under Assumption 10 and assuming secure erasures.*

Protocol $\pi_{\text{NCOM}}^{(\mathcal{M},\text{commit},\text{open})}$ – $\mathcal{F}_{\text{CRS}}^{\text{crsgen}}, \bar{\mathcal{G}}_{\text{sRO}}$ hybrid

Algorithms commit and open may use strict global random oracle $\bar{\mathcal{G}}_{\text{sRO}}$.

1. Upon input $(\texttt{validate}, par')$ from P, get par from $\mathcal{F}_{\text{CRS}}$, check $par \stackrel{?}{=} par'$, and report to P.
2. Upon $(\texttt{commit}, par', x)$ with $x \in \mathcal{M}$ from P, compute $(c, open) \leftarrow\!\!\$\ \text{commit}(par', x)$, and return $(c, open)$ to P.
3. Upon $(\texttt{open}, par', c, open)$ from some P, compute $x' \leftarrow \text{open}(par', c, open)$ and output x' to P.

Figure 9: Commitment protocol wrapper $\pi_{\text{NCOM}}^{(\mathcal{M},\text{commit},\text{open})}$.

5.2 Commitments

We build a new non-interactive extractable commitment scheme. On a high level, a commitment to m is an Elgamal ciphertext to a public key from the CRS, which makes commitments extractable. However, as we will require a hiding property of honestly generated commitments while simultaneously extracting from adversarially created commitments, we need a CCA property of the encryption scheme. We therefore encrypt to two CRS public keys and perform a Schnorr-proof proving both ciphertexts encrypt the same value, following the Naor-Yung paradigm [NY90]. The Schnorr-proof works a strict global random oracle $\bar{\mathcal{G}}_{\text{sRO}}$, as defined by Camenisch et al. [CDG+18].

$\mathcal{F}_{\text{NCOM}}$ further requires that commitments are equivocable, so the simulator must be able to open a commitment to any value decided later. To achieve this property, we do not open a commitment to message m by simply revealing the randomness used to encrypt, but insteqd the opening consists of a zero-knowledge proof that proves the ciphertext indeed encrypts m. Simplified Groth-Sahai proofs [GS08, CCS09] are used for this purpose.

We now describe commitment algorithms $(\text{commit}_{\text{gs}}, \text{open}_{\text{gs}})$ that are parametrized by a fixed family of bilinear groups $\mathsf{bilinearGroups}$, and every algorithm uses the groups of the desired security level $\mathsf{bilinearGroups}(\lambda) = (q, \mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_t, e, g, \tilde{g})$. The commitment scheme will use message set $\mathcal{M} = \mathbb{G}$.

$\text{crsgen}_{\text{gs}}()$: Take $(y_1, y_2, \tilde{g}_1, \tilde{g}_2, \tilde{g}_3, \tilde{g}_4) \leftarrow\!\!\$\ \mathbb{G}^2 \cdot \tilde{\mathbb{G}}^4$.

$\text{commit}_{\text{gs}}(par, x)$: Parse par as $(y_1, y_2, \tilde{g}_1, \tilde{g}_2, \tilde{g}_3, \tilde{g}_4) \in \mathbb{G}^2 \cdot \tilde{\mathbb{G}}^4$. Choose $(\omega, \rho, \gamma) \leftarrow\!\!\$\ \mathbb{Z}_q^3$ and compute $c = (c_1, c_2, c_3, c_4, c_5) = (g^\omega, my_1^\omega, my_2^\omega, \bar{\mathcal{G}}_{\text{sRO}}(par, c_1, c_2, c_3, g^\rho, (c_2/c_3)^\rho), \rho + c_4 \cdot \omega \pmod q)$. Compute $\texttt{open} = (\tilde{o}_1, \tilde{o}_2, o_3, o_4) = (\tilde{g}_1^\omega \tilde{g}_2^\gamma, \tilde{g}_3^\omega \tilde{g}_4^\gamma, g^\gamma, y_1^\gamma)$. Return $(c, \texttt{open})$.

$\text{open}_{\text{gs}}(par, c, \tilde{o}, m)$: Parse par as $(y_1, y_2, \tilde{g}_1, \tilde{g}_2, \tilde{g}_3, \tilde{g}_4) \in \mathbb{G}^2 \cdot \tilde{\mathbb{G}}^4$, parse c as $(c_1, c_2, c_3, c_4, c_5)$, parse $\tilde{o}$ as $(\tilde{o}_1, \tilde{o}_2, o_3, o_4) \in \tilde{\mathbb{G}}^2 \cdot \mathbb{G}^2$, and check $m \in \mathbb{G}$. Check $c_4 = \bar{\mathcal{G}}_{\text{sRO}}(par, c_1, c_2, c_3, g^{c_5} \cdot c_1^{-c_4}, (y_1/y_2^{c_5} \cdot (c_2/c_3))^{-c_4})$ and check

$$e(g, \tilde{o}_1) = e(c_1, \tilde{g}_1)e(o_3, \tilde{g}_2), \quad e(g, \tilde{o}_2) = e(c_1, \tilde{g}_3)e(o_3, \tilde{g}_4),$$
$$e(y_1, \tilde{o}_1) = e(c_2/m, \tilde{g}_1)e(o_4, \tilde{g}_2), \quad e(y_1, \tilde{o}_2) = e(c_2/m, \tilde{g}_3)e(o_4, \tilde{g}_4).$$

Output 1 if all checks hold and output 0 otherwise.

Commitment protocol wrapper π_{NCOM}, as defined in Figure 9, wraps the commitment algorithms into a UC protocol. Next, we introduce algorithms $\text{tdcom}, \text{tdopen}, \text{extract}_{\text{gs}}$, again parametrized by family of bilinear groups $\mathsf{bilinearGroups}$, such that $\pi_{\text{NCOM}}^{(\text{commit}_{\text{gs}}, \text{open}_{\text{gs}})}$ will realize $\mathcal{F}_{\text{NCOM}}^{(\mathcal{M}, \text{open}_{\text{gs}}, \text{tdcom}_{\text{gs}}, \text{tdopen}_{\text{gs}})}$.

$\text{tdcom}(par, td)$: Parse par as $(y_1, y_2, \tilde{g}_1, \tilde{g}_2, \tilde{g}_3, \tilde{g}_4) \in \mathbb{G}^2 \cdot \tilde{\mathbb{G}}^4$ and td as $(x_1, a) \in \mathbb{Z}_q^2$. Take $\omega \leftarrow\!\!\$\ \mathbb{Z}_q$ and $m \leftarrow\!\!\$\ \mathbb{G}$ and compute $c = (c_1, c_2, c_3, c_4, c_5) = (g^\omega, my_1^\omega, my_2^\omega, \bar{\mathcal{G}}_{\text{sRO}}(par, c_1, c_2, c_3, g^\rho, (c_2/c_3)^\rho), \rho + c_4 \cdot \omega)$. Let $info \leftarrow (c, td)$ and output $(c, info)$.

$\text{tdopen}(par, x, info)$: Parse par as $(y_1, y_2, \tilde{g}_1, \tilde{g}_2, \tilde{g}_3, \tilde{g}_4) \in \mathbb{G}^2 \cdot \tilde{\mathbb{G}}^4$ and $info$ as $((c_1, c_2, c_3, c_4, c_5), a)$, such that $\tilde{g}_1 = \tilde{g}_2^a$ and $\tilde{g}_3 = \tilde{g}_4^a$. Compute $\texttt{open} = (\tilde{o}_1, \tilde{o}_2, o_3, o_4) = (\tilde{g}_2^\gamma, \tilde{g}_4^\gamma, c_1^{-a} \cdot g^\gamma, (c_2/m)^{-a} \cdot y_1^\gamma)$.

$\text{extract}_{\text{gs}}(par, td, c)$: Parse par as $(y_1, y_2, \tilde{g}_1, \tilde{g}_2, \tilde{g}_3, \tilde{g}_4) \in \mathbb{G}^2 \cdot \tilde{\mathbb{G}}^4$ and td as $(x_1, a) \in \mathbb{Z}_q^2$ such that $y_1 = g^{x_1}$. Parse c as $(c_1, c_2, c_3, c_4, c_5)$ and output $c_2 \cdot c_1^{-x_1}$.

Theorem 13. *Protocol* $\pi_{\text{NCOM}}^{(\mathcal{M}, \text{commit}_{\text{gs}}, \text{open}_{\text{gs}})}$ *EUC emulates* $\mathcal{F}_{\text{NCOM}}^{(\mathcal{M}, \text{open}_{\text{gs}}, \text{tdcom}_{\text{gs}}, \text{tdopen}_{\text{gs}}, \text{extract}_{\text{gs}})}$ *in the* $\mathcal{F}_{\text{CRS}}^{\text{crsgen}_{\text{gs}}}$, $\bar{\mathcal{G}}_{\text{sRO}}$*-hybrid model under the SXDH assumption.*

Proof. We will use a sequence of games to show the indistinguishability of the real and ideal world. Let $\mathcal{F}_{\text{NCOM}}$ denote $\mathcal{F}_{\text{NCOM}}^{(\mathcal{M}, \text{open}_{\text{gs}}, \text{tdcom}_{\text{gs}}, \text{tdopen}_{\text{gs}}, \text{extract}_{\text{gs}})}$.

Game 1: This is the real world.

Game 2: We now run a functionality $\mathcal{F}$ that handles `setup`, `validate`, `corrupt-status` and `corrupt` queries like $\mathcal{F}_{\text{NCOM}}$. It further accepts `commit` inputs like $\mathcal{F}_{\text{NCOM}}$, except case 3b. In this case, $\mathcal{F}$ computes $(c, info) \leftarrow\!\!\$\ \text{commit}_{\text{gs}}(par, x)$. Similarly, $\mathcal{F}$ accepts *open* inputs like $\mathcal{F}_{\text{NCOM}}$, except case 4b, where it skips the extra checks 4(b)i and 4(b)ii. We run $\mathcal{F}$ with simulator $\mathcal{S}$ that honestly simulates $\mathcal{F}_{\text{CRS}}$ and internally runs $\mathcal{A}$. Upon first activation, it obtains par from "$\mathcal{F}_{\text{CRS}}^{\text{crsgen}_{\text{gs}}}$", sets $td \leftarrow \perp$ and gives $\mathcal{F}$ input $(\texttt{setup}, par, td)$.

Observe that this game hop only restructures Game 1, the same inputs are accepted and the outputs are generated using the same algorithms, so we have Game 2 = Game 1.

Game 3: We now change $\mathcal{F}$'s behavior when handling `open` inputs and add the check 4(b)i from $\mathcal{F}_{\text{NCOM}}$. Commitments are generated inside $\mathcal{F}$ using $\text{commit}_{\text{gs}}$, and the resulting commitment $c = (c_1, c_2, c_3, c_4, c_5)$ is such that (c_1, c_2) forms an Elgamal ciphertext under public key y_1 from the crs, so clearly, a unique message is contained in (c_1, c_2). The opening is a simplified GS proof [GS08, CCS09] that proves there exists some γ such that $c_1 = g^\gamma$ and $c_2/m = y_1^\gamma$. With overwhelming probability, we have GS parameters that guarantee perfectly sound proofs, so honestly generated commitments can only be opened to the correct message m and this check does not change the view of the environment, Game 3 $\approx$ Game 2.

Game 4: The simulator in this game behaves as in the previous game, except that when simulating $\mathcal{F}_{\text{CRS}}^{\text{crsgen}_{\text{gs}}}$, it computes y_1 by taking $x_1 \leftarrow\!\!\$\ \mathbb{Z}_q$ and setting $y_1 = g^{x_1}$. It now sets $td \leftarrow (x_1, \perp)$ as the trapdoor given to $\mathcal{F}$ with the `setup` input. Since y_1 is distributed as before and $\mathcal{F}$ does not use x_1, we have Game 4 = Game 3.

Game 5: We now change $\mathcal{F}$'s behavior when handling `open` inputs and add the check 4(b)ii from $\mathcal{F}_{\text{NCOM}}$. Note that before this check, $\mathcal{F}$ already checked the opening, i.e., it checked the simplified GS proof proving that (c_1, c_2) encrypts message m under y_1. By the soundness of the simplified GS proofs, (c_1, c_2) indeed encrypts m, so decrypting (c_1, c_2) will indeed yield m and this check will not alter the output, and therefore Game 5 $\approx$ Game 4.

Game 6: The simulator now simulates $\mathcal{F}_{\text{CRS}}^{\text{crsgen}_{\text{gs}}}$ differently, and instead of taking $(\tilde{g}_1, \tilde{g}_2, \tilde{g}_3, \tilde{g}_4) \leftarrow\!\!\$\ \tilde{\mathbb{G}}$, it takes $(\tilde{g}_2, \tilde{g}_4) \leftarrow\!\!\$\ \tilde{\mathbb{G}}$ and $a \leftarrow\!\!\$\ \mathbb{Z}_q$, and computes $\tilde{g}_1 = \tilde{g}_2^a$ and $\tilde{g}_3 = \tilde{g}_4^a$. It now sets $td \leftarrow (x_1, a)$ as the trapdoor given to $\mathcal{F}$ with the `setup` input (where x_1 is computed as before). While $(\tilde{g}_1, \tilde{g}_2, \tilde{g}_3, \tilde{g}_4)$ as now slightly differently distributed, distinguishing these is exactly XDH problem, so under the XDH assumption in $\tilde{\mathbb{G}}$, we have Game 6 $\approx$ Game 5.

Game 7: When $\mathcal{F}$ creates honest commitments, it now simulates the simplified GS proof using simulation trapdoor a. More precisely, it computes $\texttt{open} = (\tilde{o}_1, \tilde{o}_2, o_3, o_4, o_5)$ as in algorithm $\text{tdopen}_{\text{gs}}$. As the simulation is perfect, we have Game 7 = Game 6.

Game 8: $\mathcal{F}$ now creates honest commitments (case 3b) as $\mathcal{F}_{\text{NCOM}}$, i.e., it uses algorithms tdcom_{gs} and $\text{tdopen}_{\text{gs}}$. The difference is that when an honest party commits to message m, the generated commitments no longer encrypts m but a random message instead, and the simplified GS proof (which is simulated since Game 7) now proves the false statement that it does encrypt m.

We make this change gradually.

- Let GAME 8.i.0 denote the game in which the first honestly generated commitments are generated using tdcom_{gs} and $\text{tdopen}_{\text{gs}}$, and the remaining ones are generated as in GAME 7.

- Consider GAME 8.i.1, in which $\mathcal{F}$ simulates Schnorr proof (c_4, c_5) of the $i+1$-th honest commitment by taking $h \leftarrow\!\!\$\ \{0,1\}^{\ell(\lambda)}$, $c_4 \leftarrow\!\!\$\ h \pmod q$, $c_5 \leftarrow\!\!\$\ \mathbb{Z}_q$, and programming $\bar{\mathcal{G}}_{\text{sRO}}$ on $(par, c_1, c_2, c_3, g^{c_5} \cdot c_1^{-c_4}, (y_1/y_2^{c_5} \cdot (c_2/c_3))^{-c_4})$.[6]

  As this simulation is perfect, we have GAME 8.i.1 = GAME 8.i.0.

- Next, consider GAME 8.i.2, in which $\mathcal{F}$ encrypts the correct message under y_1 but not under y_2 when creating the $i+1$-th commitment. More precisely, it takes a random message $m' \leftarrow\!\!\$\ \mathbb{G}$, and computes $c_1 \leftarrow g^\omega, c_2 \leftarrow m y_1^\omega, c_3 \leftarrow m' y_2^\omega$.

  Distinguishing GAME 8.i.2 from GAME 8.i.1 is equivalent to solving XDH in $\mathbb{G}$: The reduction interprets the XDH instance as (i_1, i_2, i_3) and sets $c_1 \leftarrow i_1$, $y_2 \leftarrow i_2$, $c_2 \leftarrow c_1^{x_1} \cdot m$, and $c_3 \leftarrow i_3 \cdot m$. Observe that if we have $\log_g(i_1) = \log_{i_2}(i_3)$ this is distributed equally to GAME 8.i.1 and otherwise it is distributed equally to GAME 8.i.2, so GAME 8.i.2 $\approx$ GAME 8.i.1 under the XDH assumption.

- Next, consider GAME 8.i.3, in which $\mathcal{F}$ runs a modified $\text{extract}_{\text{gs}}$ algorithm. The trapdoor now contains x_2, the discrete logarithm of y_2, and the algorithm extracts from (c_1, c_3) instead of (c_1, c_2). By the soundness of the Schnorr proof (c_4, c_5), (c_1, c_2) decrypts to the same value as (c_1, c_3), so GAME 8.i.3 $\approx$ GAME 8.i.2.

- Next, consider GAME 8.i.4, in which $\mathcal{F}$ encrypts random message m'' y_1 and random message m' under y_2 when creating the $i+1$-th commitment. More precisely, it takes a random messages $(m', m'') \leftarrow\!\!\$\ \mathbb{G}^2$, and computes $c_1 \leftarrow g^\omega, c_2 \leftarrow m'' y_1^\omega, c_3 \leftarrow m' y_2^\omega$.

  Distinguishing GAME 8.i.4 from GAME 8.i.3 is equivalent to solving XDH in $\mathbb{G}$: The reduction interprets the XDH instance as (i_1, i_2, i_3) and sets $c_1 \leftarrow i_1$, $y_1 \leftarrow i_2$, $c_2 \leftarrow i_3 \cdot m$, and $c_3 \leftarrow c_1^{x_2} \cdot m'$. Observe that if we have $\log_g(i_1) = \log_{i_2}(i_3)$ this is distributed equally to GAME 8.i.3 and otherwise it is distributed equally to GAME 8.i.4, so GAME 8.i.4 $\approx$ GAME 8.i.3 under the XDH assumption.

- Next, consider GAME 8.i.5, in which $\mathcal{F}$ encrypts the same random message m' under both y_1 and y_2 when creating the $i+1$-th commitment. More precisely, it takes a random messages $m' \leftarrow\!\!\$\ \mathbb{G}$, and computes $c_1 \leftarrow g^\omega, c_2 \leftarrow m' y_1^\omega, c_3 \leftarrow m' y_2^\omega$.

  Distinguishing GAME 8.i.5 from GAME 8.i.4 is equivalent to solving XDH in $\mathbb{G}$: The reduction interprets the XDH instance as (i_1, i_2, i_3) and sets $c_1 \leftarrow i_1$, $y_1 \leftarrow i_2$, $c_2 \leftarrow i_3 \cdot m'$, and $c_3 \leftarrow c_1^{x_2} \cdot m'$. Observe that if we have $\log_g(i_1) = \log_{i_2}(i_3)$ this is distributed equally to GAME 8.i.5 and otherwise it is distributed equally to GAME 8.i.4, so GAME 8.i.5 $\approx$ GAME 8.i.4 under the XDH assumption.

- Next, GAME 8.i.6 switched back to the standard $\text{extract}_{\text{gs}}$ algorithm, meaning that the simulator passess trapdoor $x_1 = \log_g(y_1)$ and $\mathcal{F}$ uses (c_1, c_2) to extract. Again by the simulation soundness of Schnorr proof (c_4, c_5), we have GAME 8.i.6 $\approx$ GAME 8.i.5.

Observe that in GAME 8.i.6, the Schnorr proof proves a true statement, and the only difference between GAME 8.i.6 and GAME 8.$(i+1)$.0 is that this proof is simulated rather than honestly computed. As this simulation is perfect, GAME 8.i.6 = GAME 8.$(i+1)$.0

As GAME 8.0 = GAME 7, there are at most polynomially many commit queries, and GAME 8.i is indistinguishable from GAME 8.$(i+1)$, we have GAME 8 $\approx$ GAME 7.

□

[6]Note that we are programming the random oracle here while the strict global random oracle $\bar{\mathcal{G}}_{\text{sRO}}$ does not allow for this. This is acceptable as it is only used to show that no distinguishing environment can exist, only in the final game we need to have the strict separation between the simulator, functionality, and $\bar{\mathcal{G}}_{\text{sRO}}$. In later steps we remove this programming.

5.3 Zero-knowledge proofs

To complete the instantiation of our modular construction, we need a NIZK protocol compatible with our commitment protocol $\pi_{\text{NCOM}}^{(\mathcal{M},\text{commit}_{\text{gs}},\text{open}_{\text{gs}})}$ and our signature protocol $\pi_{\text{SIG}}^{(\text{kgen}_{\mathsf{agot}},\text{sign}_{\mathsf{agot}},\text{ver}_{\mathsf{agot}})}$. That is, we want to realize $\mathcal{F}_{\text{NIZK}}^{R}$, where R is the relation $\{((pk, crs, c), (m, \sigma, open)) : \text{ver}_{\mathsf{agot}}(pk, m, \sigma) = 1 \wedge \text{open}_{\text{gs}}(crs, c, open) = 1\}$. As our signature scheme is structure-preserving, the verification equation is a pairing equation, and the opening algorithm of our commitment scheme consists of verifying a simplified GS proof, wich again is a pairing equation. Furthermore, the witness consists only of group elements. We can therefore instantiate the NIZKs by encrypting the witness to public keys defined in the CRS using Elgamal, and using a Schnorr proof to prove that the encrypted values satisfy the pairing equations. The proof will be online extractable as the simulator can decrypt the witness using its power over the CRS. The simulator could simulate the Schnorr proofs by programming the random oracle, but this would not be compatible with the strict (non-programmable) global random oracle $\bar{\mathcal{G}}_{\text{sRO}}$. Instead, we add a value $\bar{g}_{sim}$ to the CRS, and prove a disjunction that either proves knowledge of a signature on the committed message, or knowledge of the discrete logarithm of $\bar{g}_{sim}$. By the witness indistinguishability of the Schnorr proof, the simulator can now simulate without programming the random oracle by using its knowledge of $\log_g(\bar{g}_{sim})$.

We now define π_{NIZK}^{R}, a $\mathcal{F}_{\text{CRS}}^{\text{crsgen}_{\text{nizk}}}, \bar{\mathcal{G}}_{\text{sRO}}$-hybrid protocol, where $\text{crsgen}_{\text{nizk}}$ takes $(\bar{g}_s, \bar{g}_t, \bar{g}_m, \bar{g}_{o_1}, \bar{g}_{o_2}, \bar{g}_{o_3}, \bar{g}_{o_4}, \bar{g}_{sim}) \leftarrow\!\!\$\ \mathbb{G}^3 \times \tilde{\mathbb{G}}^2 \times \mathbb{G}^3$.

- On input $(\texttt{prove}, (pk, crs, c), (m, \sigma, open))$, such that $((pk, crs, c), (m, s, open)) \in R$:
  - Obtain $(\bar{g}_s, \bar{g}_t, \bar{g}_m, \bar{g}_{o_1}, \bar{g}_{o_2}, \bar{g}_{o_3}, \bar{g}_{o_4}, \bar{g}_{sim})$ from $\mathcal{F}_{\text{CRS}}^{\text{crsgen}_{\text{nizk}}}$.
  - Parse $\sigma = (r, s, t)$, $pk = (y, x)$, $par = (y_1, y_2, \tilde{g}_1, \tilde{g}_2, \tilde{g}_3, \tilde{g}_4)$, $c = (c_1, c_2, c_3, c_4, c_5)$, $open = (\tilde{o}_1, \tilde{o}_2, o_3, o_4)$.
  - Take $u' \leftarrow\!\!\$\ \mathbb{Z}_q^*$ and set $r' \leftarrow r^{u'}$.
  - Take $z_1, z_2 \leftarrow\!\!\$\ \mathbb{Z}_q$ and set $f_1 = g^{z_1}$, $f_2 \leftarrow \tilde{g}^{z_2}$, $s' \leftarrow \bar{g}_s^{z_1} s^{1/u'}$, $t' \leftarrow \bar{g}_t^{z_1} t^{1/u'}$, $m' \leftarrow \bar{g}_m^{z_1} m$, $\tilde{o}_1' \leftarrow \bar{g}_{o_1}^{z_2} \tilde{o}_1$, $\tilde{o}_2' \leftarrow \bar{g}_{o_2}^{z_2} \tilde{o}_2$, $o_3' \leftarrow \bar{g}_{o_3}^{z_1} o_3$, $o_4' \leftarrow \bar{g}_{o_4}^{z_1} o_4$.
  - Create the following generalized Schnorr proof (as formally defined by Camenisch, Kiayias, and Yung [CKY09]), using knowledge of a witness for the first expression of the disjunction.

$$
\begin{aligned}
\pi \leftarrow \mathsf{SPK}(z_1, z_2)\{ f_1 = g^{z_1} \wedge f_2 = \tilde{g}^{z_2} \ \wedge \\
e(s', r')/(e(m', y)e(x, \tilde{g})) &= (e(\bar{g}_1, r')/e(\bar{g}_3, y))^{z_1} \ \wedge \\
e(t', r')/(e(s', y)e(g, \tilde{g})) &= (e(\bar{g}_2, r')/e(\bar{g}_2, y))^{z_1} \ \wedge \\
e(g, \tilde{o}_1')/(e(c_1, \tilde{g}_1)e(o_3', \tilde{g}_2)) &= e(g, \bar{g}_{o_1})^{z_2} \cdot e(\bar{g}_{o_3}, \tilde{g}_2)^{z_1} \ \wedge \\
e(g, \tilde{o}_2')/(e(c_1, \tilde{g}_3)e(o_3', \tilde{g}_4)) &= e(g, \bar{g}_{o_1})^{z_2} \cdot e(\bar{g}_{o_3}, \tilde{g}_4)^{z_1} \ \wedge \\
e(y_1, \tilde{o}_1')/(e(c_2/m', \tilde{g}_1)e(o_4', \tilde{g}_2)) &= e(y_1, \bar{g}_{o_1})^{z_2} \cdot (e(\bar{g}_m, \tilde{g}_1)/e(\bar{g}_{o_4}, \tilde{g}_2))^{z_1} \ \wedge \\
e(y_1, \tilde{o}_2')/(e(c_2/m', \tilde{g}_3)e(o_4', \tilde{g}_4)) &= e(y_1, \bar{g}_{o_2})^{z_2} \cdot (e(\bar{g}_m, \tilde{g}_3)/e(\bar{g}_{o_4}, \tilde{g}_4))^{z_1} \\
\vee \\
\bar{g}_{sim} &= g^{z_1} \}
\end{aligned}
$$

    Output $(\texttt{done}, (f_1, f_2, r', s', t', m', \tilde{o}_1, \tilde{o}_2', o_3', o_4', \pi))$

- On input $(\texttt{verify}, (pk, par, c), \psi)$:
  - Obtain $(\bar{g}_s, \bar{g}_t, \bar{g}_m, \bar{g}_{o_1}, \bar{g}_{o_2}, \bar{g}_{o_3}, \bar{g}_{o_4}, \bar{g}_{sim})$ from $\mathcal{F}_{\text{CRS}}^{\text{crsgen}_{\text{nizk}}}$.
  - Parse $pk = (y, x)$, $par = (y_1, y_2, \tilde{g}_1, \tilde{g}_2, \tilde{g}_3, \tilde{g}_4)$, $c = (c_1, c_2, c_3, c_4, c_5)$, $\psi = (f_1, f_2, r', s', t', m', \tilde{o}_1, \tilde{o}_2', o_3', o_4', \pi)$.
  - Check that $r' \neq 1_{\tilde{\mathbb{G}}}$, check $c_4 = \bar{\mathcal{G}}_{\text{sRO}}(par, c_1, c_2, c_3, g^{c_5} \cdot c_1^{-c_4}, (y_1/y_2^{c_5} \cdot (c_2/c_3))^{-c_4})$, and verify π.
  - Output 1 if all checks pass and 0 otherwise.

Theorem 14. *Protocol π_{NIZK} EUC emulates $\mathcal{F}_{\text{NIZK}}^{R}$ in the $\mathcal{F}_{\text{CRS}}^{\text{crsgen}_{\text{nizk}}}, \bar{\mathcal{G}}_{\text{sRO}}$-hybrid model under the SXDH assumption.*

Proof. Consider simulator $\mathcal{S}$, that simulates $\mathcal{F}_{\text{CRS}}^{\text{crsgen}_{\text{nizk}}}$ by taking $(td_s, td_t, td_m, td_{o_1}, td_{o_2}, td_{o_3}, td_{o_4}, td_{sim}) \leftarrow\$ \mathbb{Z}_q$ and computing ($\bar{g}_s \leftarrow g^{td_s}$, $\bar{g}_t \leftarrow g^{td_t}$, $\bar{g}_m \leftarrow g^{td_m}$, $\bar{g}_{o_1} \leftarrow \tilde{g}^{td_{o_1}}$, $\bar{g}_{o_2} \leftarrow \tilde{g}^{td_{o_2}}$, $\bar{g}_{o_3} \leftarrow g^{td_{o_3}}$, $\bar{g}_{o_4} \leftarrow g^{td_{o_4}}$, $\bar{g}_{sim} \leftarrow g^{td_{sim}}$. It handles queries from $\mathcal{F}_{\text{NIZK}}$ as follows.

- Upon receiving $(\texttt{prove}, (pk, par, c))$ from $\mathcal{F}_{\text{NIZK}}$:
  - Parse $pk = (y, x)$, $par = (y_1, y_2, \tilde{g}_1, \tilde{g}_2, \tilde{g}_3, \tilde{g}_4)$, $c = (c_1, c_2, c_3, c_4, c_5)$.
  - Take $r' \leftarrow\$ \tilde{\mathbb{G}}^*$, $(f_1, s', t', m', o'_3, o'_4) \leftarrow\$ \mathbb{G}^6$, $(f_2, \tilde{o}_1, \tilde{o}_2) \leftarrow\$ \tilde{\mathbb{G}}^3$.
  - Construct proof π using knowledge of witness td_{sim} for the second expression of the disjunction.
  - Send $(\texttt{done}, (f_1, f_2, r', s', t', m', \tilde{o}'_1, \tilde{o}'_2, o'_3, o'_4, \pi))$ to $\mathcal{F}_{\text{NIZK}}$.
- Upon receiving $(\texttt{verify}, (pk, par, c), (f_1, f_2, r', s', t', m', \tilde{o}_1, \tilde{o}'_2, o'_3, o'_4, \pi))$ from $\mathcal{F}_{\text{NIZK}}$:
  - Let $s \leftarrow s'/f_1^{td_s}$, $t \leftarrow t'/f_1^{td_t}$, $m \leftarrow m'/f_1^{td_m}$, $\tilde{o}_1 \leftarrow \tilde{o}'_1/f_2^{td_{o_1}}$, $\tilde{o}_2 \leftarrow \tilde{o}'_2/f_2^{td_{o_2}}$, $o_3 \leftarrow o'_3/f_1^{td_{o_3}}$, $o_4 \leftarrow o'_4/f_1^{td_{o_4}}$.
  - Send $(\texttt{witness}, (m, (r', s, t), (\tilde{o}_1, \tilde{o}_2, o_3, o_4))$ to $\mathcal{F}_{\text{NIZK}}$.

To show that no environment can distinguish the real and ideal world, we must argue that proofs generated by $\mathcal{S}$ are indistinguishable from real proofs and that $\mathcal{S}$ extracts a valid witness from any valid proof. When $\mathcal{S}$ creates a proof, it takes random values instead of creating valid Elgamal ciphertexts, which is indistinguishable under the SXDH assumption. It further constructs π using a witness satisfying the second expression of the disjunction, but as the generalized Schnorr proof is perfectly witness indistinguishable, this change is not noticable. When $\mathcal{S}$ extracts from a valid proof, by the simulation soundness of the generalized Schnorr proof, we know that a satisfying witness for the statement exists. As the proof is a disjunction, it may either satisfy the first or the second expression. If the witness satisfies the first expression, $\mathcal{S}$ decrypts a valid signature (as guaranteed by the first two pairing equations of the proof), and a valid commitment opening (as guaranteed by the last four pairing equations of the proof). What remains to show is that the probability that the witness satisifies the second equation of the disjunction instead is negligible. Suppose there exists an environment that can cause this to happen with non-negligible probability, we can solve the DL problem in $\mathbb{G}$ (which is implied by SXDH). It embeds the problem instance as $\bar{g}_{sim}$ and simulates π by programming $\bar{\mathcal{G}}_{\text{sRO}}$. Whenever $\mathcal{S}$ extracts values from a proof that do not form a valid signature and opening, it extracts from the proof by rewinding, yielding the DL solution. □

6 Discussion, conclusion, and open problems

Our paper presents a new methodology for proving the security of combined protocols in a modular way. The strength of this methodology is that it applies to protocols where one scheme depends on the *code* of another scheme, which often occurs in protocols that employ zero-knowledge proofs, a case that has not been solved with modular proofs in the literature on composable security so far. We then provide efficient instantiations for the schemes that we consider, with discrete-logarithm-based primitives.

The efficiency of our protocol is practical, while apparently not yet optimal. One reason for this is the non-interactive commitment functionality $\mathcal{F}_{\text{NCOM}}$: The version $\mathcal{F}_{\text{NIC}}$ described by Camenisch et al. [CDR16] does not in itself guarantee *binding*; it guarantees that the adversary opens any commitment to only one value, but it does not guarantee that this value is fixed at the time of commitment. Camenisch et al. prove that all schemes π_{NIC} that realize $\mathcal{F}_{\text{NIC}}$ guarantee *binding* in the expected, meaningful way, but this property then remains one of π_{NIC} and not of $\mathcal{F}_{\text{NIC}}$, which is contrary to a fully modular approach in which proofs using $\mathcal{F}_{\text{NIC}}$ should not need to refer back to its realization π_{NIC}. We instead use the concept of *extraction*, formalized as $\mathcal{F}_{\text{ENIC}}$ in [CDR16], to ensure that each commitment open only to a single value. While this

formalizes *binding* in the definition of $\mathcal{F}_{\text{NCOM}}$, it has the disadvantage that must we now be actually able to *extract* from the commitment, which further complicates our protocol. Finding a meaningful definition of $\mathcal{F}_{\text{NCOM}}$ that formalizes *binding* without the necessity of *extraction* is an open question.

Furthermore, while we describe the proof of only a single protocol in this paper, we believe that the methodology will be useful in many other protocols in which the code of one scheme is used in another, whether these are based on zero-knowledge proofs, multi-party computation, verifiable computation, or other related primitives.

References

[AGOT14] Masayuki Abe, Jens Groth, Miyako Ohkubo, and Mehdi Tibouchi. Unified, minimal and selectively randomizable structure-preserving signatures. In Yehuda Lindell, editor, *Theory of Cryptography*, volume 8349 of *LNCS*, pages 688–712. Springer, 2014.

[BMT18] Christian Badertscher, Ueli Maurer, and Björn Tackmann. On composable security for digital signatures. In Michel Abdalla and Ricardo Dahab, editors, *Public-Key Cryptography*, volume 10769 of *LNCS*, pages 494–523. Springer, 2018.

[BPW04] Michael Backes, Birgit Pfitzmann, and Michael Waidner. A general composition theorem for secure reactive systems. In Moni Naor, editor, *Theory of Cryptography*, volume 2951 of *LNCS*, pages 336–354. Springer, 2004.

[Can00] Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13:143–202, April 2000.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Foundations of Computer Science*, 2001.

[Can04] Ran Canetti. Universally composable signature, certification, and authentication. In *17th IEEE Computer Security Foundations Workshop*, 2004.

[Can05] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Report 2000/067, December 2005.

[Can18] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Report 2000/067, December 2018.

[CCS09] Jan Camenisch, Nishanth Chandran, and Victor Shoup. A public key encryption scheme secure against key dependent chosen plaintext and adaptive chosen ciphertext attacks. In Antoine Joux, editor, *Advances in Cryptology — EUROCRYPT*, volume 5479 of *LNCS*, pages 351–368. Springer, 2009.

[CDG+18] Jan Camenisch, Manu Drijvers, Tommaso Gagliardoni, Anja Lehmann, and Gregory Neven. The wonderful world of global random oracles. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology — EUROCRYPT*, volume 10820 of *LNCS*, pages 280–312. Springer, 2018.

[CDPW07] Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. Universally composable security with global setup. In Salil P. Vadhan, editor, *Theory of Cryptography*, volume 4392 of *LNCS*, pages 61–85. Springer, 2007.

[CDR16] Jan Camenisch, Maria Dubovitskaya, and Alfredo Rial. UC commitments for modular protocol design and applications to revocation and attribute tokens. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology — CRYPTO*, volume 9816 of *LNCS*, pages 208–239. Springer, 2016.

[CEK+16] Jan Camenisch, Robert R. Enderlein, Stephan Krenn, Ralf Küsters, and Daniel Rausch. Universal composition with responsive environments. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology – ASIACRYPT*, volume 10032 of *LNCS*, pages 807–840. Springer, 2016.

[CF01] Ran Canetti and Marc Fischlin. Universally composable commitments. In Joe Kilian, editor, *Advances in Cryptology — CRYPTO*, volume 2139 of *LNCS*, pages 19–40. Springer, 2001.

[CKS11] Jan Camenisch, Stefan Krenn, and Victor Shoup. A framework for practical universally composable zero-knowledge protocols. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology – ASIACRYPT*, volume 7073 of *LNCS*, pages 449–467. Springer, 2011.

[CKY09] Jan Camenisch, Aggelos Kiayias, and Moti Yung. On the portability of generalized schnorr proofs. In Antoine Joux, editor, *Advances in Cryptology - EUROCRYPT 2009, 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings*, volume 5479 of *Lecture Notes in Computer Science*, pages 425–442. Springer, 2009.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *Symposium on the Theory of Computer Science*, 2002.

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

[DM00] Yevgeniy Dodis and Silvio Micali. Parallel reducibility for information-theoretically secure computation. In Mihir Bellare, editor, *Advances in Cryptology — CRYPTO*, volume 1880 of *LNCS*, pages 74–92. Springer, 2000.

[GMR88] Shafi Goldwasser, Silvio Micali, and Ron Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *Siam Journal of Computing*, 17(2):281–308, April 1988.

[GOS12] Jens Groth, Rafail Ostrovsky, and Amit Sahai. New techniques for noninteractive zero-knowledge. *Journal of the ACM*, 59(3), June 2012.

[GS08] Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In Nigel Smart, editor, *Advances in Cryptology — EUROCRYPT*, volume 4965 of *LNCS*, pages 415–432. Springer, 2008.

[HS15] Dennis Hofheinz and Victor Shoup. GNUC: a new universal composability framework. *Journal of Cryptology*, 28(3):423–508, July 2015.

[KT13] Ralf Küsters and Max Tuengerthal. Joint state composition theorems for public-key encryption and digital signature functionalities with local computation. Cryptology ePrint Report 2008/006, August 2013.

[KTR18] Ralf Küsters, Max Tuengerthal, and Daniel Rausch. The IITM model: a simple and expressive model for universal composability. Cryptology ePrint Report 2013/025, December 2018.

[MR11] Ueli Maurer and Renato Renner. Abstract cryptography. In *Innovations in Computer Science*, 2011.

[NY90] Moni Naor and Moti Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In Harriet Ortiz, editor, *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*, pages 427–437. ACM, 1990.

[PW01] Birgit Pfitzmann and Michael Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *IEEE Symposium on Security and Privacy*, pages 184–200, 2001.

A Recursive composition for UC and GUC

We state here recursive composition theorems for UC. We remark that the composition theorem using the composition operation $\text{UC}(\cdot,\cdot,\cdot)$ as specified in [Can18] only applies to protocols ρ which call its sub-protocol ϕ that is to be instantiated by π only from the main parties of ρ, but not from any of its subsidiaries. Consider the following example: Let ρ be a protocol that uses ϕ as a subroutine, and that also has another subroutine ψ using the same instance of ϕ as a subroutine. (This is a priori allowed in UC, since such a protocol ϕ can still be subroutine respecting and ρ can still be compliant.) Now applying the composition operation $\text{UC}(\rho,\phi,\pi)$ replaces the calls to ϕ in ρ by calls to π. It does, however, not apply to ψ. As a result, protocol $\rho^{\phi\to\pi}$ uses subroutine π, whereas its subroutine ψ invokes its subroutine ϕ with the same identity that π uses! This will obviously fail.

We therefore state the following version of the UC composition theorem with recursive composition, which applies to all protocols ρ that are compliant in the sense of [Can18], even if they share the same instances of subroutines with code ϕ with any other of their subroutines.

Theorem 15. *Let ρ,π,ϕ be PPT protocols such that π UC-emulates ϕ and both ϕ and π are subroutine respecting, and ρ is compliant. Then protocol $\rho^{\phi\to\pi}$ UC-emulates protocol ρ.*

There is actually no need to modify the proof of the composition theorem in [Can18], with the recursive type of composition, the proof also applies to general protocols (still due to the restrictions stated in that paper). We initially planned to also provide such a theorem for the GUC framework in [CDPW07], however, as the model definition is ambiguous and the formal notion of GUC-realization is uninstantiable for non-trivial protocols, we refrained from making a formal statement. We conjecture that for a formally consistent version of the GUC framework, recursive composition would also apply.