A Framework for Practical Universally Composable Zero-Knowledge Protocols*

Jan Camenisch[1], Stephan Krenn[2], and Victor Shoup[3]

[1] IBM Research – Zurich, Rüschlikon, Switzerland
jca@zurich.ibm.com
[2] Bern University of Applied Sciences, Biel-Bienne, Switzerland, and University of Fribourg, Switzerland
stephan.krenn@bfh.ch
[3] Department of Computer Science, New York University, USA
shoup@cs.nyu.edu

Abstract. Zero-knowledge proofs of knowledge (ZK-PoK) for discrete logarithms and related problems are indispensable for practical cryptographic protocols. At *Eurocrypt 2009*, Camenisch, Kiayias, and Yung provided a specification language (the *CKY-language*) for such protocols, which allows one to modularly design and analyze cryptographic protocols: protocol designers just need to specify the statement they want to prove in zero-knowledge and are ensured that an efficient proof protocol exists and indeed proves the specified statement, provided that the specification was in the CKY-language.
However, as specifications in the CKY-language are realized by so-called Σ-protocols, the resulting protocols only satisfy the classical notion of zero-knowledge proofs of knowledge, which *not* retained if they are composed with themselves or with other protocols, e.g., when used as building blocks for higher-level applications. This problem can be tackled by moving to the Universal Composability (UC) framework, which guarantees retention of security when composing protocols and, in particular, when using them as building blocks in arbitrary contexts. While there exists generic transformations from Σ-protocols to protocols that are secure under this stronger security notion, these transformation are often not efficient enough for the design of practical protocols.
In this paper we are aiming for practically efficient ZK-PoK in the UC-framework by introducing a specification language akin to the CKY-language and a compiler such that protocols specified in our language are UC-secure and efficient. To this end we propose an extension of the UC-framework addressing the problem that UC-secure zero-knowledge proofs are always proofs *of knowledge*, and state a special composition theorem which allows one to use the weaker – but more efficient and often sufficient – notion of proofs *of existence* in the UC-framework for the first time. We believe that our contributions enable the design of practical protocols that are UC-secure and thus themselves can be used as building blocks.
Keywords: Universal Composability; Protocol Design; Zero-Knowledge; Proof of Knowledge;

* This work was in part funded by the Swiss Hasler Foundation, the EU FP7 grants 216483 and 216499, as well as by the NSF grant CNS-0716690.

1 Introduction

The probably most demanding task when designing a practical cryptographic protocol is to define its security properties and then to prove that it indeed satisfies them. For this security analysis it is often assumed that the "world" consists only of one instance of the protocol and only of the involved parties, rather than of many parties running many instances of the same protocol as well as other protocols at the same time. While this approach allows for a relatively simple analysis of protocols, it does not properly model reality and therefore provides little if any security guarantees. Also, this approach does not allow for a modular usage of the protocols, i.e., when a protocol is used as a building block for another protocol, the security analysis must be all done from scratch.

To address these problems, a number of frameworks have been proposed over the years, e.g., [3,17,58]. The so-called *Universal Composability* (UC) framework by Canetti [17] seems to be the one that is most prevalent. The probably most fundamental result in this model is its very strong composition theorem: once a protocol is proved secure in this model, it can be used in arbitrary contexts retaining its security properties. This allows one to split a protocol into smaller subroutines so that the security analysis becomes easier as each subprotocol can be analyzed separately. In particular, each protocol only has to be analyzed once and for all, and the analysis does not have to be repeated for each specific context.

This modularity and the high security guarantees suggest that protocols should always be designed and proven secure in the UC-framework rather than in an ad-hoc way. However, only a fraction of the protocols found in the literature is proved UC-secure, examples including oblivious transfer [57] and encryption- [40,41,52], signature- [1,39,70] and commitment schemes [45]. Furthermore, only very few UC-secure protocols are actually deployed in the real world, e.g., [19,38]. We believe that one main reason for this is the high computational overhead coming along with most UC-secure protocols compared to those which have not been proved secure in the UC-framework.

When designing practical cryptographic protocols, efficient *zero-knowledge proofs of knowledge* (ZK-PoK) for discrete logarithms and related problems have turned out to be indispensable. On a high level, these are two party protocols between a prover and a verifier which allow the former to convince the latter that it possesses some secret piece of information, without the verifier being able to learn anything about it. This allows protocol designers to enforce a protocol participant to assure other parties that its actions are consistent with its internal knowledge state. The shorthand notation for such proofs, introduced in [16], has been extensively employed in the past and contributed to the wide employment of these protocols in cryptographic design. This notation suggested using, e.g., $\mathsf{PK}\left[(\alpha) : y = g^{\alpha}\right]$ to denote a proof of the discrete logarithm $\alpha = \log_g y$, and it appeared in many works to describe quite complex discrete logarithm based relations, e.g., [2,5,7,8,9,10,12,15,31,34,42,47,48,50,51,64,65,66,67,68,69]. This informal notion was recently formalized and refined by Camenisch, Kiayias and Yung who have provided a specification language (*CKY-language*) for such protocols [13]. The language allows for the modular design and analysis of cryptographic protocols: protocol designers just needs to specify the statement the ZK-PoK shall prove and, if the specification is in the CKY-language, they are ensured that the proof protocol exists and indeed proves the specified statement. Their realizations are based on so-called Σ-protocols, which satisfy the classical notion of ZK-PoK. Unfortunately, these protocols are not UC-secure and thus cannot be used for protocol design in the UC-framework. While generic transformations from Σ-protocols to UC-ZK protocols are known [33], they come along with a significant computational overhead, making the resulting protocols impracticable for real-world usage.

Our Contributions. In this paper we aim at closing the gap between high security guarantees and practical usability and efficiency for ZK-PoK. To this end, we first present an exhaustive language and a compiler which allow protocol designers to efficiently and modularly specify and obtain UC-ZK protocols. We then give an extension of the UC-framework allowing protocol designers to also make usage of the more efficient proofs of *existence* (as opposed to proofs of *knowledge*) for the first time, which we also incorporate into our language. Let us explain this in more detail.

A language for UC-ZK protocols. We provide an intuitive language for specifying ZK-PoK for discrete logarithms akin to the CKY-language [13] where the specification also allows one to assess the complexity of the specified protocol. We then provide a compiler which translates these specifications into concrete protocols. Even though this compiler is mainly based on existing techniques, it offers unified and unambiguous interfaces and semantics for the associated protocols and functionalities for the first time. It thus enables protocol designers to treat specifications in our language as black-boxes, while having clearly defined security guarantees.

Proving existence rather than knowledge. In the UC-framework, all proof protocols are necessarily proofs of *knowledge*. This is, because the prover must provide the witness of the statement to be proved to the UC-ZK ideal functionality as otherwise the functionality would typically not be able to verify the correctness of a statement in polynomial time. However, when designing higher-level protocols, is often only necessary to prove that some computation was done correctly, but not to show that the secret quantities are actually 'known'. This difference can be exploited to obtain much more efficient protocols: in the security proofs for proofs of knowledge the simulator needs to be able to extract the witness from the adversary – a requirement which is computationally expensive to achieve in the UC-framework, and which is not needed for a proof that a protocol step was done correctly.

To allow protocol designer to also make use of these more efficient protocols (which are not proofs *of knowledge* any more), we extend our language and provide the necessary framework to prove UC-security. Loosely speaking, we therefore formulate the so-called *gullible ZK ideal functionality* $\mathcal{F}_{\text{gZK}}$, and provide a special composition theorem which allows protocol designers to use existence-proofs "as if they were ideal functionalities", if they are later instantiated as described in our compiler. On a high level the theorem states that proving the correctness of a protocol using $\mathcal{F}_{\text{gZK}}$ in a slightly non-UC-compliant way is sufficient for the protocol where $\mathcal{F}_{\text{gZK}}$ is instantiated as described in our compiler to be UC-secure in the standard sense.

Related Work. The UC-framework has first been introduced by Canetti [17]. The notion of Ω-protocols was introduced in [32,33], and so far the most efficient UC-secure zero-knowledge proofs of knowledge have been proposed in [49]. Further, [28] analyzes UC-ZK in the presence of global setup [18]. The idea of committed proofs was first mentioned in [37]. We combine the techniques of [37,49] to compile proof specifications in our language to real protocols. In particular this allows us to realize proofs *of existence*.

A language for specifying discrete logarithm based ZK-PoK was presented by Camenisch and Stadler in [16] and later refined in [13], but neither of their realizations are UC-secure. Nevertheless, our notation is strongly inspired by theirs. In fact, our language has already turned out to be very useful to describe ZK-PoK in a companion paper[11], and in this paper we fulfill the promises given therein.

Functionalities similar to $\mathcal{F}_{\text{gZK}}$ have already been used by Lindell [43,44] and Pass and Rosen [55] in different contexts. That is, all this work is on two-party protocols which preserve their security

guarantees under bounded-concurrent self-composition and not on full UC-security. Prabhakaran and Sahai [59,60] also suggest generalizations of the UC-framework in which functionalities can be realized that cannot be realized in the plain UC-framework. Their work differs from ours in that they leave the standard model of polynomial time computation by granting the adversary access to some super-polynomially powerful oracle ("imaginary angel"), while our approach works in the standard computational model. Furthermore, they suggest generic solutions for ZK-PoK while we are aiming at practically efficient protocols.

Roadmap. After introducing some notation, recapitulating fundamental theory and presenting two running examples in §2, we describe a basic language for specifying UC-secure ZK-PoK protocols in detail in §3. In §4 we show how proofs of existence rather than knowledge can be UC-realized, resulting in much more efficient protocols, and extend our language accordingly. We show how such specifications can be compiled to actual protocols in §5. Finally, in §6 we give several extensions to our basic language.

2 Preliminaries

Let us introduce some notation first. By $s \in_R \mathcal{S}$ we denote the uniform random choice of some element s in set $\mathcal{S}$. For bitstrings n and m, the bitwise XOR is written as $n \oplus m$. The group of signed quadratic residues [36] for some modulus n is denoted by $\mathbb{SR}_n$. We recap some properties of this group in §A.3. For two random ensembles, $\overset{s}{\approx}$ denotes statistical indistinguishability. Finally, two party protocols between parties P and V with common input y and private input w to P are written as $(\mathsf{P}(w), \mathsf{V})(y)$.

We assume that the reader is familiar with the notion of Σ- and Ω-protocols, and only give informal definitions here. A protocol $(\mathsf{P}(w), \mathsf{V})(y)$ is called a *Σ-protocol* [22], if it is an honest verifier ZK-PoK in the non-UC model, consisting of three messages being exchanged (a *commitment* t, a *challenge* $c \in_R \mathcal{C} = \{0,1\}^k$, and a *response* r), such that the secret w can be computed from any two valid protocol transcripts with the same commitment but different challenges. A protocol is called an *Ω-protocol* [33], if it further takes a common reference string σ as additional input, such that knowing a trapdoor to σ it is possible to compute the prover's secret input from any successful run of the protocol. For formal definitions we refer to §A.1.

We further say that an Ω-protocol *f-extractable*, if it is not possible to compute w from any successful run, but only $f(w)$ for some function f. In particular, we will make use of two types of f-extractable protocols: one the one hand we will use $f(w_1, \ldots, w_n) = (w_1, \ldots, w_k)$ for some $k \leq n$, i.e., protocols which only allow to extract parts of the witness. On the other hand, we will have $f(w_1, \ldots, w_n) = (w_1, \ldots, w_n, \mathsf{A}(w_1, \ldots, w_n))$, i.e., functions f which in addition to all witnesses additionally output some additional values depending on these witnesses. These constructions will allow for an efficiency speedup compared to using plain Ω-protocols, while still ensuring appropriate security guarantees for many applications.

2.1 The UC-Model

We next want to briefly recapitulate the Universal Composability (UC) framework [17].

By a *party* we refer to a probabilistic polynomial time (PPT) interactive Turing machine. Each party P is uniquely determined by a pair $(\mathtt{PID}_P, \mathtt{SID}_P)$, where $\mathtt{PID}_P$ and $\mathtt{SID}_P$ are its *party ID* and its *session ID*, respectively. By convention, two parties share the same session ID if and only if

they are participants of the same instance of a protocol. The only purpose of party IDs is to be able to distinguish between different participants of the same protocol instance. Following [11], we assume that session IDs are structured as pathnames. That is, for a protocol with session ID `SID`, the session ID of any of its subprotocols is given by `SID/subsession`, where `subsession` is a unique local identifier, containing the party IDs of all participating parties and shared public parameters.

The main concept of the UC framework is that of UC-emulation. Loosely speaking, a protocol ρ *UC-emulates* some protocol ϕ, if ρ does not affect the security of anything else than ϕ would have, no matter how many other instances of ρ or arbitrary other protocols are executed concurrently. This implies that ρ can safely be used on behalf of ϕ without compromising security. The most interesting case is where ϕ is some *ideal functionality* $\mathcal{F}$, which can be thought of as an incorruptible trusted party that takes inputs from all parties, locally computes a certain cryptographic task, and hands back outputs to each participating party. Ideal functionalities are secure by definition, and can therefore be seen as "formal specifications" of cryptographic tasks. Now, if ρ UC-emulates $\mathcal{F}$, one can infer that ρ does not leak any information to an adversary than $\mathcal{F}$ would have, and therefore securely realizes the task at hand in arbitrary contexts. For a more precise description we refer to [17] and §A.2.

A protocol ρ may itself make use of some ideal functionality $\mathcal{F}$ as a subroutine. In this case, ρ is called an $\mathcal{F}$-*hybrid* protocol. If not stated otherwise, all protocols we are going to present are $\mathcal{F}_{\text{ach}}$-hybrid protocols, where $\mathcal{F}_{\text{ach}}$ is an ideal functionality realizing authenticated (but not necessarily private) channels. The functionality takes as input a message x from some sender, which it simply forwards to a receiver. The adversary $\mathcal{A}$ learns the message x, and is allowed to change its content before it is delivered to the receiver, whenever the sender is corrupted. See §A.2 for a formal description of $\mathcal{F}_{\text{ach}}$.

The corruption model underlying our discussion is *adaptive corruptions with erasures*. This can be seen as a bit of a compromise: while only considering static corruptions would not properly reflect reality, assuming secure data erasures is necessary to obtain efficient protocols in this setting. However, even if implementing erasures might be difficult, it is not impossible.

The Basic UC-ZK Ideal Functionality. In the following we discuss the basic ideal zero-knowledge functionality, which is formally specified in Figure 1. It is parametrized by two binary relations, R, R', which have the following meaning: the relation R specifies the set of inputs (y, w) the functionality accepts from an honest prover. For such inputs, the functionality informs the verifier that the prover knows a witness for y, while an adversary does not learn w. Yet, if the prover is corrupted, it is allowed to supply inputs from a binary relation $R' \supseteq R$, in which case the ZK property does not have to be satisfied any more.

The relation R might itself be parametrized by system parameters, specifying, e.g., the concrete groups being used. We will model all such parameters as public coin parameters, i.e., the environment might know the random coins being used to generate the system parameters. This is helpful if the same parameters are used in other protocols as well, e.g., to sign messages.

The functionality defined in Figure 1 differs from the standard one found in the literature in two ways. Firstly, we delay revealing the claimed statement y to V and $\mathcal{A}$ until the last possible moment, and only give $\ell(y)$ to the adversary in the first step, where ℓ is a leakage function, which roughly gives some information about the "size and shape" of y to $\mathcal{A}$ (to be precise, $\ell()$ is a parameter of $\mathcal{F}_{\text{ZK}}$ as well which will be disregarded in the remainder of our discussion). This approach prevents the simulator from being over-committed in our constructions, and to the best of our knowledge $\mathcal{F}_{\text{ZK}}^{R,R'}$ can safely be used instead of the standard UC-ZK functionality in any application. In §C

The zero-knowledge functionality $\mathcal{F}_{\mathrm{ZK}}^{R,R'}$

1. Wait for an input $(\texttt{prove}, y, w)$ from P such that $(y, w) \in R$ if P is honest, or $(y, w) \in R'$ if P is corrupt. Send $(\texttt{prove}, \ell(y))$ to $\mathcal{A}$. Further wait for a message `ready` from V, and send `ready` to $\mathcal{A}$.
2. Wait for a message `lock` from $\mathcal{A}$.
3. Upon receiving a message `done` from $\mathcal{A}$, send `done` to P. Further wait for an input `proof` from $\mathcal{A}$ and send $(\texttt{proof}, y)$ to V.

Corruption rules:

- ▷ If P gets corrupted after sending $(\texttt{prove}, y, w)$ and before Step 2, $\mathcal{A}$ is given (y, w) and is allowed to change this value to any value $(y', w') \in R'$ at any time before Step 2.

Fig. 1: The basic zero-knowledge functionality $\mathcal{F}_{\mathrm{ZK}}^{R,R'}$, parametrized by two binary relations R, R' such that $R' \supseteq R$ [11].

we give an illustrative example of an efficient protocol which can be proved to be UC-secure using our formalization, but not using the standard one. Secondly, we allow corrupt parties to supply witnesses from a larger set than honest parties. This relaxation stems from the *soundness gap* of most known efficient constructions for ZK-PoK for discrete logarithms in the non-UC case [27] (which are underlying the constructions for UC-ZK protocols): there, the verifier can only infer that the prover knows a witness w such that $(y, w) \in R'$, whereas an honest prover is ensured that for $(y, w) \in R$ the verifier cannot learn the secret. We will discuss the precise relation between R and R' later in §5 and §B. The same formalization of the ZK functionality was also used in [11].

2.2 Running Examples

We next introduce two running examples, which we are going to use throughout the discussion to illustrate our techniques.

Example 2.1 (Running Example 1). Let be given an integer commitment $y \in \mathbb{SR}_n$ for some safe RSA modulus n. Let further be given two generators g, h of $\mathbb{SR}_n$. In this example, a prover is interested in proving knowledge of integers w, r such that $y = g^{\omega} h^{\rho}$ and $\omega \geq 0$. □

Numerous practically relevant applications require such proof goals as basic building blocks for more complex protocols, e.g., [8,12].

Example 2.2 (Running Example 2). Let be given a cyclic group $\mathcal{H}$ of prime order q, and two generators, g, h of $\mathcal{H}$. Let further be given a triple $(u_1, u_2, e) \in \mathcal{H}^3$, and let one be interested in proving that (u_1, u_2, e) is a valid encryption of $g^{\alpha} \in \mathcal{H}$ for some $\alpha \in \mathbb{Z}_q$ known to the prover under the semantically secure version of the Cramer-Shoup cryptosystem [25,37]. That is, the task is to prove that (u_1, u_2, e) is of the form $(g^{\rho}, h^{\rho}, g^{\alpha} c^{\rho})$ for a publicly known $c \in \mathcal{H}$. □

This example stems from [11], where such proofs are repeatedly needed in the context of credential-authenticated key-exchange and identification protocols.

3 A Language For Specifying UC-ZK Protocols

As shown in [20], any ideal functionality can be UC-realized given only functionalities realizing commitments and ZK proofs, respectively. This result suggests that ZK proofs are important building

blocks of higher-level applications, and will thus often be deployed when UC-realizing cryptographic tasks.

Taking this as a motivation, we describe an intuitive language for specifying universally composable zero-knowledge protocols. The language is strongly inspired by the standard notation for describing zero-knowledge proofs of knowledge in the non-UC case which was introduced in [16]. We stress that similar to there, our notation does not only specify proof goals (i.e., what one wants to prove), but concrete protocols. That is, together with the techniques presented in §5 and §B, each statement in our language describes a unique protocol. Especially for our results given in §4, this unambiguity is important.

We start by describing a basic language, which allows one to specify arbitrary Boolean combinations of protocols proving knowledge of discrete logarithms (or representations) in arbitrary groups. In many cases the complexity of the resulting protocol can be inferred directly from the proof specification.

A protocol proving knowledge of integers $\omega_1, \ldots, \omega_n$ satisfying a *predicate* $\phi(\omega_1, \ldots, \omega_n)$ is denotes as follows:

$$\text{ZK}\left\{ \omega_1 \in \mathcal{I}^*(m_{\omega_1}), \ldots, \omega_n \in \mathcal{I}^*(m_{\omega_n}) \,:\, \phi(\omega_1, \ldots, \omega_n) \right\}. \tag{1}$$

Here, each witness ω_i belongs to some integer domain $\mathcal{I}^*(m_{\omega_i})$. The predicate $\phi(\omega_1, \ldots, \omega_n)$ is a Boolean formula containing ANDs ($\wedge$) and ORs ($\vee$), built from *atomic predicates* of the following form:

$$y = \prod_{i=1}^{u} g_i^{F_i(\omega_1, \ldots, \omega_n)}.$$

The g_i and y are elements of some commutative group, and each $F_i \in \mathbb{Z}[X_1, \ldots, X_n]$ is an integer polynomial. Similar to [16], we make the convention that values of which knowledge has to be proved are denoted by Greek letters, whereas all other quantities are assumed to be publicly known.

In the following we want to discuss the single components of our basic language in more detail.

Groups. Different atomic predicates may use different groups. Besides efficiently evaluable group operations we only require that group elements are efficiently recognizable, and that the group order does not contain any small prime divisors, where "small" can be seen as an implementation dependent parameter which typically will have $160 - 256$ bits. In particular, we do not make any intractability assumption for the groups. However, if the group order is unknown, this can be exploited for efficiency gains, as we will discuss in §6.4.

We stress that the group of quadratic residues modulo a safe RSA modulus n (i.e., $n = pq$, where $p, q, \frac{p-1}{2}, \frac{q-1}{2}$ are prime, denoted by $\mathbb{QR}_n$) does not satisfy the above requirements, as group membership cannot be efficiently verified. We recommend using the group of *signed* quadratic residues, recapitulated in §A.3, instead. However, for the case that using the quadratic residues cannot be avoided, we will show how the requirement of no small divisors can be dropped, if the soundness guarantees may be loosened slightly in §6.3.

Predicates. We allow predicates to be arbitrary combinations of atomic predicates by the Boolean connectives AND and OR. Also, witnesses may be reused across different atomic predicates. However, this may introduce some additional overhead in the case of Boolean ORs as discussed in §5.

Domains. In general, we allow the secret values $\omega_1, \ldots, \omega_n$ to be arbitrary integers. However, for implementation issues, it is necessary to specify an integer m_{ω_i} such that

$$\omega_i \in \mathcal{I}(m_{\omega_i}) := \{l \in \mathbb{Z} \,:\, -m_{\omega_i} \leq l \leq m_{\omega_i}\}$$

for each ω_i. The value of m_{ω_i} (and therefore the size of the interval) can be chosen arbitrarily large, and is only needed for the resulting protocol to be statistically zero-knowledge for any $\omega_i \in \mathcal{I}(m_{\omega_i})$. The protocols resulting from the constructions given in §5 will then guarantee that the prover knows witnesses in a slightly larger interval, i.e., they prove knowledge of witnesses ω_i^* satisfying

$$\omega_i^* \in \mathcal{I}^*(m_{\omega_i}) := \{l \in \mathbb{Z} \,:\, -tm_{\omega_i} \leq l \leq tm_{\omega_i}\},$$

where t is an implementation dependent parameter, which usually will have about $160 - 256$ bits and which is independent of the groups used in the predicate. In particular, $\mathcal{I}^*(m_{\omega_i})$ is thus uniquely defined even if ω_i is used across different atomic predicates. More precisely, we roughly have $t \approx 2^{k+l}$, where 2^{-k} is the success probability of a malicious prover, and l is a security parameter controlling the tightness of the statistical ZK property of the protocol.

Formally, the gap between $\mathcal{I}(m_{\omega_i})$ and $\mathcal{I}^*(m_{\omega_i})$ is achieved by allowing corrupt provers to hand in values satisfying a relation $R' \supseteq R$ to the ideal functionality, whereas honest parties have to supply values in R, cf. §2.1.

As a special case, we allow to define $\mathcal{I}^*(m_{\omega_i}) = \mathcal{I}(m_{\omega_i}) = \mathbb{Z}_q$, if (i) the secret ω_i only occurs in atomic predicates for which the order of the group is known, and (ii) the integer q is a common multiple of all these group orders.

Induced Relation. Each proof specification `spec` of the form (1) induces two binary relations, $R = R(\texttt{spec}) \subseteq R'(\texttt{spec}) = R'$, and a protocol $\pi = \pi(\texttt{spec})$ the compilation of which will be discussed in §5. The protocol π will be proved to UC-emulate $\mathcal{F}_{\mathrm{ZK}}^{R,R'}$, i.e., for inputs $(y, w) \in R$ the protocol ensures the prover to not leak any information about w to the verifier, while the verifier is guaranteed that the prover supplied an input $(y', w') \in R'$.

Let us now illustrate our basic language by means of our two running examples.

Example 3.1 (Running Example 1). As our basic language defined above does not allow one to specify algebraic properties of witnesses directly, we first have to resolve the condition $\omega \geq 0$ to the form (1). We do this by applying the technique proposed in [46]. That is, we prove that ω can be written as $\omega = \sum_{i=1}^{4} \chi_i^2$ for some $\chi_i \in \mathbb{Z}$. This relation then can easily be rewritten as an expression in our basic language.

As discussed above, it is necessary to know an interval ω lies in. Let it be given by $[-T, T]$, i.e., $m_\omega = T$. Then, clearly, we have that $m_{\chi_i} = \lfloor \sqrt{T} \rfloor$ for $i = 1, \ldots, 4$. Also, for y to be blinding, we can assume that $m_\rho = \lfloor n/4 \rfloor$.

The proof goal is thus given by:

$$\mathsf{N}\, \rho \in \mathcal{I}^*(\lfloor n/4 \rfloor), \{\chi_i\}_{i=1}^4 \in \mathcal{I}^*(\lfloor \sqrt{T} \rfloor), : y = g^{\chi_1^2+\chi_2^2+\chi_3^2+\chi_4^2} h^\rho \,. \qquad \square$$

Example 3.2 (Running Example 2). In this case, all secret values, i.e., α and ρ, are elements of $\mathbb{Z}_q$, where q is the order of $\mathcal{H}$. We therefore can specify the protocol by the following expression

$$\mathsf{N}\, \alpha, \rho \in \mathbb{Z}_q \,:\, u_1 = g^\rho \,\wedge\, u_2 = h^\rho \,\wedge\, e = g^\rho c^\alpha \,.$$

In particular note that the requirement that $\operatorname{ord} \mathcal{H}$ does not have small prime divisors is satisfied as q was assumed to be prime, cf. Example 2.1. $\square$

4 Proving Existence Rather Than Knowledge

As we will see in detail in the next section, realizing proofs of knowledge in the UC-framework is a computationally expensive task. On a high level this is because the simulator needs to be able to extract the secret witness without rewinding, and the most efficient currently known way to achieve this is to include a Paillier encryption of the witness into the proof. Now, in larger protocols, zero-knowledge proof protocols are often only used to ensure that a computation was done correctly, and indeed the simulator does not require the witnesses in the security proof. This is for instance the case in Example 2.2, where proving the existence of a ρ is sufficient to imply the required wellformedness of the ciphertext.

Thus, what would often be sufficient is a functionality realizing the following steps:

1. Wait for an input $(\texttt{prove}, y, w)$ from P such that there is a $\tilde{w}$ satisfying $(y, \tilde{w}) \in R$ and $f(\tilde{w}) = w$, and send $(\texttt{prove}, \ell(y))$ to $\mathcal{A}$. Further wait for a message `ready` from V, and send `ready` to $\mathcal{A}$.
2. Wait for a message `lock` from $\mathcal{A}$.
3. Upon receiving a message `done` from $\mathcal{A}$, send `done` to P. Further wait for an input `proof` from $\mathcal{A}$ and send $(\texttt{proof}, y)$ to V.

That is, one is aiming for a functionality which checks whether the prover knows some (partial) information $\tilde{w} = f(w)$ for a full witness w, and informs the verifier if this is the case. However, the problem is that by definition any zero-knowledge proof in the UC-Framework is *always* a proof of knowledge. This is, because in general the existence of $\tilde{w}$ cannot be checked efficiently, and thus the witness has to be given as an input for the functionality to be able to check whether or not the statement is true. In this section we give a framework for how proof protocols that are not proofs of knowledge but only proofs of existence can still be used to construct UC-secure protocols and be exploited for a considerable efficiency gain.

We first extend our basic language by the additional $\exists$-quantifier. Its semantics is that for secrets quantified under $\exists$ (instead of $\between$) only their existence (instead of their knowledge) is proved to the verifier. Having said this, a generalized specification of a proof goal looks as follows:

$$\between \{\omega_i \in \mathcal{I}^*(m_{\omega_i})\}_{i=1}^{n} : \exists \left\{\chi_j \in \mathcal{I}^*(m_{\chi_j})\right\}_{j=1}^{m} : \phi(\omega_1, \ldots, \omega_n, \chi_1, \ldots, \chi_m) \tag{2}$$

The computational costs of values quantified by $\exists$ are the same as for ZK-PoK in the non-UC setting, as they are compiled to standard Σ-protocols. In particular the computationally expensive Paillier encryption [54] of these values can be avoided. We will detail the compilation in §5.

Before stating our composition theorem, we have to introduce some definitions.

The *gullible zero-knowledge functionality* $\mathcal{F}_{\text{gZK}}^{R,R}$ expects the prover to supply an image y and a pair (w, x) as inputs, and always informs the verifier that $(y, (w, x)) \in R'$, no matter whether this is the case or not, cf. Figure 2. For an honest prover, w will be the part of the witness for which knowledge has to be proved, whereas x is the part for which only existence has to be proved. Upon corruption of the prover, the adversary only learns y and w, but not x. This is to model the intuitive goal of proofs of existence more appropriately. However, it therefore is important that submitting x to $\mathcal{F}_{\text{gZK}}$ and securely erasing it is possible in one step, i.e., the prover cannot be corrupted between submission and erasure of x.

Our special composition theorem guarantees that $\rho^{\pi/\mathcal{F}_{\text{gZK}}^{R,R'}}$ UC-emulates some other protocol ϕ, if ρ UC-emulates ϕ with respect to a certain type of environments, called *nice environments*, which we define next. On a high level, these are environments which (almost) never ask the dummy adversary to send incorrect inputs to the weak zero-knowledge functionality:

The gullible zero-knowledge functionality $\mathcal{F}_{\text{gZK}}^{R,R'}$

1. Wait for an input $(\texttt{prove}, y, (w, x))$ from P and send $(\texttt{prove}, \ell(y))$ to $\mathcal{A}$. Further wait for a message `ready` from V, and send `ready` to $\mathcal{A}$.
2. Wait for a message `lock` from $\mathcal{A}$.
3. Upon receiving a message `done` from $\mathcal{A}$, send `done` to P. Further wait for an input `proof` from $\mathcal{A}$ and send $(\texttt{proof}, y)$ to V.

Corruption rules:

- ▷ If P gets corrupted after sending $(\texttt{prove}, y, (w, x))$ and before Step 2, $\mathcal{A}$ is given $(y, (w, \perp))$ and is allowed to change this value at any time before Step 2.

Fig. 2: The gullible zero-knowledge functionality $\mathcal{F}_{\text{gZK}}^{R,R'}$ always informs the verifier that the proof was correct.

Definition 4.1. *Let $\mathcal{A}^*$ denote the dummy adversary attacking some $\mathcal{F}_{\text{gZK}}^{R,R'}$-hybrid protocol. We call an environment $\mathcal{Z}$* nice*, if the statements it requires $\mathcal{A}^*$ to send to $\mathcal{F}_{\text{gZK}}^{R,R'}$ acting as a prover are true with overwhelming probability. That is, with overwhelming probability $\mathcal{Z}$ prompts $\mathcal{A}^*$ to send pairs $(y, (w, x))$ to $\mathcal{F}_{\text{gZK}}^{R,R'}$, for which there is an $\tilde{w}$ satisfying $(y, \tilde{w}) \in R$ and $f(\tilde{w}) = w$.*

In analogy to the standard case we now define UC-emulation with respect to nice environments:

Definition 4.2. *Let ρ be an $\mathcal{F}_{\text{gZK}}^{R,R'}$-hybrid protocol. We say that ρ* UC-emulates a protocol ϕ with respect to the dummy adversary $\mathcal{A}^*$ and nice environments*, if there is an efficient simulator $\mathcal{S}$ such that no nice environment can distinguish whether it is interacting with ρ and $\mathcal{A}^*$ or with ϕ and $\mathcal{S}$. That is, for every nice environment $\mathcal{Z}$ it holds that*

$$\text{EXEC}(\rho, \mathcal{A}^*, \mathcal{Z}) \approx \text{EXEC}(\phi, \mathcal{S}, \mathcal{Z}) .$$

Here, $\text{EXEC}(\rho, \mathcal{A}, \mathcal{Z})$ denotes the random variable given by the output of $\mathcal{Z}$ when interacting with ρ and $\mathcal{A}$, and analogously for $\text{EXEC}(\phi, \mathcal{S}, \mathcal{Z})$.

Having said this, we can now state our special composition theorem, which allows a protocol designer to use zero-knowledge proofs of existence in a UC-compliant way. The theorem is illustrated in Figure 3. On a high level, the theorem allows one to use zero-knowledge proofs of existence as black-boxes in protocol design, as if they were zero-knowledge proofs of knowledge. If a protocol ϕ uses such a proof as a subroutine, one only has to prove the UC-security of the $\mathcal{F}_{\text{gZK}}^{R,R'}$-hybrid variant of ϕ, and this only with respect to nice environments.

Theorem 4.3. *Let `spec` be a proof specification of the form (2), and let $R = R(\texttt{spec})$, $R' = R'(\texttt{spec})$, and $\pi = \pi(\texttt{spec})$. Let further ρ be an $\mathcal{F}_{\text{gZK}}^{R,R'}$-hybrid protocol, such that ρ UC-emulates a protocol ϕ with respect to the dummy adversary and nice environments, and let ρ, ϕ be subroutine respecting. Then $\rho^{\pi/\mathcal{F}_{\text{gZK}}^{R,R'}}$ UC-emulates ϕ (in the standard sense) with respect to adaptive corruptions if securely erasing data is possible.*

Proof (Sketch). Due to space limitations we omit a full proof here, and only give the underlying intuition. Let therefore $\mathcal{S}$ be the simulator for nice environments, which exists by assumption. We have to show that there exists an efficient simulator $\hat{\mathcal{S}}$ such that for arbitrary environments $\mathcal{Z}$ we have that

$$\text{EXEC}(\rho^{\pi/\mathcal{F}_{\text{gZK}}^{R,R'}}, \mathcal{A}^*, \mathcal{Z}) \overset{s}{\approx} \text{EXEC}(\phi, \hat{\mathcal{S}}, \mathcal{Z}) .$$

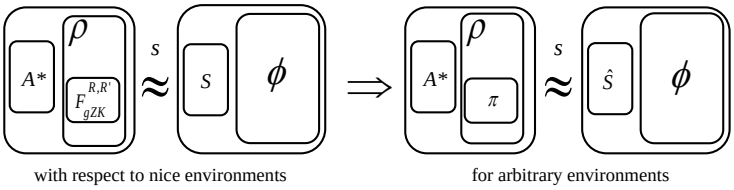

Fig. 3: Illustration of Theorem 4.3: for proving that $\rho^{\pi/\mathcal{F}_{\mathrm{gZK}}^{R,R'}}$ UC-emulates ϕ, it is sufficient to show that ρ emulates ϕ for nice environments.

The idea is that $\hat{\mathcal{S}}$ runs a copy of $\mathcal{S}$ and one of $\mathcal{A}^*$ internally, and all messages sent to or received from $\mathcal{Z}$ are routed through the simulated $\mathcal{A}^*$. In general, all communication is further forwarded to $\mathcal{S}$, and $\hat{\mathcal{S}}$ outputs whatever $\mathcal{S}$ does. The only exception is made when encountering a call to π between two parties, P and V. In this case $\hat{\mathcal{S}}$ internally executes the protocol on the given inputs and behaves as follows (independent of the corruption state of the parties):

- ▷ If the run is successful, then with overwhelming probability the input was correct (i.e., $\mathcal{Z}$ "behaved nicely"), as the underlying Σ-protocol is an interactive proof system [35]. Thus, $\hat{\mathcal{S}}$ outputs whatever $\mathcal{S}$ outputs on the same inputs.
- ▷ If however the run of π is not successful, the given input was incorrect. In this case, $\hat{\mathcal{S}}$ behaves as $\mathcal{S}$ in the case that no `proof`-message had been sent by the attacker.

The remainder of the proof works similar to that in [49]. The main difference is that in the real protocol an honest prover that gets corrupted also hands all witnesses quantified by $\exists$ to the attacker. This can be simulated because of the usage of the committed proof technique [37]: $\hat{\mathcal{S}}$ may choose these secrets arbitrarily at random whenever necessary. It then computes the corresponding image y', and opens the commitment made in its first message accordingly. As in [49], this is possible because of the trapdoor property of the used commitment scheme. □

Summarizing, this modeling for the first time allows one to use ZK proofs *of existence* in the UC-framework. We believe that this decrease of the computational costs of UC-secure protocol can contribute substantially to a broader employment of such protocols in practical applications.

5 Compiling Specifications to Protocols

Due to space limitations we here only give a brief overview about how protocol specifications are compiled into protocols. All details of the construction can be found in §B, where we also show how to estimate the costs of the protocol resulting from a specification of the form (1) in advance.

- ▷ First, the proof specification is rewritten to a predicate which only contains atomic predicates having homogeneous linear relations in their exponents. This can be done by applying standard techniques [6,23,27,30,46,63], cf. §B.1.
- ▷ In a second step, the prover computes integer commitments to all secret witnesses quantified by $\nexists$, see §B.2.
- ▷ Next, using the technique proposed in [27], each conjunctive term in the specification (i.e., each subformula of ϕ not containing any OR connectives) is translated into a Σ-protocol which additionally proves that the witnesses used are the same as in the integer commitments, cf. §B.3.
- ▷ Now, the different Σ-protocols are combined by the Boolean connectives as specified by the predicate ϕ. See [24] and §B.4 for details.

- ▷ As a fifth step, the Σ-protocol is transformed into an Ω-protocol [32,33]. This is achieved by Paillier-encrypting the witnesses quantified by $\forall$ [54], and proving that the encrypted witnesses are the same as in the integer commitments. See §B.5 for details of this step.
- ▷ Finally, using a simulation sound trapdoor commitment [49] and the committed-proof idea of [37], the Ω-protocol is transformed into a protocol UC-emulating $\mathcal{F}_{\text{gZK}}^{R,R'}$. For details, see §B.7.

If the initial proof goal did not contain any secret values quantified by $\exists$, one can show that the resulting protocol UC-emulates the ideal functionality $\mathcal{F}_{\text{ZK}}^{R,R'}$ with respect to adaptive corruptions if securely erasing data is possible. If this is not the case the protocol is still secure for static party corruptions. On a very high level, the idea is that the required simulator can simulate a run of the Ω-protocol for the challenge received from the verifier, and then use the trapdoor of the commitment scheme to open its commitment in the correct way. See [49] for a full proof.

If the initial proof goal contains $\exists$-quantifiers, the resulting protocol can be used as an instantiation of the gullible zero-knowledge functionality $\mathcal{F}_{\text{gZK}}^{R,R'}$, cf. Theorem 4.3.

Let us discuss the potential speed-up and the semantical consequences coming along with the usage of the $\exists$-quantifier by means of our two running examples.

Example 5.1 (Running Example 1). For being able to see the speed-up, we first have to resolve the polynomial relation of Example 3.1. Using the technique from [46], we obtain the following equivalent proof specification:

$$\forall \{\rho_i\}_{i=1}^4 \in \mathcal{I}^*(\lfloor n/4 \rfloor), \{\chi_i\}_{i=1}^4 \in \mathcal{I}^*(\lfloor \sqrt{T} \rfloor), \rho' \in \mathcal{I}^*((4\lfloor \sqrt{T} \rfloor + 1)\lfloor n/4 \rfloor) \; : \; \bigwedge_{i=1}^4 y_i = g^{\chi_i} h^{\rho_i} \; \wedge \; y = y_1^{\chi_1} y_2^{\chi_2} y_3^{\chi_3} y_4^{\chi_4} h^{\rho'}$$

Now one can see that if one additionally proves knowledge of ω, ρ, the quantifiers of all χ_i, ρ_i and ρ' can be changed to existence quantifiers without changing the semantics, as the existence of all these values follows from Lagrange's Four Square Theorem and they can be computed efficiently from ω, ρ by using the algorithm of Rabin and Shallit [61]. We thus obtain the following proof specification:

$$\forall \omega \in \mathcal{I}^*(T), \rho \in \mathcal{I}^*(\lfloor n/4 \rfloor) \; : \; \exists \{\rho_i\}_{i=1}^4 \in \mathcal{I}^*(\lfloor n/4 \rfloor), \{\chi_i\}_{i=1}^4 \in \mathcal{I}^*(\lfloor \sqrt{T} \rfloor), \rho' \in \mathcal{I}^*((4\lfloor \sqrt{T} \rfloor + 1)\lfloor n/4 \rfloor) \; : \; y = g^{\omega} h^{\rho} \; \wedge \; \bigwedge_{i=1}^4 y_i = g^{\chi_i} h^{\rho_i} \; \wedge \; y = y_1^{\chi_1} y_2^{\chi_2} y_3^{\chi_3} y_4^{\chi_4} h^{\rho'}$$

While not touching the semantics of all, this last rewriting yields a significant efficiency speedup, as only Paillier encryptions for 2 instead of 9 values are required. Overall, the prover thus saves 14 Paillier encryptions and evaluations of the integer commitment scheme. Similarly, the verifier's computational complexity is reduced by 7 such operations. □

In this example, changing from the $\forall$- to the $\exists$-quantifier is a purely syntactical step, which increases the efficiency of the protocol. This can be seen by considering the underlying Ω-protocol as f-extractable, where $f(w) = (w, \mathsf{A}(w))$ and A is the algorithm of [61]. However, in general it is not possible to efficiently compute the witnesses quantified by $\exists$, and even their existence cannot be verified efficiently, as is illustrated by the following example.

Example 5.2 (Running Example 2). The following specification is sufficient for proving the required wellformedness of the ciphertext:

$$\nexists\, \alpha \in \mathbb{Z}_q \,:\, \exists\, \rho \in \mathbb{Z}_q \,:\, u_1 = g^\rho \,\wedge\, u_2 = h^\rho \,\wedge\, e = g^\rho c^\alpha \,.$$

This observation reduces the complexity of the prover's first algorithm in the resulting protocol by 2 Paillier encryptions (namely, that of ρ and the corresponding randomness in the underlying Σ-protocol), and 2 evaluations of the integer commitment scheme. □

Here, the underlying Ω-protocol is f-extractable, where f is of the form $f(w_1, \ldots, w_n) = (w_1, \ldots, w_k)$ for $k < n$, such that the remaining w_i cannot be computed. This implies that in general it is not possible to construct an ideal functionality which captures the semantics of an expression such as (2), as it would have to run in probabilistic polynomial time by definition [17].

6 Enhancing the Basic Language

Even if the basic language presented in §3 allows one to describe almost arbitrary algebraic properties of and relations among the secret values, the corresponding specifications sometimes become hard to interpret, as these properties may not be specified explicitly. Also, the requirements that all witnesses must be integer exponents, and that the groups orders must not have small divisors, may seem overly restrictive.

For those reasons we next give some enhancements of our basic language, which allow one to drop any of these conditions. More precisely, we will first define a set of macros for specifying algebraic properties of the secret witnesses, and then give conditions under which knowledge of group elements can be proved instead of integers. Finally, we show how small divisors in the group order can be handled by loosing the soundness guarantees of the protocol slightly.

6.1 Using Macros to Specify Algebraic Properties of Witnesses

The language described in §3 does not allow to directly specify algebraic properties of the secrets or algebraic relations among them, and thus it becomes inconvenient to use for complex proof goals. We therefore extend the set of atomic predicates by so-called *macros*, which allow one to directly describe algebraic properties of the integer witnesses ω_i. In particular, we allow additional atomic predicates of the following forms, all of which can easily be translated into polynomial relations:

- ▷ $\boldsymbol{\omega \geq 0}$. Such statements can easily be translated into statements of the above form by proving knowledge of integers $\chi_1, \ldots, \chi_4$ such that $\omega = \sum_{i=1}^{4} \chi_i^2$, see [46].
  More generally, we also allow expressions of the form $\omega \in [a, b]$, where $a, b \in \mathbb{Z}$ are public. Such an expression is equivalent to $\omega - a \geq 0 \,\wedge\, b - \omega \geq 0$. If $b - a$ is even, this can be rewritten to the even more efficient proof goal $(\omega - m)^2 \geq d^2$, where $m = \frac{a+b}{2}$ and $d = \frac{b-a}{2}$.
  If the length of the interval is very small, i.e., $b - a \approx 5$, it might even be more efficient to perform an OR-proof, and to show that $\omega = a \vee \omega = a + 1 \vee \cdots \vee \omega = b$.
- ▷ $\boldsymbol{\mathbf{gcd}(\nu_1, \nu_2) = 1}$, where each ν_1, ν_2 can be either public or private. As before, such expressions can be rewritten to a polynomial form by introducing additional integers α_1, α_2 and proving knowledge of $\alpha_1, \alpha_2, \nu_1, \nu_2$ such that $\alpha_1\nu_1 + \alpha_2\nu_2 = 1$.
- ▷ $\boldsymbol{\nu_1 | \nu_2}$, where ν_1, ν_2 can be either public or private. By introducing an additional secret δ, such relations can be expressed in polynomial form as $\delta\nu_1 - \nu_2 = 0$.

Example 6.1 (Running Example 1). Using the first of our specific macros, a protocol for proving knowledge of a non-negative opening of the integer commitment y can be described as follows:

$$\mathsf{K}\,\omega \in \mathcal{I}^*(T), \rho \in \mathcal{I}^*(\lfloor n/4 \rfloor) \,:\, y = g^{\omega} h^{\rho} \,\wedge\, \omega \geq 0 \qquad \square$$

Before moving to the next extension of our basic language, we point out that using macros impedes the possibility of estimating the computational costs of the protocol from its specification, which was a favorable property of our basic language. This can be seen by comparing Example 6.1 to Example 3.1: the seemingly simple macro $w \geq 0$ entails 5 atomic predicates, and 9 secret witnesses, and thus conceals very much of the computational costs of the resulting protocol.

As an important remark we note that every auxiliary variables χ_i, which has to be introduced when resolving any of these macros, can be quantified by $\exists$. This can easily be seen by noting that considering the resulting Ω-protocol as f-extractable for $f(w) = (w, \mathsf{A}(w))$, where A is the algorithm the honest prover used to compute the χ_i from ω.

6.2 Proving Knowledge of Group Elements

Sometimes it is required to prove knowledge of *group elements* instead of integers, which is not possible in our basic language. For instance, one might be interested in proving possession of a digital signature on a given message, which, in the case of CL-signatures [14], essentially boils down to proving knowledge of a group element ω such that $e(\omega, z) = y$, where e is a bilinear map, and y, z are publicly known.

We thus also allow one to specify protocols proving knowledge of a preimage $\omega \in \mathcal{G}$ under some group homomorphism $\psi : \mathcal{G} \to \mathcal{H}$, if ψ satisfies two basic properties: (i) the finite group $\mathcal{G}$ comes along with a generator g and an upper bound B on its order, and (ii) the discrete logarithm problem is hard in $\mathcal{H}$. Then expressions of the following form, which, of course, can arbitrarily be combined with expressions of the basic language, may be used:

$$\mathsf{K}\,\omega \in \mathcal{G} \,:\, y = \psi(\omega)\,.$$

When compiling protocol specifications containing such expressions, one first has to perform the following steps, and then proceeds as in §5. The idea of the construction is to first blind the secret preimage ω using g, and then to prove knowledge of the blinding:

1. Set $m' = 2^l B$, where B is the given upper bound on $\operatorname{ord}\mathcal{G}$, and l is a security parameter.
2. Choose $\omega' \in_R \mathcal{I}(m')$, and set $u = g^{\omega'}\omega$, where g is a generator of $\mathcal{G}$. Further, set $y' = \psi(u)/y$, and $g' = \psi(g)$.
3. Rewrite the proof goal to
$$\mathsf{K}\,\omega' \in \mathcal{I}^*(m') \,:\, y' = g'^{\omega'}\,,$$
and add u to the commitment of the underlying Σ-protocol.

6.3 Allowing Small Divisors in the Group Order

In certain situations, in particular when extending an existing system, it might not be possible to use groups the order of which has large prime divisors only. Let consequently be $\mathcal{H}$ the group used in some atomic predicate, and let d be such that $(\operatorname{ord}\mathcal{H})/d$ only has large divisors, i.e., let d be the product of all (not necessarily distinct) small primes dividing $\operatorname{ord}\mathcal{H}$.

The protocol resulting from the construction in §5 now convinces the verifier that the prover knows secret values $\omega_i^* \in \mathcal{I}^*(m_{\omega_i})$, and that there exists $\mu \in \mathcal{H}$, such that the following is satisfied:

$$\mu \prod_{i=1}^{u} g_i^{l_i(\omega_1^*,\ldots,\omega_n^*)} = y \qquad \text{and} \qquad \mu^d = 1\,.$$

That is, knowledge of the witnesses is only proved "modulo a factor with small order".

Example 6.2. Let $n = pq$ be a safe RSA modulus, and let $\operatorname{lcm}(p-1, q-1) = 2$. As we have $\operatorname{ord} \mathbb{Z}_n^* = 4\frac{p-1}{2}\frac{q-1}{2}$, we have $d = 4$. Thus, the construction of §5 yields protocols that only guarantee that the prover knows a representation up to a factor of order 4. (Actually we get $d = 2$ and soundness modulo a square root of unity by the Carmichael function.) □

6.4 Using Private Coin System Parameters to Avoid Integer Commitments

In the construction given in §B.5 an integer commitment has to be computed for every secret value ω_i. This was necessary to ensure that the values used in the public values y_j and in the Paillier-encryptions are equal over the integers. In the following we give sufficient conditions under which these integer commitments can be avoided.

Proposition 6.3. *Let, for some secret ω_i and every subformula only containing ANDs of predicate ϕ, the following condition be satisfied: ω_i appears at least once as an exponent in a group, which (i) can be modeled as a* private coin *system parameter, and for which (ii) computing the group order from its description is computationally hard. Then no integer commitment is needed for ω_i.*

While the second condition is straightforward to check, the former conditions requires some more discussion. In §2.1 we required all system parameters specifying the binary relation R to be modeled as *public coin* system parameters. This implies that random coins for generating the group description, and thus also the group orders, have to be assumed to be known, and thus the y_j cannot be seen as integer commitments to the ω_i, but as commitments modulo the order at most.

In general, modeling the parameters as public coin cannot be avoided, as system parameters can potentially be used across various protocols of different types. Now, if a group is also used in, e.g., a signature scheme, then typically the factorization of the group order is known to the signer, which could lead to serious security issues in the UC-ZK protocols if this factorization is assumed to be unknown. However, if for instance an RSA modulus n can be assumed to be issued by a trusted party which does not (have to) give the prime factors of n to any party, then $\mathbb{SR}_n$ could safely be modeled as a private coin system parameter.

References

1. M. Abe and M. Ohkubo. A framework for universally composable non-committing blind signatures. In M. Matsui, editor, *ASIACRYPT '09*, volume 5912 of *LNCS*, pages 435–450. Springer, 2009.
2. G. Ateniese, D. X. Song, and G. Tsudik. Quasi-efficient revocation in group signatures. In M. Blaze, editor, *FC '02*, volume 2357 of *LNCS*, pages 183–197. Springer, 2002.
3. M. Backes, B. Pfitzmann, and M. Waidner. The reactive simulatability (RSIM) framework for asynchronous systems. *Information and Computation*, 205(12):1685–1720, 2007.
4. M. Bellare and O. Goldreich. On defining proofs of knowledge. In E. F. Brickell, editor, *CRYPTO '92*, volume 740 of *LNCS*, pages 390–420. Springer, 1993.
5. F. Boudot. Efficient proofs that a committed number lies in an interval. In B. Preneel, editor, *EUROCRYPT '00*, volume 1807 of *LNCS*, pages 431–444. Springer, 2000.

6. S. Brands. Rapid demonstration of linear relations connected by boolean operators. In W. Fumy, editor, *EUROCRYPT '97*, volume 1233 of *LNCS*, pages 318–333. Springer, 1997.
7. E. Bresson and J. Stern. Efficient revocation in group signatures. In K. Kim, editor, *PKC '01*, volume 1992 of *LNCS*, pages 190–206. Springer, 2001.
8. E. Brickell, J. Camenisch, and L. Chen. Direct anonymous attestation. In V. Atluri, M. Backes, D. A. Basin, and M. Waidner, editors, *ACM CCS '04*, pages 132–145. ACM Press, 2004.
9. L. Bussard, R. Molva, and Y. Roudier. History-based signature or how to trust anonymous documents. In C. D. Jensen, S. Poslad, and T. Dimitrakos, editors, *iTrust*, volume 2995 of *LNCS*, pages 78–92. Springer, 2004.
10. L. Bussard, Y. Roudier, and R. Molva. Untraceable secret credentials: Trust establishment with privacy. In *PerCom Workshops*, pages 122–126. IEEE, 2004.
11. J. Camenisch, N. Casati, T. Groß, and V. Shoup. Credential authenticated identification and key exchange. In T. Rabin, editor, *CRYPTO '10*, volume 6223 of *LNCS*, pages 255–276. Springer, 2010.
12. J. Camenisch and E. V. Herreweghen. Design and implementation of the idemix anonymous credential system. In V. Atluri, editor, *ACM CCS '02*, pages 21–30. ACM Press, 2002.
13. J. Camenisch, A. Kiayias, and M. Yung. On the portability of Generalized Schnorr Proofs. In A. Joux, editor, *EUROCRYPT '09*, volume 5479 of *LNCS*, pages 425–442. Springer, 2009.
14. J. Camenisch and A. Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In M. K. Franklin, editor, *CRYPTO '04*, volume 3152 of *LNCS*, pages 56–72. Springer, 2004.
15. J. Camenisch and V. Shoup. Practical verifiable encryption and decryption of discrete logarithms. In D. Boneh, editor, *CRYPTO '03*, volume 2729 of *LNCS*, pages 126–144, 2003.
16. J. Camenisch and M. Stadler. Efficient group signature schemes for large groups (extended abstract). In B. Kaliski, editor, *CRYPTO '97*, volume 1294 of *LNCS*, pages 410–424. Springer, 1997.
17. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS '01*, pages 136–145. IEEE, 2001. Revised version available at `http://eprint.iacr.org/2000/067`.
18. R. Canetti, Y. Dodis, R. Pass, and S. Walfish. Universally composable security with global setup. In S. Vadhan, editor, *TCC '07*, volume 4392 of *LNCS*, pages 61–85. Springer, 2007.
19. R. Canetti and H. Krawczyk. Security analysis of IKE's signature-based key-exchange protocol. In M. Yung, editor, *CRYPTO '02*, volume 2442 of *LNCS*, pages 143–161. Springer, 2002.
20. R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai. Universally composable two-party and multi-party secure computation. In *STOC '02*, pages 494–503. ACM Press, 2002.
21. R. Canetti and T. Rabin. Universal composition with joint state. In D. Boneh, editor, *CRYPTO '03*, volume 2729 of *LNCS*, pages 265–281. Springer, 2003.
22. R. Cramer. *Modular Design of Secure yet Practical Cryptographic Protocols*. PhD thesis, CWI and University of Amsterdam, 1997.
23. R. Cramer and I. Damgård. Zero-knowledge proofs for finite field arithmetic; or: Can zero-knowledge be for free? In H. Krawczyk, editor, *CRYPTO '98*, volume 1462 of *LNCS*, pages 424–441. Springer, 1998.
24. R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In Y. Desmedt, editor, *CRYPTO '94*, volume 839 of *LNCS*, pages 174–187. Springer, 1994.
25. R. Cramer and V. Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In H. Krawczyk, editor, *CRYPTO '98*, volume 1462 of *LNCS*. Springer, 1998.
26. I. Damgård. On Σ-protocols, 2004. Lecture on Cryptologic Protocol Theory; Faculty of Science, University of Aarhus.
27. I. Damgård and E. Fujisaki. A statistically-hiding integer commitment scheme based on groups with hidden order. In Y. Zheng, editor, *ASIACRYPT '02*, volume 2501 of *LNCS*, pages 77–85. Springer, 2002.
28. Y. Dodis, V. Shoup, and S. Walfish. Efficient constructions of composable commitments and zero-knowledge proofs. In D. Wagner, editor, *CRYPTO '08*, volume 5157 of *LNCS*, pages 515–535. Springer, 2008.
29. R. Fischlin and C. Schnorr. Stronger security proofs for RSA and Rabin bits. *Journal of Cryptology*, 13(2):221–244, 2000.
30. E. Fujisaki and T. Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In B. Kaliski, editor, *CRYPTO '97*, volume 1294 of *LNCS*, pages 16–30. Springer, 1997.
31. J. Furukawa and S. Yonezawa. Group signatures with separate and distributed authorities. In C. Blundo and S. Cimato, editors, *SCN' 04*, volume 3352 of *LNCS*, pages 77–90. Springer, 2004.
32. J. A. Garay, P. MacKenzie, and K. Yang. Strengthening zero-knowledge protocols using signatures. In E. Biham, editor, *EUROCRYPT '03*, volume 2656 of *LNCS*, pages 177–194. Springer, 2003.
33. J. A. Garay, P. MacKenzie, and K. Yang. Strengthening zero-knowledge protocols using signatures. *Journal of Cryptology*, 19(2):169–209, 2006.
34. M. Gaud and J. Traoré. On the anonymity of fair offline e-cash systems. In R. N. Wright, editor, *FC '03*, volume 2742 of *LNCS*, pages 34–50. Springer, 2003.

35. S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof-systems. In *STOC '85*, pages 291–304. ACM Press, 1985.
36. D. Hofheinz and E. Kiltz. The group of signed quadratic residues and applications. In S. Halevi, editor, *CRYPTO '09*, volume 5677 of *LNCS*, pages 637–653. Springer, 2009.
37. S. Jarecki and A. Lysyanskaya. Adaptively secure threshold cryptography: Introducing concurrency, removing erasures. In B. Preneel, editor, *EUROCRYPT '00*, volume 1807 of *LNCS*, pages 221–242. Springer, 2000.
38. H. Krawczyk. HMQV: A high-performance secure Diffie-Hellman protocol. In V. Shoup, editor, *CRYPTO '05*, volume 3621 of *LNCS*, pages 546–566. Springer, 2005.
39. K. Kurosawa and J. Furukawa. Universally composable undeniable signature. In L. Aceto, I. Damgård, L. Goldberg, M. Halldórsson, A. Ingólfsdóttir, and I. Walukiewicz, editors, *ICALP (Part II) '08*, volume 5126 of *LNCS*, pages 524–535. Springer, 2008.
40. R. Küsters and M. Tuengerthal. Universally composable symmetric encryption. In *Computer Security Foundations Symposium – CSF 09*, pages 293–307. IEEE, 2009.
41. P. Laud and L. Ngo. Threshold homomorphic encryption in the universally composable cryptographic library. In J. Baek, F. Bao, K. Chen, and X. Lai, editors, *Proceedings of the 2nd International Conference on Provable Security – ProvSec 08*, volume 5324 of *LNCS*, pages 298–312. Springer, 2008.
42. T. Van Le, K. Quoc Nguyen, and V. Varadharajan. How to prove that a committed number is prime. In K.-Y. Lam, E. Okamoto, and C. Xing, editors, *ASIACRYPT '99*, volume 1716 of *LNCS*, pages 208–218. Springer, 1999.
43. Y. Lindell. Bounded-concurrent secure two-party computation without setup assumptions. In *STOC '03*, pages 683–692. ACM Press, 2003.
44. Y. Lindell. Protocols for bounded-concurrent secure two-party computation in the plain model. *Chicago Journal of Theoretical Computer Science*, 2006(1):1–50, 2006.
45. Y. Lindell. Highly-efficient universally-composable commitments based on the DDH assumption. In *EUROCRYPT '11*, 2011. (to appear).
46. H. Lipmaa. On diophantine complexity and statistical zero knowledge arguments. In C.-S. Laih, editor, *ASIACRYPT '03*, volume 2894 of *LNCS*, pages 398–415. Springer, 2003.
47. A. Lysyanskaya and Z. Ramzan. Group blind digital signatures: A scalable solution to electronic cash. In R. Hirschfeld, editor, *FC '98*, volume 1465 of *LNCS*, pages 184–197. Springer, 1998.
48. P. MacKenzie and M. Reiter. Two-party generation of dsa signatures. In J. Kilian, editor, *CRYPTO '01*, volume 2139 of *LNCS*, pages 137–154. Springer, 2001.
49. P. MacKenzie and K. Yang. On simulation-sound trapdoor commitments. In C. Cachin and J. Camenisch, editors, *EUROCRYPT '04*, volume 3027 of *LNCS*, pages 382–400. Springer, 2004.
50. E. Mykletun, M. Narasimha, and G. Tsudik. Signature bouquets: Immutability for aggregated/condensed signatures. In P. Samarati, Peter Y. A. Ryan, D. Gollmann, and R. Molva, editors, *ESORICS '04*, volume 3193 of *LNCS*, pages 160–176. Springer, 2004.
51. T. Nakanishi, M. Shiota, and Y. Sugiyama. An efficient online electronic cash with unlinkable exact payments. In K. Zhang and Y. Zheng, editors, *ISC '04*, volume 3225 of *LNCS*, pages 367–378. Springer, 2004.
52. R. Nishimaki, Y. Manabe, and T. Okamoto. Universally composable identity-based encryption. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E91-A(1):262–271, 2008.
53. National Institute of Standards and Technology. *FIPS PUB 186-3: Digital Signature Standard (DSS)*. National Institute for Standards and Technology, Gaithersburg, MD, USA, June 2009.
54. P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In J. Stern, editor, *EUROCRYPT '99*, volume 1592 of *LNCS*, pages 223–238. Springer, 1999.
55. R. Pass and A. Rosen. Bounded-concurrent secure two-party computation in a constant number of rounds. In *FOCS '03*, pages 404–413. IEEE, 2003.
56. T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In J. Feigenbaum, editor, *CRYPTO '91*, volume 576 of *LNCS*, pages 129–140. Springer, 1991.
57. C. Peikert, V. Vaikuntanathan, and B. Waters. A framework for efficient and composable oblivious transfer. In D. Wagner, editor, *CRYPTO '08*, volume 5157 of *LNCS*, pages 554–571. Springer, 2008.
58. B. Pfitzmann and M. Waidner. Composition and integrity preservation of secure reactive systems. In *ACM CCS '00*, pages 245–254, 2000.
59. M. Prabhakaran and A. Sahai. New notions of security: achieving universal composability without trusted setup. In *STOC '04*, pages 242–251. ACM Press, 2004.
60. M. Prabhakaran and A. Sahai. Relaxing environmental security: Monitored functionalities and client-server computation. In J. Kilian, editor, *TCC '05*, volume 3378 of *LNCS*, pages 104–127. Springer, 2005.
61. M. Rabin and J. Shallit. Randomized algorithms in number theory. *Communications in Pure and Applied Mathematics*, 39(S1):239–256, 1986.
62. N. P. Smart. *Cryptography: An Introduction (3rd Edition)*. `http://www.cs.bris.ac.uk/~nigel/`, 2008.

63. N. P. Smart, editor. *Final Report on Unified Theoretical Framework of Efficient Zero-Knowledge Proofs of Knowledge*. `http://www.cace-project.eu`, 2009. CACE Project Deliverable.
64. D. Xiaodong Song. Practical forward secure group signature schemes. In *ACM CCS '01*, pages 225–234. ACM press, 2001.
65. W. Susilo and Y. Mu. On the security of nominative signatures. In C. Boyd and J. Manuel González Nieto, editors, *ACISP '05*, volume 3574 of *LNCS*, pages 329–335. Springer, 2005.
66. C. Tang, Z. Liu, and M. Wang. A verifiable secret sharing scheme with statistical zero-knowledge. Cryptology ePrint Archive, Report 2003/222, 2003. `http://eprint.iacr.org/`.
67. P. Tsang and V. Wei. Short linkable ring signatures for e-voting, e-cash and attestation. In R. H. Deng, F. Bao, H. Pang, and J. Zhou, editors, *ISPEC '05*, volume 3439 of *LNCS*, pages 48–60. Springer, 2005.
68. P. Tsang, V. Wei, T. Chan, M. Ho Au, J. K. Liu, and D. S. Wong. Separable linkable threshold ring signatures. In A. Canteaut and K. Viswanathan, editors, *INDOCRYPT '04*, volume 3348 of *LNCS*, pages 384–398. Springer, 2004.
69. V. Wei. Tracing-by-linking group signatures. In J. Zhou, J. Lopez, R. H. Deng, and F. Bao, editors, *ISC '05*, volume 3650 of *LNCS*, pages 149–163. Springer, 2005.
70. J. Zhou, Y. Zhu, T. Chang, and Y. Zhang. Universally composable key-evolving signature. In *CISIM '07*, pages 97–102. IEEE, 2007.

A Basics and Definitions

For self-containment we here recapitulate the basic definitions used in the main part of this paper.

A.1 Σ- and Ω-Protocols

We first want to recapitulate the basic notions of Σ- and Ω-protocols, which constitute fundamental building blocks of the protocols constructed in §5 and §B.

On a high level, a Σ-protocol [22] is a three-move protocol between two parties, which realizes a zero-knowledge proofs of knowledge with respect to honest-but-curious verifiers in the non-UC setting:

Definition A.1. *Let R be a binary relation, and let L_R be the associated language. A* Σ-protocol *for R then is a two party protocol $((\mathsf{P}_1, \mathsf{P}_2)(w), \mathsf{V})(y, \sigma)$ between a prover and a verifier, satisfying the following properties:*

- ▷ **3-move form:** *First, the prover computes $(t, r) = \mathsf{P}_1(y, w, \sigma)$, and sends the* commitment *t to the verifier, whereas r describes its internal state after executing P_1. The verifier draws a random* challenge *c from some predefined challenge set $\mathcal{C}$, and sends it to the prover, who replies with a* response *$s = \mathsf{P}_2(y, w, c, r, \sigma)$. Upon receiving s, the verifier computes a bit $b = \mathsf{V}(y, t, c, s, \sigma)$ and* accepts *the protocol run if and only if $b = 1$.*
- ▷ **Completeness:** *Whenever both parties follow the protocol and $(y, w) \in R$, the verifier always accepts.*
- ▷ **Special soundness:** *There is an efficient algorithm E, which on input $y \in L_R$, σ and two accepting protocol transcripts (t, c_1, s_1) and (t, c_2, s_2) satisfying $c_1 \neq c_2$ outputs w' such that $(y, w') \in R$.*
- ▷ **Special honest-verifier zero-knowledge:** *There exists an efficient algorithm M such that the following is satisfied: on input $y \in L_R$, σ and $c \in \mathcal{C}$, M outputs tuples $(\tilde{t}, c, \tilde{s})$ which are (statistically) indistinguishable from real protocol transcripts with challenge c.*

Here, σ denotes a potentially empty common reference string. For the remainder of this paper we will assume $\mathcal{C} = [0, 2^k - 1]$, where k is a security parameter. Typical values are $k = 80$ or $k = 128$.

It can be shown that any Σ-protocol satisfies the standard definition of ZK-PoK [4] with knowledge error $1/2^k$ with respect to honest verifiers, e.g., [26]. That is, having rewindable black box access to a prover making the verifier accept with probability larger than $1/2^k$, it is possible to compute the witness in a time proportional to the prover's advantage. Yet, in the UC model the notion of rewindable blackbox access is not applicable any more. Thus, it must be possible to extract the prover's private input already after one successful execution of the protocol. To circumvent this problem, Garay et al. introduced the notion of Ω-protocols in [32]. On a high level, an Ω-protocol is a Σ-protocol using a common reference string for which there is a trapdoor τ to σ which allows one to extract witness without having to rewind the prover.

Definition A.2. *Let R be a binary relation. A two party protocol $((\mathsf{P}_1, \mathsf{P}_2)(w), \mathsf{V})(y, \sigma)$ is called an* Ω-protocol *for R, if it is a Σ-protocol which additionally satisfies the following requirements:*

- ▷ **Common reference string:** *A common reference string σ is drawn according to some predefined distribution, and is given to $(\mathsf{P}_1, \mathsf{P}_2)$ and V as an additional common input.*
- ▷ **Trapdoor extraction:** *There exists a pair of algorithms $(\mathsf{E}_1, \mathsf{E}_2)$ satisfying the following two conditions: (i) algorithm E_1 outputs pairs (σ', τ'), such that the distribution of σ' is statistically indistinguishable from that of uniformly chosen common reference strings, and (ii) if for some t there exist two accepting transcripts (t, c_1, s_1) and (t, c_2, s_2) satisfying $c_1 \neq c_2$, we have that $(y, \mathsf{E}_2(y, \tau', \sigma', t, c, s)) \in R$ for all c, s such that $\mathsf{V}(y, t, c, s, \sigma') = 1$.*

We call an Ω-protocol *f-extractable*, if algorithm E_2 does not output a witness for y, but only $f(w)$ for some w satisfying $(y, w) \in R$. By allowing this relaxation we will show how to achieve higher efficiency in the resulting protocols, while still ensuring appropriate security guarantees for many practical applications.

A.2 The Concept of UC-Emulation

After having given an informal definition of UC-emulation in the main part of the paper, let us now describe formally what it means that a protocol ρ UC-emulates another protocol ϕ. Namely, for every efficient *attacker* $\mathcal{A}$ on ρ, there has to be an efficient *simulator* $\mathcal{S}$ for ϕ such that no efficient *environment* $\mathcal{Z}$ can distinguish whether it is interacting with ρ and $\mathcal{A}$, or with ϕ and $\mathcal{S}$. That is, the simulator $\mathcal{S}$ performs an equivalent attack on ϕ as $\mathcal{A}$ did on ρ.

Let $\mathrm{EXEC}(\rho, \mathcal{A}, \mathcal{Z})$ denote the random variable given by the output of $\mathcal{Z}$ when interacting with ρ and $\mathcal{A}$, and let $\mathrm{EXEC}(\phi, \mathcal{S}, \mathcal{Z})$ be defined analogously.

Definition A.3. *For PPT protocols ρ and ϕ we say that ρ* UC-emulates *ϕ, if for any PPT adversary $\mathcal{A}$ there exists a PPT simulator $\mathcal{S}$ such that for any PPT environment $\mathcal{Z}$ it holds that*

$$\mathrm{EXEC}(\rho, \mathcal{A}, \mathcal{Z}) \overset{s}{\approx} \mathrm{EXEC}(\phi, \mathcal{S}, \mathcal{Z})\,.$$

When proving that ρ UC-emulates ϕ, it is sufficient to show that there exists a PPT simulator for the *dummy adversary* $\mathcal{A}^*$, which often simplifies the security analysis significantly [17]. The dummy adversary is an adversary which is completely under control of the environment $\mathcal{Z}$ and only forwards messages from $\mathcal{Z}$ to the parties participating in ρ and vice versa.

Now, the following composition theorem guarantees that a protocol ρ UC-emulating ϕ can securely be used instead of ϕ in arbitrary contexts [17], whenever ρ and ϕ are *subroutine respecting*, i.e., whenever no sub-party of ρ respectively ϕ accepts inputs (or passes outputs) from (or to) any party which is not a sub-party of ρ respectively ϕ itself.

The authenticated channel functionality $\mathcal{F}_{\text{ach}}$

1. Wait for an input $(\texttt{send}, x)$ from P and `ready` from Q, and send $(\texttt{send}, x)$ respectively `ready` to $\mathcal{A}$.
2. Wait for a message `deliver` from $\mathcal{A}$ and send $(\texttt{deliver}, x)$ to Q.

Corruption rules:

- ▷ If P gets corrupted after sending $(\texttt{send}, x)$ and before Step 2, $\mathcal{A}$ is allowed to change the value of x at any time before Step 2.

Fig. 4: The authenticated channel functionality $\mathcal{F}_{\text{ach}}$.

The secure channel functionality $\mathcal{F}_{\text{sch}}$

1. Wait for an input $(\texttt{send}, x)$ from P and `ready` from Q, and send $(\texttt{send}, \ell(x))$ respectively `ready` to $\mathcal{A}$.
2. Wait for a message `deliver` from $\mathcal{A}$ and send $(\texttt{deliver}, x)$ to Q.

Corruption rules:

- ▷ If P gets corrupted after sending $(\texttt{send}, x)$ and before Step 2, $\mathcal{A}$ learns x and is allowed to change its value any time before Step 2.

Fig. 5: The secure channel functionality $\mathcal{F}_{\text{sch}}$.

Theorem A.4. *Let π, ϕ, ρ be PPT protocols such that ρ UC-emulates ϕ, and ϕ, ρ are subroutine respecting. Then $\pi^{\rho/\phi}$ UC-emulates π, where $\pi^{\rho/\phi}$ denotes the same protocol as π with all instances of ϕ being substituted by instances of ρ.*

Typically, one assumes that the communication channel being used is at least authenticated, but not necessarily private. That is, the functionality $\mathcal{F}_{\text{ach}}$ delivers a message together with the ID of its sender to the receiver. The attacker is only allowed to change the content (but not the ID of the sender) for corrupted parties. See Figure 4 for a formal description of $\mathcal{F}_{\text{ach}}$. Secure (or private) channels are realized by the functionality $\mathcal{F}_{\text{sch}}$ (cf. Figure 5) in which an attacker does not even learn the content of a message, but only some information leakage $\ell(x)$ (such as the length of x, etc.).

A.3 The Group of Signed Quadratic Residues

Let n be a Blum integer, i.e., $n = pq$ where p, q are primes congruent 3 modulo 4, and let elements of $\mathbb{Z}_n^*$ be represented by elements of the set $\{-\frac{n-1}{2}, \ldots, \frac{n-1}{2}\}$. The *signed quadratic residues* modulo n [29,36], denoted by $\mathbb{SR}_n$, are then given by

$$\mathbb{SR}_n = \{|x| \,:\, x = y^2 \text{ for some } y \in \mathbb{Z}_n^*\}\,,$$

where $|.|$ denoted the absolute value. The group operation is then defined by

$$(gh)_{\mathbb{SR}_n} := |(gh)_{\mathbb{Z}_n^*}|\,.$$

We will omit the subscripts if there is no danger of confusion.

It is well known that $\mathbb{SR}_n$ is a group of order $\frac{p-1}{2}\frac{q-1}{2}$, which does not contain small prime factors if n is a safe RSA modulus, i.e., p, q are each twice a prime plus 1, and $p \neq q$. Further, in contrast to the $\mathbb{QR}_n$, the group of signed quadratic residues is efficiently recognizable. More precisely, we

have that $\mathbb{SR}_n$ consists of all elements in $\{0,\ldots,\frac{n-1}{2}\}$ whose Jacobi symbol is equal to 1. See [62] for how to compute the Jacobi symbol. We finally note that $\mathbb{SR}_n$ is a cyclic group whenever $\mathbb{QR}_n$ is, and that for a generator g of $\mathbb{QR}_n$ we have that $|g|$ generates $\mathbb{SR}_n$.

For a detailed discussion of the group of signed quadratic residues we refer to [36].

B Compilation Details

In the following we show how proof specifications are compiled into concrete protocols. We consider a specification of the form

$$\forall\, \omega_1 \in \mathcal{I}^*(m_{\omega_1}),\ldots,\omega_n \in \mathcal{I}^*(m_{\omega_n}) \,:\, \exists\, \chi_1 \in \mathcal{I}^*(m_{\chi_1}),\ldots,\chi_{n'} \in \mathcal{I}^*(m_{\chi_{n'}}) \,:\, \phi(\omega_1,\ldots,\omega_n,\chi_1,\ldots,\chi_{n'})\,.$$

B.1 Resolving Algebraic Relations to Discrete Logarithm Based Proofs

In a first step, all non-linear polynomial relations occurring in the exponents of atomic predicates have to be resolved. Principally, this can be done using either standard technique found in the literature, e.g., [6,23,27,30,46,63]. However, in many of these constructions the computationally inexpensive $\exists$-quantifier can be used for certain auxiliary witnesses.

In particular, this is the case for all randomnesses which are used to blind additional commitments when resolving multiplicative relations. As an example, consider the case where one wants to prove knowledge of the square root contained in a Pedersen commitment [56]. That is, one wants to perform the following proof:

$$\forall\, \omega,\rho \in \mathbb{Z}_q \,:\, y = g^{\omega^2} h^{\rho}\,.$$

Standard techniques resolve this polynomial expression to

$$\forall\, \omega,\rho_1,\rho_2 \in \mathbb{Z}_q \,:\, y' = g^{\omega} h^{\rho_2} \quad\wedge\quad y = y'^{\omega} h^{\rho_1}\,.$$

However, it is easy to see that the following statement would still be equivalent to the original proof goal, which saves the evaluation of two integer commitments and Paillier encryptions, as will be shown in the following sections:

$$\forall\, \omega,\rho_1 \in \mathbb{Z}_q \,:\, \exists\, \rho_2 \in \mathbb{Z}_q \,:\, y' = g^{\omega} h^{\rho_2} \quad\wedge\quad y = y'^{\omega} h^{\rho_2}\,.$$

Furthermore, if working over the integers, auxiliary commitments may already be computed using the common reference string used in §B.2, and can then be reused there without any further computations.

After this step, all atomic predicates in the reformulated proof specification are of the form

$$y = \prod_{i=1}^{u} g_j^{l_j(\omega_1,\ldots,\omega_n,\chi_1,\ldots,\chi_{n'})}$$

for homogeneous linear functions $l_j(\omega_1,\ldots,\omega_n,\chi_1,\ldots,\chi_{n'})$, where the ω_i are quantified by $\forall$, and the χ_j by $\exists$.

B.2 Computing Integer-Commitments

Next, an integer commitment $\tilde{\mathfrak{y}}_{\omega_i}$ has to be computed for each secret witness ω_i. To this end, we assume the availability of a common reference string $\tilde{\sigma}$ of the following form:

▷ A triple $(\tilde{\mathfrak{n}}, \tilde{\mathfrak{g}}, \tilde{\mathfrak{h}})$ where $\tilde{\mathfrak{n}}$ is a safe RSA modulus of unknown factorization, and $\tilde{\mathfrak{g}}, \tilde{\mathfrak{h}}$ are generators of the signed quadratic residues modulo $\tilde{\mathfrak{n}}$, such that their relative discrete logarithms are also unknown [27,30].

Then the following computations are performed:

▷ For each witness ω_i, the prover draws $\tilde{\mathfrak{w}}_i \in_R [-\lfloor \tilde{\mathfrak{n}}/4 \rfloor, \lfloor \tilde{\mathfrak{n}}/4 \rfloor]$ and computes $\tilde{\mathfrak{y}}_i = \tilde{\mathfrak{g}}^{\omega_i} \tilde{\mathfrak{h}}^{\tilde{\mathfrak{w}}_i}$. It sets $\tilde{\mathfrak{y}} = (\tilde{\mathfrak{y}}_1, \ldots, \tilde{\mathfrak{y}}_n)$ and similarly for $\tilde{\mathfrak{w}}$. Here and in the following, $\lfloor a \rfloor$ denotes the largest integer smaller then a.

Clearly, if in the previous step a commitment of this form has already been introduced in the rewriting process, it can be reused here, and no new commitment has to be computed.

B.3 From Predicates to Σ-Protocols

The construction proposed in this section has to be performed for each conjunctive clause, i.e., for each subformula of ϕ which only contains Boolean ANDs:

$$\forall\, \omega_1 \in \mathcal{I}^*(m_{\omega_1}), \ldots, \omega_n \in \mathcal{I}^*(m_{\omega_n}) \,:\, \exists\, \chi_1 \in \mathcal{I}^*(m_{\chi_1}), \ldots, \chi_{n'} \in \mathcal{I}^*(m_{\chi_{n'}}) \,:\, \bigwedge_{j=1}^{v} y_j = \prod_{i=1}^{u_j} g_{ji}^{l_{ji}(\omega_1,\ldots,\omega_n,\chi_1,\ldots,\chi_{n'})} \tag{3}$$

Consider next a relation as (3). We now construct a corresponding Σ-protocol, which in addition proves that the integer commitments $\tilde{\mathfrak{y}}_i$ are indeed well formed. The protocol therefore takes $\tilde{\sigma} = (\tilde{\mathfrak{n}}, \tilde{\mathfrak{g}}, \tilde{\mathfrak{h}})$ as additional input.

▷ In its first round, the prover computes $(t, r) = \mathsf{P}_1(y, w, x, \tilde{\sigma})$ as follows:
- It computes $\{\tilde{\mathfrak{w}}_i\}_{i=1}^n, \{\tilde{\mathfrak{y}}_i\}_{i=1}^n$ as in §B.2 by using $\tilde{\sigma}$.
- It draws randomnesses $r_i \in_R [-2^{l+k} m_{\omega_i}, 2^{l+k} m_{\omega_i}]$ for $i = 1, \ldots, n$, where l is a security parameter controlling the tightness of the zero-knowledge property. It further draws $r'_i \in_R [-2^{l+k} m_{\chi_j}, 2^{l+k} m_{\chi_j}]$ for $j = 1, \ldots, n'$ and $\tilde{\mathfrak{r}}_i \in_R [-2^{l+k} \lfloor \tilde{\mathfrak{n}}/4 \rfloor, 2^{l+k} \lfloor \tilde{\mathfrak{n}}/4 \rfloor]$ for all i;
- it computes commitments for each y_j by setting $t_j = \prod_{i=1}^{u_j} g_{ij}^{l_{ij}(r_1,\ldots,r_n,r'_1,\ldots,r'_{n'})}$ for $j = 1, \ldots, v$; further, commitments for all $\tilde{\mathfrak{y}}_i$ are computed as $\tilde{\mathfrak{t}}_i = \tilde{\mathfrak{g}}^{r_i} \tilde{\mathfrak{h}}^{\tilde{\mathfrak{r}}_i}$.
- it outputs $r = (r_1, \ldots, r_n, r'_1, \ldots, r'_{n'}, \tilde{\mathfrak{r}}_1, \ldots, \tilde{\mathfrak{r}}_n, \tilde{\mathfrak{w}}_1, \ldots, \tilde{\mathfrak{w}}_n)$ and $t = (t_1, \ldots, t_v, \tilde{\mathfrak{t}}_1, \ldots, \tilde{\mathfrak{t}}_v, \tilde{\mathfrak{y}}_1, \ldots, \tilde{\mathfrak{y}}_n)$.

▷ Upon receiving $c \in \mathcal{C}$, the prover computes $s = \mathsf{P}_2(y, w, x, c, r, \tilde{\sigma})$ as follows:
- It sets $s_i = r_i + c\omega_i$ for $i = 1, \ldots, n$ and $s'_j = r'_j + c\chi_j$ for $j = 1, \ldots, n'$;
- it computes $\tilde{\mathfrak{s}}_i = \tilde{\mathfrak{r}}_i + c\tilde{\mathfrak{w}}_i$ for $i = 1, \ldots, n$;
- it outputs $s = (s_1, \ldots, s_n, s'_1, \ldots, s'_{n'}, \tilde{\mathfrak{s}}_1, \ldots, \tilde{\mathfrak{s}}_n)$.

▷ The algorithm V outputs 1 if and only if the following five conditions are satisfied:
- For all $j = 1, \ldots, u$ it holds that $t_j y_j^c = \prod_{i=1}^{u_j} g_{ji}^{l_{ji}(s_1,\ldots,s_n,s'_1,\ldots,s'_{n'})}$, and
- for all $i = 1, \ldots, n$ it holds that $\tilde{\mathfrak{t}}_i \tilde{\mathfrak{y}}_i^c = \tilde{\mathfrak{g}}^{s_i} \tilde{\mathfrak{h}}^{\tilde{\mathfrak{s}}_i}$, and
- for all $i = 1, \ldots, n$ we have $s_i \in [-2^{l+k} m_{\omega_i} - (2^k - 1) m_{\omega_i}, 2^{l+k} m_{\omega_i} + (2^k - 1) m_{\omega_i}]$;

– for all $j = 1, \ldots, n'$ we have $s'_j \in [-2^{l+k} m_{\chi_j} - (2^k - 1)m_{\chi_j}, 2^{l+k} m_{\chi_j} + (2^k - 1)m_{\chi_j}]$;
– for all $i = 1, \ldots, n$ we have $\tilde{\mathfrak{s}}_i \in [-2^{l+k}\lfloor \tilde{\mathfrak{n}}/4 \rfloor - (2^k - 1)\lfloor \tilde{\mathfrak{n}}/4 \rfloor, 2^{l+k}\lfloor \tilde{\mathfrak{n}}/4 \rfloor + (2^k - 1)\lfloor \tilde{\mathfrak{n}}/4 \rfloor]$.

Of course, integer commitments for witnesses which are not occurring in the considered subformula can be ignored in this construction.

From the third step of the verifier's algorithm, and from the standard knowledge extractor for Σ-protocols [27], one can infer that the prover supplied witnesses satisfying $\omega_i^* \in [-2^{l+k} m_{\omega_i} - (2^k - 1)m_{\omega_i}, 2^{l+k} m_{\omega_i} + (2^k - 1)m_{\omega_i}]$, while we insist that honest parties supply witnesses $\omega_i \in \mathcal{I}(m_{\omega_i})$. We thus get that the parameter t introduced in §3 is given by

$$t := 2^{l+k} + (2^k - 1) \approx 2^{l+k} .$$

If for some witness we have $\omega_i \in \mathbb{Z}_q$, all intervals in the construction above can just be replaced by $\mathbb{Z}_q$, and accordingly the computation of s_i in P_2 is also modulo q. It is easy to see that in this case we obtain $t = 1$. In particular we note that t only depends on the security parameters l and k, and on whether the group order is known or not, but not on the concrete group or the value of m_ω.

B.4 Combining Σ-Protocols by Boolean Connectives

For self-containment, we want to give a brief overview of how Σ-protocols can be combined by the Boolean ANDs and ORs. More advanced approaches for Boolean formulae of certain types are given in [24].

If the predicate ϕ is built from atomic predicates by nested ANDs and ORs, then the following composition techniques have to be applied recursively. For instance, if ψ_1, ψ_2, ψ_3 are the atomic predicates, and ϕ is given by $\phi = (\psi_1 \wedge \psi_2) \vee \psi_3$, the AND composition is performed to obtain a Σ-protocol for $\psi_1 \wedge \psi_2$, which is then connected to ψ_3 by the OR composition technique.

Composing Σ-protocols by Boolean ANDs. Let be given Σ-protocols $((\mathsf{P}_{i1}, \mathsf{P}_{i2})(\omega_i), \mathsf{V}_i)(y_i)$ for $i = 1, \ldots, n$, and let one be interested in proving knowledge of *all* witnesses $\omega_1, \ldots, \omega_n$ simultaneously. This can be achieved by running the following Σ-protocol:

▷ In its first move, the prover performs the following computations.
  – It sets $(t_i, r_i) := \mathsf{P}_{i1}(y_i, \omega_i)$ for all $i = 1, \ldots, n$.
  – It sets $t = (t_1, \ldots, t_n)$ and $r = (r_1, \ldots, r_n)$.
▷ Upon receiving $c \in \mathcal{C}$, the prover computes its response as follows.
  – It sets $s_i := \mathsf{P}_{i2}(y_i, \omega_i, c, r_i)$ for all $i = 1, \ldots, n$.
  – It output $s := (s_1, \ldots, s_n)$
▷ The verifier accepts the proof, if and only if $\mathsf{V}(y_i, t_1, c, s_i) = 1$ for all i.

Composing Σ-protocols by Boolean ORs. Proving knowledge of *at least one* out of a set $\{\omega_1, \ldots, \omega_n\}$ of witnesses is a bit more involved. Let therefore S_i denote the simulator of the i^{th} Σ-protocol to combine. The following construction then realizes an OR composition:

▷ In its first move, the prover
  – For the known secret ω_j, set $(t_j, r_j) := \mathsf{P}_{j1}(y_i, \omega_j)$.
  – Set $(t_i, c_i, s_i) = \mathsf{S}^i(y_i, c_i)$, set $r_i := (c_i, s_i)$ for all $i \neq j$.
  – Output $t = (t_1, \ldots, t_n)$ and $r = (r_1, \ldots, r_n)$.

▷ Upon receiving $c \in_R \mathcal{C}$, the prover proceeds as follows.
  – It sets $c_j = (c \oplus \bigoplus_{i \neq j} c_i)$.
  – It computes $s_j = \mathsf{P}_{j2}(y_j, \omega_j, c_j, r_j)$.
  – It outputs $s = (s_1, \ldots, s_n, c_1, \ldots, c_n)$.
▷ The verifier accepts the proof, if and only if the following conditions are satisfied:
  – The challenge c_j was computes correctly, i.e., $c = \bigoplus_{i=1}^{n} c_i$, and
  – all triples (t_i, c_i, s_i) are accepting communication transcripts, i.e., $\mathsf{V}_i(y_i, t_i, c_i, s_i) = 1$ for all $i = 1, \ldots, n$.

B.5 From Σ-Protocols to Ω-Protocols

As discussed in §A.1, Σ-protocols are not applicable for ZK proofs in the UC model, as their security guarantees rely on rewindable blackbox access to malicious provers. We therefore next explain how the obtained Σ-protocols can be transformed into efficient Ω-protocols, cf. Definition A.2.

The idea of the construction is to Paillier encrypt the witnesses ω_i, and to send these encryptions $\hat{\mathfrak{y}}_i$ to the verifier. Then it is proved that the values inside the $\hat{\mathfrak{y}}_i$ are satisfying the relation (3). Using the integer commitments computed in §B.2, the problem of different orders of the groups used in the predicate and for the Paillier encryptions can be circumvent. Now, the private key of the Paillier scheme forms the trapdoor required in Definition A.2: if it is known, the witnesses can efficiently be extracted by decrypting the $\hat{\mathfrak{y}}_i$.

The common reference string $\bar{\sigma}$ of the resulting Ω-protocol is given by the CRS of the underlying Σ-protocol, and a public key of the Paillier encryption scheme [54]. That is, it is given by:

▷ A common reference string $\tilde{\sigma} = (\tilde{\mathfrak{n}}, \tilde{\mathfrak{g}}, \tilde{\mathfrak{h}})$ as in §B.2.
▷ a pair $(\hat{\mathfrak{n}}, \hat{\mathfrak{g}})$, where $\hat{\mathfrak{n}}$ is a safe RSA modulus, and $\hat{\mathfrak{g}} \in \mathbb{Z}^*_{\hat{\mathfrak{n}}^2}$ such that the order of $\hat{\mathfrak{g}}$ is a multiple of $\hat{\mathfrak{n}}$. If the prime factors of $\hat{\mathfrak{n}}$ are of equivalent length, one can choose $\hat{\mathfrak{g}} = \hat{\mathfrak{n}} + 1$. Further, we require that $\frac{\hat{\mathfrak{n}}}{2} > \max_i(tm_{\omega_i})$.

The trapdoor $\bar{\tau}$ consists of the factorizations of $\tilde{\mathfrak{n}}$ and $\hat{\mathfrak{n}}$.

For describing the prover's and verifier's algorithms, we denote the algorithms of the underlying Σ-protocol be given by $(\mathsf{P}_1^{\Sigma}, \mathsf{P}_2^{\Sigma})$ and V^{Σ}, respectively.

▷ In its first step, the prover computes $(t, r) = \mathsf{P}_1(y, w, x, \bar{\sigma})$ as follows. It computes
  – $(t^{\Sigma}, r^{\Sigma}) = \mathsf{P}_1^{\Sigma}(y, w, x, \tilde{\sigma})$;
  – Paillier-encryptions for all ω_i. That is, it draws $\hat{\mathfrak{w}}_i \in_R \mathbb{Z}^*_{\hat{\mathfrak{n}}}$ and sets $\hat{\mathfrak{y}}_i = \hat{\mathfrak{g}}^{\omega_i} \hat{\mathfrak{w}}_i^{\hat{\mathfrak{n}}} \mod \hat{\mathfrak{n}}^2$;
  – further, it draws $\hat{\mathfrak{r}}_i \in_R \mathbb{Z}^*_{\hat{\mathfrak{n}}}$ and computes $\hat{\mathfrak{t}}_i = \hat{\mathfrak{g}}^{\bar{r}_i} \hat{\mathfrak{r}}_i^{\hat{\mathfrak{n}}}$;
  – it sets $\bar{\mathfrak{t}}_i = \tilde{\mathfrak{g}}^{\bar{r}_i} \tilde{\mathfrak{h}}^{\bar{\mathfrak{r}}_i}$ for $\bar{\mathfrak{r}}_i \in_R [-2^{l+k}\lfloor \tilde{\mathfrak{n}}/4 \rfloor, 2^{l+k}\lfloor \tilde{\mathfrak{n}}/4 \rfloor]$;
  – Finally, it sets $t = (t^{\Sigma}, \{\hat{\mathfrak{y}}_i\}_{i=1}^n, \{\hat{\mathfrak{t}}_i\}_{i=1}^n, \{\bar{\mathfrak{t}}_i\}_{i=1}^n)$, and $r = (r^{\Sigma}, \{\hat{\mathfrak{w}}_i\}_{i=1}^n, \{\bar{\mathfrak{r}}_i\}_{i=1}^n, \{\hat{\mathfrak{r}}_i\}_{i=1}^n)$.
▷ Upon receiving a challenge $c \in_R \mathcal{C}$, the prover computes $s = \mathsf{P}_2(y, w, c, r, \bar{\sigma})$ as follows. It computes
  – $s^{\Sigma} = \mathsf{P}_2^{\Sigma}(y, w, x, c, r^{\Sigma}, \tilde{\sigma})$;
  – $\hat{\mathfrak{s}}_i = \hat{\mathfrak{r}}_i \hat{\mathfrak{w}}_i^c$ for all i;
  – $\bar{s}_i = \bar{r}_i + c\omega_i$ for all $i = 1, \ldots, n$.
  – $\tilde{\mathfrak{s}}_i = \bar{\mathfrak{r}}_i + c\tilde{\mathfrak{w}}_i$ for all $i = 1, \ldots, n$.
  – It outputs $s = (s^{\Sigma}, \{\hat{\mathfrak{s}}_i\}_{i=1}^n, \{\bar{s}_i\}_{i=1}^n, \{\bar{\mathfrak{s}}_i\}_{i=1}^n)$.
▷ The verifier accepts, i.e., $\mathsf{V}(y, t, c, s, \bar{\sigma}) = 1$, if and only if the following conditions are satisfied:
  – $\mathsf{V}^{\Sigma}(y, t^{\Sigma}, c, s^{\Sigma}, \tilde{\sigma}) = 1$;
  – the $\hat{\mathfrak{y}}_i$ are valid Paillier-encryptions of the witnesses, i.e., $\hat{\mathfrak{t}}_i \hat{\mathfrak{y}}_i^c = \hat{\mathfrak{g}}^{\bar{s}_i} \hat{\mathfrak{s}}_i^{\hat{\mathfrak{n}}}$ for all $i = 1, \ldots, n$;
  – for all $i = 1, \ldots, n$ we have $\tilde{\mathfrak{s}}_i \in [-2^{l+k}\lfloor \tilde{\mathfrak{n}}/4 \rfloor - (2^k - 1)\lfloor \tilde{\mathfrak{n}}/4 \rfloor, 2^{l+k}\lfloor \tilde{\mathfrak{n}}/4 \rfloor + (2^k - 1)\lfloor \tilde{\mathfrak{n}}/4 \rfloor]$;
  – for all $i = 1, \ldots, n$ it holds that $\bar{\mathfrak{t}}_i \tilde{\mathfrak{y}}_i^c = \tilde{\mathfrak{g}}^{\bar{s}_i} \tilde{\mathfrak{h}}^{\tilde{\mathfrak{s}}_i}$, and

For more details on this construction we refer to the original work of Garay et al. [32,33].

B.6 Optimizations

We briefly want to discuss an optimization of the techniques presented so far in the case that the predicate ϕ does not contain any Boolean OR connectives. Namely, in §B.2, a single joint integer commitment can be computed for all witnesses as $\tilde{\mathfrak{y}} = \tilde{\mathfrak{g}}_1^{\omega_1} \dots \tilde{\mathfrak{g}}_n^{\omega_n} \tilde{\mathfrak{h}}^{\tilde{\mathfrak{w}}}$, where $\tilde{\mathfrak{w}} \in_R [-\lfloor \tilde{\mathfrak{n}}/4 \rfloor, \lfloor \tilde{\mathfrak{n}}/4 \rfloor]$. This joint integer commitment can then be used in §B.3 instead of $\tilde{\mathfrak{y}}_1, \dots, \tilde{\mathfrak{y}}_n$. Further, the techniques in §B.3 and §B.5 can be combined into one step, which avoids proving knowledge of the preimage of $\tilde{\mathfrak{y}}$ twice.

Similar optimizations can be performed in the case that, in a Boolean formula containing ORs, every secret is only used within one conjunctive term, i.e., no two atomic predicates connected by an OR are using a shared secret ω_i. In this case, the above optimization can be done for each conjunctive subformula of ϕ.

B.7 From Ω-Protocols to UC-ZK Protocols

The protocols obtained so far admit some attacks which have to be prevented when aiming for protocols realizing the ideal functionality $\mathcal{F}_{\text{ZK}}^{R,R'}$:

- ▷ First, Ω-protocols are only zero-knowledge against honest verifiers. That is, if the verifier does not choose its challenge c uniformly from $\mathcal{C}$, but, e.g., depending on the commitment t, there does not exist an efficient algorithm simulating protocol transcripts any more. This issue can be solved by not sending t in the clear in the prover's first move, but by changing the protocol flow as follows: the prover first sends a commitment to t to the verifier, who then sends a challenge c back to the prover. The prover now responses with the response s, together with an opening of the commitment (including t) to the verifier, who additionally checks this opening is valid.
- ▷ Second, a malicious party could run two instances of the Ω-protocol in parallel, acting as a verifier in the first, and acting as a prover in the second instance. By simply forwarding the commitment from the first instance in the second, using the challenge from the second instance in the first, and again by forwarding the response from the first instance in the second, it could convince a verifier in the second instance that it knows the secret witnesses without actually doing so. This problem can be tackled by introducing a tag, which binds each message to a unique prover/verifier pair.
- ▷ Finally, in the constructed Ω-protocol the public value y is given as a common input to both parties, whereas in the ideal functionality $\mathcal{F}_{\text{ZK}}^{R,R'}$ the verifier only learns y after the proof has been performed.

The following construction realizes the above observation, and shows how Ω-protocols can be transformed into protocols that UC-emulate the ideal functionality $\mathcal{F}_{\text{ZK}}^{R,R'}$. The construction was first proposed in [49], with the committed proof idea stemming from [37], and is based on the Digital Signature Algorithm (DSA) [53]. Although this construction works independently from the underlying Ω-protocol, using the techniques from §B.5 is crucial for the optimizations presented in §4 to work.

The presented UC-ZK protocol is secure for adaptive corruptions, assuming that data can be erased securely and efficiently. If this is not the case, [49] also show that the resulting protocol is secure in the static corruptions model. The protocol uses a common reference string σ which is given by $(\bar{\sigma}, \mathfrak{g}, \mathfrak{p}, \mathfrak{q}, \mathfrak{y})$, where

- ▷ $\bar{\sigma}$ is the common reference string of the underlying Ω-protocol, as described at the onset of §B.5, and

▷ $(\mathfrak{g}, \mathfrak{p}, \mathfrak{q}, \mathfrak{y})$ is a public DSA key. That is, $\mathfrak{p}, \mathfrak{q}$ are prime such that $\mathfrak{q} | (\mathfrak{p} - 1)$. Further, $\mathfrak{g}, \mathfrak{y} \in \mathbb{Z}_{\mathfrak{p}}$ with $\operatorname{ord} \mathfrak{g} = \mathfrak{q}$ and $\mathfrak{y} = \mathfrak{g}^{\mathfrak{x}}$ for some secret $\mathfrak{x} \in \mathbb{Z}_q$.

The trapdoor τ of the protocol is given by $\tau = (\bar{\tau}, \mathfrak{x})$, where $\bar{\tau}$ is the trapdoor of the Ω-protocol, and $\mathfrak{x}$ is the secret discrete logarithm of y in base g.

The protocol still consists of three messages being exchanged, and makes use of a cryptographic hash function H. The prover's and verifier's algorithms are detailed in the following:

▷ In its first move, the prover performs the following computations. It
  – computes $(t^{\Omega}, r^{\Omega}) = \mathsf{P}_1^{\Omega}(y, w, x, \bar{\sigma})$ as in the underlying Ω-protocol;
  – sets $\mathsf{tag} = (PID_P, PID_V)$, and $m = H(y \| \tilde{\mathfrak{y}} \| t^{\Omega})$;
  – draws $\mathfrak{a}, \mathfrak{b} \in_R \mathbb{Z}_{\mathfrak{q}}$ and computes $\mathfrak{g}' = \mathfrak{g}^{\mathfrak{a}}$ and $\mathfrak{h} = \mathfrak{g}^{H(\mathsf{tag})} \mathfrak{y}^{\mathfrak{g}'}$. Finally, it sets $\mathfrak{c} = \mathfrak{g}'^{\mathfrak{b}} \mathfrak{h}^m$, and
  – sends $(\mathfrak{g}', \mathfrak{c})$ to the verifier.

▷ Upon receiving $c \in \mathcal{C}$, the prover
  – computes $s^{\Omega} = \mathsf{P}_2^{\Omega}(y, w, x, c, r^{\Omega}, \bar{\sigma})$,
  – deletes *all local data* stemming form the protocol, in particular r^{Ω} and all quantities introduced in the reformulation in §B.1, and
  – sends $(y, \tilde{\mathfrak{y}}, t^{\Omega}, s^{\Omega}, \mathfrak{b})$ to the verifier.

▷ The verifier accepts the protocol run, if and only if the following two conditions are satisfied:
  – $\mathsf{V}^{\Omega}(y, t^{\Omega}, c, s^{\Omega}, \bar{\sigma}) = 1$;
  – it holds that $\mathfrak{c} = \mathfrak{g}'^{\mathfrak{b}} \mathfrak{h}^m$, where $\mathsf{tag} = (PID_P, PID_V)$, $m = H(y \| \tilde{\mathfrak{y}} \| t^{\Omega})$ and $\mathfrak{h} = \mathfrak{g}^{H(\mathsf{tag})} \mathfrak{y}^{\mathfrak{g}'}$.

Theorem B.1. *Let* spec *be a proof specification without any quantities being quantified by* $\exists$*, and let* $R = R(\mathsf{spec})$*,* $R' = R'(\mathsf{spec})$*, and* $\pi = \pi(\mathsf{spec})$*. Then* π *UC-realizes* $\mathcal{F}_{\mathrm{ZK}}^{R,R'}$ *with respect to adaptive corruptions, assuming that securely erasing data is possible. If this is not the case, it still UC-realizes* $\mathcal{F}_{\mathrm{ZK}}^{R,R'}$ *with respect to static corruptions.*

The proof of this theorem follows immediately from [49]. What is actually shown there is that the resulting protocols multi-realize $\mathcal{F}_{\mathrm{ZK}}^{R,R'}$ with joint access to the common reference string σ in the $\mathcal{F}_{\mathrm{ach}}$-hybrid model [21].

B.8 Estimating the Computational Costs

When designing protocols it is often helpful if the computational costs of a protocol can already be roughly estimated at an early stage. We thus want to support a protocol designer by listing the accruing costs of a UC-ZK protocol here. The only compilation step that has to be done before being able to read off the costs from a proof specification in the basic language is to resolve all polynomial relations in the exponents of atomic predicates to at most linear ones.

The computational costs of the prover are now given by:

▷ 2 evaluations of the Paillier-homomorphism (for the encryption itself and the proof of its correctness) per secret witness quantified by $\aleph$.
  Typical length of modulus: 4096 bit ($\hat{\mathfrak{n}}^2$ for a 2048 bit safe RSA modulus $\hat{\mathfrak{n}}$)

▷ For predicates ϕ of the form (1):
  – 2 computations of integer commitments of the form $g^w h^r$ (for the commitment itself and the proof of its correctness) per secret witness quantified by $\aleph$.
  – 1 integer commitment of the form $g^w h^r$ per secret witness quantified by $\aleph$ in each conjunctive clause.

The equality testing functionality $\mathcal{F}_{\text{eq}}$

1. Wait for an input (`left-input`, u) from P and (`right-input`, v) from Q, and send `left-input` respectively `right-input` to $\mathcal{A}$.
2. Wait for a message `lock` from $\mathcal{A}$ and set $res = 1$ if $u = v$, and $res = 0$ otherwise.
3. Wait for a message `deliver-left` from $\mathcal{A}$ and send (`return`, res) to P. Wait for a message `deliver-right` from $\mathcal{A}$ and send (`return`, res) to Q.

Corruption rules:

- ▷ If P (Q) gets corrupted between Steps 1 and 2, $\mathcal{A}$ is given u (v) and is allowed to change this value at any time before Step 2.
- ▷ If P (Q) gets corrupted between Steps 2 and 3, $\mathcal{A}$ is given u (v).

Fig. 6: The equality testing functionality $\mathcal{F}_{\text{eq}}$.

If the optimization of §B.6 can be applied:

- 2 computations of multi integer commitments of the form $g_1^{w_1} \dots g_n^{w_n} h^r$ (for the commitment itself and the proof of its correctness) for each conjunctive clause in ϕ, where the w_i are only quantities of which knowledge has to be proved.

Typical length of modulus: 2048 bit

- ▷ 1 evaluation of each atomic predicate in the predicate ϕ.
  Typical length of modulus: depending on proof goal
- ▷ 3 exponentiations in the DSA group specified by the common reference string.
  Typical length of modulus: 1024 bit (cf. [53])

The costs of the verifier can be estimated similarly by inspecting the constructions given in §B.2 to §B.7. In particular one can see here that the Paillier-encryptions can be avoided for all values quantified by $\exists$, which typically poses the most expensive part of the proof because of the lengths of its modulus.

C An Example

In this section we give an example how our composition theorem can be applied in the analysis of higher-level protocols. We therefore chose the ideal equality testing functionality $\mathcal{F}_{\text{eq}}$, which expects inputs u from P, and v from Q and which outputs 1 if and only if $u = v$, cf. Figure 6.

The ideal functionality as well as the following protocol realizing it are taken from [11], where they are used in the context of credential authenticated identification. The protocol assumes that the inputs α and β of P and Q, respectively, are encoded as elements of $\mathbb{Z}_q$ for some prime number q. Furthermore, it makes use of a group $\mathcal{G}$ of order q, in which the Decisional Diffie-Hellman problem is assumed to be computationally infeasible, and a generator g of $\mathcal{G}$.

On a high level, the idea of the protocol is the following. First, P computes an encryption of g^α under the semantically secure version of the Cramer-Shoup encryption schemes [25,37]. Then, Q uses the homomorphic property of the scheme to obtain a random encryption of $g^{\sigma(\alpha-\beta)}$ for some random $\sigma \in \mathbb{Z}_q$. Finally, P decrypts this value and raises it to a random power ζ, yielding $g^{\zeta\sigma(\alpha-\beta)}$, which is equal to 1 if and only if $\alpha = \beta$.

All messages are accompanied by proofs showing that they were computed correctly. For these proofs, the gullible zero-knowledge functionality $\mathcal{F}_{\text{gZK}}$ for the respective relations is used.

1. P computes:
   $h \in_R \mathcal{G}$, $\chi_1, \chi_2, \rho \in_R \mathbb{Z}_q$
   $c = g^{\chi_1} h^{\chi_2}$, $u_1 = g^\rho$, $u_2 = h^\rho$, $e = g^\alpha h^\rho$,
   and proves the following statement to Q using $\mathcal{F}_{\text{gZK}}^{R,R'}$:

$$\curlywedge\, \alpha \in \mathbb{Z}_q \,:\, \exists\, \rho \in \mathbb{Z}_q \,:\, u_1 = g^\rho \wedge u_2 = h^\rho \wedge e = g^\alpha c^\rho \,.$$

   Note that h, c, u_1, u_2, e are delivered to Q via $\mathcal{F}_{\text{gZK}}^{R,R'}$ after erasing ρ.
2. Q computes:
   $\sigma \in_R \mathbb{Z}_q \setminus \{0\}$, $\tau \in_R \mathbb{Z}_q$,
   $\tilde{u}_1 = u_1^\sigma g^\tau$, $\tilde{u}_2 = u_2^\sigma h^\tau$, $\tilde{e} = e^\sigma g^{-\beta\sigma} c^\tau$,
   and proves the following statement to P using $\mathcal{F}_{\text{gZK}}^{R,R'}$:

$$\curlywedge\, \beta \in \mathbb{Z}_q \,:\, \exists\, \sigma, \tau \in \mathbb{Z}_q \,:\, \tilde{u}_1 = u_1^\sigma g^\tau \wedge \tilde{u}_2 = u_2^\sigma h^\tau \wedge \tilde{e} = e^\sigma g^{-\beta\sigma} c^\tau \wedge \gcd(\sigma, q) = 1 \,.$$

   Note that $\tilde{u}_1, \tilde{u}_2, \tilde{e}$ are delivered to P via $\mathcal{F}_{\text{gZK}}^{R,R'}$ after erasing σ, τ.
3. P computes:
   $\zeta \in_R \mathbb{Z}_q \setminus \{0\}$,
   $d = \tilde{e}^\zeta \tilde{u}_1^{-\zeta\chi_1} \tilde{u}_2^{-\zeta\chi_2}$,
   and proves the following statement to Q using $\mathcal{F}_{\text{gZK}}^{R,R'}$:

$$\exists\, \chi_1, \chi_2, \zeta \in \mathbb{Z}_q \,:\, c = g^{\chi_1} h^{\chi_2} \wedge d = \tilde{e}^\zeta \tilde{u}_1^{-\zeta\chi_1} \tilde{u}_2^{-\zeta\chi_2} \wedge \gcd(\zeta, q) = 1 \,.$$

   Again, note that d is delivered to Q via $\mathcal{F}_{\text{gZK}}^{R,R'}$ after erasing χ_1, χ_2, ζ.
4. After erasing all local data, both parties output 1 if $d = 1$, and 0 otherwise.

Theorem C.1. *The $\mathcal{F}_{\text{sch}}$-hybrid protocol π described above UC-emulates $\mathcal{F}_{\text{eq}}$ with respect to adaptive corruptions, assuming that the Decisional Diffie-Hellman Problem is hard in $\mathcal{G}$, and that securely erasing data is possible.*

A sketch of the following proof without the $\exists$-optimization in the proof protocols can also be found in [11].

Proof. By Theorem 4.3 we have to show that the $\mathcal{F}_{\text{sch}}, \mathcal{F}_{\text{gZK}}$-hybrid version of π UC-emulates $\mathcal{F}_{\text{eq}}$ with respect to the dummy adversary and nice environments. That is, we have to show that for every nice environment $\mathcal{Z}$, there exists an efficient simulator $\mathcal{S}$ such that for the dummy adversary $\mathcal{A}^*$ we have that

$$\text{EXEC}(\pi, \mathcal{A}^*, \mathcal{Z}) \approx \text{EXEC}(\mathcal{F}_{\text{eq}}, \mathcal{S}, \mathcal{Z}) \,.$$

We start with some basic observations:

- ▷ We may assume that the environment $\mathcal{Z}$ always requires $\mathcal{A}^*$ to send correct statements to $\mathcal{F}_{\text{gZK}}$, and not only with overwhelming probability. It can easily be seen that the probability distributions of $\text{EXEC}(\pi, \mathcal{A}^*, \mathcal{Z})$ and $\text{EXEC}(\mathcal{F}_{\text{eq}}, \mathcal{S}, \mathcal{Z})$ are only altered negligibly by this modification.
- ▷ Simulation is trivial in the case that neither P nor Q are corrupted, as we are using secure channels. Namely, in this case, the adversary only sees notifications without content. All messages received from the adversary (i.e., `lock`, `deliver-left`, `deliver-right`) by $\mathcal{S}$ are just forwarded to its internal copy of $\mathcal{F}_{\text{eq}}$.

▷ In the case where (at least) one of the parties gets corrupted, we can assume that either P or Q has already been corrupted at the beginning of the protocol run: if this is not the case, the simulator can just start the party's execution using its inputs until the point it actually gets corrupted. This works because of the usage of secure channels, and the structure of $\mathcal{F}_{\text{gZK}}$ (the statement to be proved is only revealed in the last step). However, we have to deal with the possibility that the other party gets corrupted at any time during the execution of the protocol.

Let us now describe the simulator for the case where Q is corrupt.

1. If P is already corrupted before the `lock` message is sent by the adversary, $\mathcal{S}$ obtains P's input α to $\mathcal{F}_{\text{eq}}$, and proceeds as described in the protocol. Otherwise, $\mathcal{S}$ chooses

$$\rho_1, \mu_1, \mu_2 \in_R \mathbb{Z}_q, \quad \omega \in_R \mathbb{Z}_q \setminus \{0\}, \quad \rho_2 \in_R \mathbb{Z}_q \setminus \{\rho_1\}\,,$$

   and uses the fact that the semantically secure version of the Cramer-Shoup encryption scheme is receiver-non-committing to compute a fake encryption of the unknown α which can later be opened correctly to any α, cf. [37]:

$$h = g^{\omega}, \quad u_1 = g^{\rho_1}, \quad u_2 = g^{\rho_2}, \quad c = g^{\mu_1}, \quad e = g^{\mu_2}\,.$$

   Whenever P gets corrupted in the subsequent, $\mathcal{S}$ obtains α from $\mathcal{F}_{\text{eq}}$ and computes a valid internal state χ_1, χ_2, α of P consistent to g, h, u_1, u_2, c, e by computing χ_1, χ_2 such that

$$\chi_1 + \omega\chi_2 = \mu_2 \quad \text{and} \quad \rho_1\chi_1 + \rho_2\omega\chi_2 = \mu_1 - \alpha\,.$$

2. As Q is corrupt, $\mathcal{S}$ gets Q's input β to $\mathcal{F}_{\text{gZK}}$, as well as the outputs $\tilde{u}_1, \tilde{u}_2, \tilde{e}$. Then, $\mathcal{S}$ submits β as Q's inputs to its internal copy of $\mathcal{F}_{\text{eq}}$.
3. If P gets corrupted before the `lock` message was sent by the adversary, $\mathcal{S}$ gets α, χ_1, χ_2 as described in Step 1, and draws $\zeta \in_R \mathbb{Z}_q \setminus \{0\}$. Otherwise, $\mathcal{S}$ sets $d = 1$ if $res = 1$, and $d \in_R \mathcal{G} \setminus \{1\}$ if $res = 0$. It then requests $\mathcal{F}_{\text{eq}}$ to deliver the output to $\mathcal{Z}$ after the protocol ended.

We now have to show that the output of this simulator is indeed indistinguishable from real protocol runs.

We first note that under the Decisional Diffie-Hellman assumption for $\mathcal{G}$, the environment cannot distinguish between correctly computed tuples (g, h, u_1, u_2, e) and faked ones as has be shown in [37, Lemma 5]. By the same result, if P gets corrupted at any later point in time, (α, χ_1, χ_2) cannot be distinguished from correct openings of the encryption. The correctness of the first step of the simulation finally follows from the fact that in the protocol $\rho = \log_g u_1 = \log_h u_2$ is deleted before delivering h to Q. Namely, the simulator does not have to provide ρ to $\mathcal{Z}$ upon corruption of P, and thus $\mathcal{Z}$ cannot test whether the discrete logarithms are equal.

In the second step, as $\mathcal{Z}$ is nice by assumption, we have that $(\tilde{u}_1, \tilde{u}_2, \tilde{e})$ really satisfies the relation claimed by Q, and thus that β is indeed the correct input of $\mathcal{Z}$. Note that the plain UC-ZK functionality $\mathcal{F}_{\text{ZK}}^{R,R'}$ would also hand back σ and τ to $\mathcal{S}$ here.

As in the first step, the distribution of $(\alpha, \chi_1, \chi_2, \zeta \in_R \mathbb{Z}_q \setminus \{0\})$ in the last step is computationally close to that in the real protocol run. Similarly, as in the real protocol $d = 1$ happens if and only if $b - a = 0$ as g has prime order, and $d \in_R \mathcal{G}$ otherwise. Thus, the last step of the simulation is indistinguishable from the real protocol as well. Note again that, as χ_1, χ_2, ζ are quantified by $\exists$, they never have to be given to the adversary after calling $\mathcal{F}_{\text{gZK}}^{R,R'}$.

Even though not being symmetric, the simulation and proof work similar for the case that P starts corrupted. □

If in the first step of the above protocol we had that *knowledge* of ρ must be proved, then it is crucial that our formulation of the ideal UC-ZK functionality only delivers the statement to prove in the last step. Assume for a moment that the protocol was built using the standard formulation found, e.g., in [17,28,49], where the attacker already learns y in the first step of the ideal functionality. Then, if P gets corrupted during the proof of the first step from above, the simulator would have to reveal α, ρ satisfying the relation on the right-hand side for fixed u_1, u_2, e. However, this could not be done, because u_1, u_2 were chosen by the simulator such that $\log_g u_1 \neq \log_h u_2$. If on the other hand our ideal functionality $\mathcal{F}_{\text{ZK}}$ had been used, the simulator had the possibility to change u_1, u_2, e such that α, ρ satisfy the claimed relation, as it is not committed to any values before the last message in the functionality. While multiple further examples for such applications of $\mathcal{F}_{\text{ZK}}$ can be found in [11], we are not aware of any situations where our formulation can *not* be used instead of the standard formulation.