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. Recently, Camenisch, Kiayias, and Yung provided a specification language (the *CKY-language*) for such protocols which allows for a modular design and protocol analysis: for every zero-knowledge proof specified in this language, protocol designers are ensured that there exists an efficient protocol which indeed proves the specified statement.

However, the protocols resulting from their compilation techniques only satisfy the classical notion of ZK-PoK, which is not retained are when they used as building blocks for higher-level applications or composed with other protocols. This problem can be tackled by moving to the Universal Composability (UC) framework, which guarantees retention of security when composing protocols in arbitrary ways. While there exist generic transformations from Σ-protocols to UC-secure protocols, these transformation are often too inefficient for practice.

In this paper we introduce a specification language akin to the CKY-language and a compiler such that the resulting protocols are UC-secure and efficient. To this end, we propose an extension of the UC-framework addressing the issue that UC-secure zero-knowledge proofs are by definition 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 membership* in the UC-framework. We believe that our contributions enable the design of practically efficient protocols that are UC-secure and thus themselves can be used as building blocks.

Keywords: UC-Framework, Protocol Design, Zero-Knowledge Proof.

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

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

D.H. Lee and X. Wang (Eds.): ASIACRYPT 2011, LNCS 7073, pp. 449–467, 2011.

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., [1–3]. The so-called *Universal Composability* (UC) framework by Canetti [2] seems to be the most prevalent one. A 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 of each subprotocol can be analyzed separately, making the security of the overall protocol much easier. In particular, each (sub-)protocol needs to be analyzed only once and for all and 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. However, this is only the case for a small fraction of the proposed cryptographic schemes, such as oblivious transfer [4] and encryption- [5,6], and commitment schemes [7]. Furthermore, only very few UC-secure protocols are actually deployed in the real world, e.g., [8, 9]. We believe that one main reason for this is the high computational overhead which is often required to achieve UC-security.

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 one party to assure other parties that its actions are consistent with its internal knowledge state. The shorthand notation for such proofs, introduced in [10], has been extensively used in the past and contributed to the wide employment of ZK-PoK in cryptographic design. This notation suggests using, e.g., $\mathsf{PK}\,[(\alpha) : y = g^{\alpha}]$ to denote a proof of the discrete logarithm $\alpha = \log_g y$, and it has appeared in many works sometimes with quite complex statements, e.g., [11–23]. This informal notion was recently formalized and refined by Camenisch, Kiayias and Yung who have provided a specification language (*CKY-language*) for such protocols [24]. 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.

The realizations given by Camenisch et al. [24] are based on Σ-protocols and satisfy the classical notion of ZK-PoK but not that of UC zero-knowledge. On a high level, the problem here is that the classical notion only requires that a valid

witness can be extracted from every convincing prover given rewindable access to that prover. However, in the UC-framework this has to be possible without rewinding. While generic transformations from Σ-protocols to UC-ZK protocols are known [25], they come along with a significant computational overhead, making the resulting protocols impracticable for real-world usage.

However, the security proofs of many cryptographic protocols only require the existence of a witness, and not that the prover actually knows it. Intuitively, this should be easier to achieve than proofs of knowledge. Yet, in the UC-framework zero-knowledge proofs are always proofs of *knowledge*. This is because otherwise the ideal functionality generally could not decide whether or not a given statement is true in polynomial time. In this paper we are aiming at closing the gap between high security guarantees and modularity on the one hand, and practical usability and efficiency of the resulting protocols on the other hand.

Our Contributions. 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*), which we also incorporate into our language. Let us explain this in more detail in the next paragraphs.

A language for UC-ZK protocols. We provide an intuitive language for specifying ZK-PoK for discrete logarithms akin to the CKY-language [24] 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 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 ZK proofs are necessarily proofs of *knowledge*. However, when designing higher-level protocols, it is often sufficient to prove that some computation was done correctly, but not to show that the secret quantities are actually known. To allow protocol designers 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 *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. Roughly, 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 by the real-world protocol to be UC-secure in the standard sense.

Related Work. The UC-framework has first been introduced by Canetti [2]. The notion of Ω-protocols was introduced in [25, 26], and so far the most efficient UC-secure zero-knowledge proofs of knowledge have been proposed in [27]. Further, [28] analyzes UC-ZK in the presence of global setup [29]. The idea

of committed proofs was first mentioned in [30]. We combine the techniques of [27, 30] to compile proof specifications in our language to real protocols. In particular this allows us to realize proofs *of existence*.

A language for specifying ZK-PoK for discrete logarithms was presented by Camenisch and Stadler [10] and later refined by Camenisch et al. [24], but neither of their realizations are UC-secure. 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 [31], and in this paper we fulfill the promises given there.

Functionalities similar to $\mathcal{F}_{\text{gZK}}$ have already been used by Lindell [32,33] and Pass and Rosen [34] 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 [35, 36] 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. Finally, ideas similar to ours have also been suggested in unpublished work by Nielsen [37].

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. In this section we further show how such specifications can be compiled to actual protocols. We give several extensions to our basic language in §5 and briefly conclude in §6.

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 $\mathcal{S}$. The group of signed quadratic residues [38] for some modulus n is denoted by $\mathbb{SR}_n$. 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* [39], 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* [25], if it further takes a common reference string σ as additional input, such that when knowing a trapdoor to σ it is possible to compute the prover's secret input from any successful run of the protocol.

An Ω-protocol is said to be *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 further values depending on these witnesses. These constructions will allow for an efficiency speedup compared to using plain Ω-protocols, while often still ensuring appropriate security guarantees.

2.1 The UC-Model

We next briefly recapitulate the Universal Composability (UC) framework [2].

A *party* is a probabilistic polynomial time 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*. Two parties share the same session ID if and only if they are participants of the same instance of a protocol. Party IDs are solely used to distinguish between participants of the same protocol instance. Following [31], 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 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, performs some local computations, and hands back outputs to the parties. Ideal functionalities can be seen as formal specifications of cryptographic tasks and are secure by definition. Now, if ρ UC-emulates $\mathcal{F}$, one can infer that ρ does not leak any other information to an adversary than $\mathcal{F}$ would have, and therefore securely realizes the given task in arbitrary contexts. For a more precise description see [2].

Protocols using an ideal functionality $\mathcal{F}$ as a subroutine are called *$\mathcal{F}$-hybrid*. 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 a sender, and forwards it to a receiver. The adversary learns x, and, upon corruption of the sender, is allowed to change it before it is delivered.

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 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 $R' \supseteq R$ [31]

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 and 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 $\mathcal{F}_{\mathrm{ZK}}$ as well which will be disregarded in the remaining discussion). This approach prevents the simulator from being over-committed in our constructions, and to the best of our knowledge $\mathcal{F}_{\mathrm{ZK}}^{R,R'}$ can safely be used instead of the standard UC-ZK functionality in any application. 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 [40] (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 further elaborate on this in §3.

The same formalization of the ZK functionality was also used in [31].

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 ω, ρ 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., [14, 16].

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 [30, 41]. 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 [31], 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 [42], 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 ZK-PoK in the non-UC case which was introduced in [10]. We stress that similar to there, our notation does not only specify proof goals (i.e., what one wants to prove), but concrete protocols. 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 denoted as follows:

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

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 the F_i are integer polynomials, i.e., $F_i \in \mathbb{Z}[X_1, \ldots, X_n]$. Similar to [10], 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.

We next 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.

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 instead [38].

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.

Domains. We allow the secret values $\omega_1, \ldots, \omega_n$ to be arbitrary integers. However, for implementation issues, for each i an integer m_{ω_i} satisfying

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

is required. The value of m_{ω_i} can be chosen arbitrarily large, and is only needed for the protocols resulting from the construction in §4.1 to be statistically zero-knowledge for any $\omega_i \in \mathcal{I}(m_{\omega_i})$. They then guarantee that the prover knows witnesses in a 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 have $t \approx 2^{k+l} + 2^k - 1$, 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 modeled 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. This slightly increases the efficiency of the resulting protocols because of shorter exponents in the modular exponentiations in the protocol.

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 $\pi = \pi(\texttt{spec})$, cf. §4.1. The protocol π then UC-emulates $\mathcal{F}_{\mathrm{ZK}}^{R,R'}$, i.e., it is zero-knowledge for $(y, w) \in R$, and guaranteed the verifier that the prover supplied $(y', w') \in R'$.

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

Example 3.1 (Running Example 1). We start by resolving the condition $\omega \geq 0$ into the form (1) by rewriting it to $\omega = \sum_{i=1}^{4} \chi_i^2$ [43]. Let ω be an element of $[-T, T]$, i.e., $m_\omega = T$. Then, clearly, we have that $m_{\chi_i} = \lfloor \sqrt{T} \rfloor$ for all i. Also, for y to be blinding, we can assume that $m_\rho = \lfloor n/4 \rfloor$.

The proof goal is thus given by:

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

Example 3.2 (Running Example 2). In this case, all secret values are elements of $\mathbb{Z}_q$, where q is the order of $\mathcal{H}$. We therefore get the following proof specification:

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

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

Realizing ZK-PoK 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 Paillier encryptions of the witnesses into the proof. Now, in larger protocols, ZK-PoK are often only used to ensure that a computation was done correctly, and the simulators of these higher-level protocols do not make usage of the witnesses. For instance, in Example 2.2 proving the existence of ρ is sufficient to imply the required well-formedness of the ciphertext.

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

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 $w = f(\tilde{w})$ for a full witness $\tilde{w}$, 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 the statement is true. We now propose a framework that circumvents this problem and allows one to use proofs of existence in the UC-model.

We extend our basic language by the additional $\exists$-quantifier. For secrets quantified under $\exists$ (instead of $\nexists$) only their existence (instead of their knowledge) is proved. A generalized specification of a proof goal now looks as follows:

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

In the following we show how such specifications are compiled into protocols, and then describe the underlying theory and composition theorem which allow to use such specifications as modular building blocks in larger protocols.

4.1 Compiling Specifications to Protocols

Due to space limitations we here only give a brief overview about how protocol specifications are compiled into protocols. For a detailed description we refer to the full version of this paper [44].

- ▷ 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 [40, 43, 45–48].
- ▷ In a second step, the prover computes integer commitments $\mathfrak{y}_i$ to all secret witnesses ω_i quantified by $\nexists$.
- ▷ Next, using the technique proposed in [40], 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 being used are the same as in the $\mathfrak{y}_i$.
- ▷ Now, the different Σ-protocols are combined by the Boolean connectives as specified by the predicate ϕ [48, 49].
- ▷ As a fifth step, the Σ-protocol is transformed into an Ω-protocol [25, 26]. This is achieved by Paillier-encrypting the witnesses quantified by $\nexists$ [50], and proving that the encrypted witnesses are the same as in the $\mathfrak{y}_i$.
- ▷ Using a simulation sound trapdoor commitment [27] and the committed-proof idea of [30], one finally obtains a protocol UC-emulating $\mathcal{F}_{\text{gZK}}^{R,R'}$.

Theorem 4.1. *Let* spec *be a proof specification of the form (1), and let* $R = R(\texttt{spec})$, $R' = R'(\texttt{spec})$, *and* $\pi = \pi(\texttt{spec})$. *Then* π *UC-realizes* $\mathcal{F}_{\text{ZK}}^{R,R'}$ *with respect to adaptive corruptions, assuming that securely erasing data is possible. If this is not the case, it still UC-realizes* $\mathcal{F}_{\text{ZK}}^{R,R'}$ *with respect to static corruptions.*

The proof of this theorem is a straightforward adaption of that in [27] and is omitted due to space limitations.

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 4.2 (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 [43], we obtain the following equivalent proof specification:

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

Keeping in mind that the χ_i, ρ_i and ρ' can be computed efficiently from ω, ρ using Lagrange's Four Square Theorem and the Rabin-Shallit algorithm [51], it is easy to see that this specification is semantically equivalent to the following:

$$\nu\,\omega \in \mathcal{I}^*(T), \rho \in \mathcal{I}^*(\lfloor n/4 \rfloor) \;:\; \exists\,\{\rho_i\}_{i=1}^4 \in \mathcal{I}^*(\lfloor n/4 \rfloor), \{\chi_i\}_{i=1}^4 \in \mathcal{I}^*(\lfloor \sqrt{T} \rfloor),$$
$$\rho' \in \mathcal{I}^*((4\lfloor \sqrt{T} \rfloor + 1)\lfloor n/4 \rfloor) \;:\; y = g^{\omega} h^{\rho} \;\wedge\; \bigwedge_{i=1}^4 y_i = g^{\chi_i} h^{\rho_i} \;\wedge\; y = \prod_{i=1}^4 y_i^{\chi_i} h^{\rho'}$$

This rewriting yields a significant efficiency speedup, as only Paillier encryptions for 2 instead of 9 values are required. Overall, the prover (verifier) thus saves 14 (7) Paillier encryptions and evaluations of the integer commitment scheme. □

In this example, changing from the ν- 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 [51]. 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 4.3 (Running Example 2). The following specification is sufficient for proving the required well-formedness of the ciphertext:

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

This observation reduces the complexity of the prover's algorithms in the protocol by 2 Paillier encryptions and 2 evaluations of the integer commitment scheme (one each for their computation and their commitment in the Σ-protocol). □

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 [2].

4.2 The Gullible ZK Functionality and a Composition Theorem

In the following we describe the theoretical framework which allows protocols designers to treat specifications containing values quantified by $\exists$ (almost) as if they were quantified by ν.

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

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 appropriately.

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 gullible zero-knowledge functionality:

Definition 4.4. *Let $\mathcal{A}^*$ be the dummy adversary attacking some $\mathcal{F}_{\text{gZK}}^{R,R'}$-hybrid protocol ρ. We call an environment $\mathcal{Z}$* nice *(with respect to ρ), 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 $\mathcal{Z}$ asks $\mathcal{A}^*$ 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 $f(\tilde{w}) = w$.*

Note that the value of x submitted by a nice environment is not restricted by this definition, but only w has to be a valid partial witness.

We now define UC-emulation with respect to nice environments:

Definition 4.5. *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 (w.r.t. ρ), 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 $\text{EXEC}(\phi, \mathcal{S}, \mathcal{Z})$.

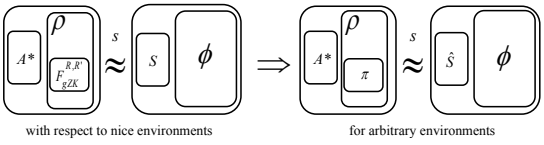

Fig. 3. Illustration of Theorem 4.6: for proving that $\rho^{\pi/\mathcal{F}_{\mathrm{gZK}}^{R,R'}}$ UC-emulates ϕ, it is sufficient to show that ρ emulates ϕ for nice environments

Note that any non-nice environment could potentially distinguish between ρ and ϕ by just submitting a false statement, which will always be accepted by $\mathcal{F}_{\mathrm{gZK}}^{R,R'}$. Informally, our special composition theorem now states that every non-nice environment can be detected if $\mathcal{F}_{\mathrm{gZK}}^{R,R'}$ is instantiated by π as described in the previous section, and thus $\rho^{\pi/\mathcal{F}_{\mathrm{gZK}}^{R,R'}}$ is secure against arbitrary environments. This allows a protocol designer to use ZK proofs of existence in a UC-compliant way, almost as if they were ZK-PoK. The theorem is illustrated in Figure 3.

Theorem 4.6. *Let* spec *be a proof specification of the form (2), and let* $R = R(\mathtt{spec})$, $R' = R'(\mathtt{spec})$, *and* $\pi = \pi(\mathtt{spec})$. *Let further* ρ *be an* $\mathcal{F}_{\mathrm{gZK}}^{R,R'}$*-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* $\rho^{\pi/\mathcal{F}_{\mathrm{gZK}}^{R,R'}}$ *UC-emulates* ϕ *(in the standard sense) with respect to adaptive corruptions if securely erasing data is possible.*

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

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 [52]. Thus, $\hat{\mathcal{S}}$ proceeds like the simulator for Theorem 4.1, cf. [30] and [27], expect for the following difference: secret values quantified by $\exists$ are given to the attacker in the real protocol π, but not in the ideal functionality $\mathcal{F}_{\mathrm{gZK}}$. This can be simulated because of the committed proof technique by choosing these secrets at random within their domains whenever necessary. Then, $\hat{\mathcal{S}}$ computes the corresponding image y' and opens the commitment made in its first message accordingly. As in [27], this is possible because of the trapdoor property of the used commitment scheme. Note

here that these values are deleted before sending out the final message, so the simulator never has to supply them after the adversary learned y.

▷ 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 theorem can be applied as follows by a protocol designer: He first designs a high-level protocol using proofs of existence as if there was a corresponding ideal functionality. Then, in the security proof, he shows that the protocol using $\mathcal{F}_{\text{gZK}}^{R,R'}$ UC-emulates a target functionality ϕ, where he may restrict himself to nice environments only. Finally, after instantiating $\mathcal{F}_{\text{gZK}}^{R,R'}$ by $\pi(\texttt{spec})$, he obtains a protocol emulating ϕ in the full UC-sense.

5 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, it might often be more convenient to declare them explicitly. Also, the requirement that all witnesses must be integers may seem overly restrictive.

To solve this problems, we next give some enhancements of our basic language. 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.

5.1 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 [43].
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 $d = \frac{b-a}{2}$.

▷ $\boldsymbol{\text{gcd}(\nu_1, \nu_2) = 1}$, 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 5.1 (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 5.1 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.

5.2 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 [53], 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 §4.1. 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 l is a security parameter.
2. Choose $\omega' \in_R \mathcal{I}(m')$, and set $u = g^{\omega'}\omega$, $y' = \psi(u)y^{-1}$, and $g' = \psi(g)$.
3. Rewrite the proof goal to $\mathsf{K}\, \omega' \in \mathcal{I}^*(m') \,:\, y' = g'^{\omega'}$, and add u to commitment of the Σ-protocol.

6 Conclusion

We presented a framework enabling the use of efficient zero-knowledge protocols in the construction of UC-secure protocols. These protocols can be specified in a unified and unambiguous notation and then generated by a compiler. To make proving security of construction that make use proof of existence protocols easy, we provide a special composition theorem. By means of two running examples we illustrated that using proofs of *existence* (as opposed to proofs of *knowledge*) can significantly reduce the computational overhead required to achieve UC-security for many practical applications without affecting security.

We believe that by reducing the costs of UCZK protocols to a practically acceptable level in many cases our result can contribute to a wider employment of UC-secure protocols in the real world.

References

1. Backes, M., Pfitzmann, B., Waidner, M.: The reactive simulatability (RSIM) framework for asynchronous systems. Information and Computation 205(12), 1685–1720 (2007)
2. Canetti, R.: Universally Composable Security: A New Paradigm for Cryptographic Protocols. In: FOCS 2001, pp. 136–145. IEEE (2001), Revised version at `http://eprint.iacr.org/2000/067`
3. Pfitzmann, B., Waidner, M.: Composition and integrity preservation of secure reactive systems. In: ACM CCS 2000, pp. 245–254 (2000)
4. Peikert, C., Vaikuntanathan, V., Waters, B.: A Framework for Efficient and Composable Oblivious Transfer. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 554–571. Springer, Heidelberg (2008)
5. Küsters, R., Tuengerthal, M.: Universally Composable Symmetric Encryption. In: Computer Security Foundations Symposium – CSF 2009, pp. 293–307. IEEE (2009)
6. Laud, P., Ngo, L.: Threshold Homomorphic Encryption in the Universally Composable Cryptographic Library. In: Baek, J., Bao, F., Chen, K., Lai, X. (eds.) ProvSec 2008. LNCS, vol. 5324, pp. 298–312. Springer, Heidelberg (2008)
7. Lindell, Y.: Highly-Efficient Universally-Composable Commitments Based on the DDH Assumption. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 446–466. Springer, Heidelberg (2011)
8. Canetti, R., Krawczyk, H.: Security Analysis of IKE's Signature-Based Key-Exchange Protocol. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 143–161. Springer, Heidelberg (2002)
9. Krawczyk, H.: HMQV: A High-Performance Secure Diffie-Hellman Protocol. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 546–566. Springer, Heidelberg (2005)
10. Camenisch, J., Stadler, M.: Efficient Group Signature Schemes for Large Groups (Extended Abstract). In: Kaliski Jr., B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 410–424. Springer, Heidelberg (1997)
11. Ateniese, G., Song, D.X., Tsudik, G.: Quasi-Efficient Revocation in Group Signatures. In: Blaze, M. (ed.) FC 2002. LNCS, vol. 2357, pp. 183–197. Springer, Heidelberg (2003)

12. Boudot, F.: Efficient Proofs that a Committed Number Lies in an Interval. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 431–444. Springer, Heidelberg (2000)
13. Bresson, E., Stern, J.: Efficient Revocation in Group Signatures. In: Kim, K.-c. (ed.) PKC 2001. LNCS, vol. 1992, pp. 190–206. Springer, Heidelberg (2001)
14. Brickell, E., Camenisch, J., Chen, L.: Direct Anonymous Attestation. In: ACM CCS 2004, pp. 132–145. ACM Press (2004)
15. Bussard, L., Roudier, Y., Molva, R.: Untraceable Secret Credentials: Trust Establishment with Privacy. In: PerCom Workshops, pp. 122–126. IEEE (2004)
16. Camenisch, J., Herreweghen, E.V.: Design and implementation of the idemix anonymous credential system. In: ACM CCS 2002, pp. 21–30. ACM Press (2002)
17. Camenisch, J., Shoup, V.: Practical Verifiable Encryption and Decryption of Discrete Logarithms. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 126–144. Springer, Heidelberg (2003)
18. Furukawa, J., Yonezawa, S.: Group Signatures with Separate and Distributed Authorities. In: Blundo, C., Cimato, S. (eds.) SCN 2004. LNCS, vol. 3352, pp. 77–90. Springer, Heidelberg (2005)
19. Nakanishi, T., Shiota, M., Sugiyama, Y.: An Efficient Online Electronic Cash with Unlinkable Exact Payments. In: Zhang, K., Zheng, Y. (eds.) ISC 2004. LNCS, vol. 3225, pp. 367–378. Springer, Heidelberg (2004)
20. Song, D.X.: Practical Forward Secure Group Signature Schemes. In: ACM CCS 2001, pp. 225–234. ACM press (2001)
21. Tang, C., Liu, Z., Wang, M.: A Verifiable Secret Sharing Scheme with Statistical zero-knowledge. Cryptology ePrint Archive, Report 2003/222 (2003), `http://eprint.iacr.org/`
22. Tsang, P.P., Wei, V.K.: Short Linkable Ring Signatures for E-Voting, E-Cash and Attestation. In: Deng, R.H., Bao, F., Pang, H., Zhou, J. (eds.) ISPEC 2005. LNCS, vol. 3439, pp. 48–60. Springer, Heidelberg (2005)
23. Tsang, P.P., Wei, V.K., Chan, T.K., Au, M.H., Liu, J.K., Wong, D.S.: Separable Linkable Threshold Ring Signatures. In: Canteaut, A., Viswanathan, K. (eds.) INDOCRYPT 2004. LNCS, vol. 3348, pp. 384–398. Springer, Heidelberg (2004)
24. Camenisch, J., Kiayias, A., Yung, M.: On the Portability of Generalized Schnorr Proofs. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 425–442. Springer, Heidelberg (2009)
25. Garay, J.A., MacKenzie, P., Yang, K.: Strengthening Zero-Knowledge Protocols Using Signatures. Journal of Cryptology 19(2), 169–209 (2006)
26. Garay, J.A., MacKenzie, P., Yang, K.: Strengthening Zero-Knowledge Protocols Using Signatures. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 177–194. Springer, Heidelberg (2003)
27. MacKenzie, P., Yang, K.: On Simulation-Sound Trapdoor Commitments. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 382–400. Springer, Heidelberg (2004)
28. Dodis, Y., Shoup, V., Walfish, S.: Efficient Constructions of Composable Commitments and Zero-Knowledge Proofs. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 515–535. Springer, Heidelberg (2008)
29. Canetti, R., Dodis, Y., Pass, R., Walfish, S.: Universally Composable Security with Global Setup. In: Vadhan, S.P. (ed.) TCC 2007. LNCS, vol. 4392, pp. 61–85. Springer, Heidelberg (2007)
30. Jarecki, S., Lysyanskaya, A.: Adaptively Secure Threshold Cryptography: Introducing Concurrency, Removing Erasures (Extended Abstract). In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 221–242. Springer, Heidelberg (2000)

31. Camenisch, J., Casati, N., Groß, T., Shoup, V.: Credential Authenticated Identification and Key Exchange. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 255–276. Springer, Heidelberg (2010)
32. Lindell, Y.: Bounded-concurrent secure two-party computation without setup assumptions. In: STOC 2003, pp. 683–692. ACM Press (2003)
33. Lindell, Y.: Protocols for Bounded-Concurrent Secure Two-Party Computation in the Plain Model. Chicago Journal of Theoretical Computer Science 2006(1), 1–50 (2006)
34. Pass, R., Rosen, A.: Bounded-Concurrent Secure Two-Party Computation in a Constant Number of Rounds. In: FOCS 2003, pp. 404–413. IEEE (2003)
35. Prabhakaran, M., Sahai, A.: New notions of security: achieving universal composability without trusted setup. In: STOC 2004, pp. 242–251. ACM Press (2004)
36. Prabhakaran, M., Sahai, A.: Relaxing Environmental Security: Monitored Functionalities and Client-Server Computation. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 104–127. Springer, Heidelberg (2005)
37. Nielsen, J.: Universally Composable Zero-Knowledge Proof of Membership. Technical report, University of Aarhus (2005)
38. Hofheinz, D., Kiltz, E.: The Group of Signed Quadratic Residues and Applications. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 637–653. Springer, Heidelberg (2009)
39. Cramer, R.: Modular Design of Secure yet Practical Cryptographic Protocols. PhD thesis. CWI and University of Amsterdam (1997)
40. Damgård, I., Fujisaki, E.: A Statistically-Hiding Integer Commitment Scheme Based on Groups with Hidden Order. In: Zheng, Y. (ed.) ASIACRYPT 2002. LNCS, vol. 2501, pp. 125–142. Springer, Heidelberg (2002)
41. Cramer, R., Shoup, V.: A Practical Public Key Cryptosystem Provably Secure against Adaptive Chosen Ciphertext Attack. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 13–25. Springer, Heidelberg (1998)
42. Canetti, R., Lindell, Y., Ostrovsky, R., Sahai, A.: Universally composable two-party and multi-party secure computation. In: STOC 2002, pp. 494–503. ACM Press (2002)
43. Lipmaa, H.: On Diophantine Complexity and Statistical Zero-Knowledge Arguments. In: Laih, C.-S. (ed.) ASIACRYPT 2003. LNCS, vol. 2894, pp. 398–415. Springer, Heidelberg (2003)
44. Camenisch, J., Krenn, S., Shoup, V.: A Framework for Practical Universally Composable Zero-Knowledge Protocols. Cryptology ePrint Archive, Report 2011/228 (2011), http://eprint.iacr.org/
45. Brands, S.: Rapid Demonstration of Linear Relations Connected by Boolean Operators. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 318–333. Springer, Heidelberg (1997)
46. Cramer, R., Damgård, I.: Zero-Knowledge Proofs for Finite Field Arithmetic or: Can Zero-Knowledge Be for Free? In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 424–441. Springer, Heidelberg (1998)
47. Fujisaki, E., Okamoto, T.: Statistical Zero Knowledge Protocols to Prove Modular Polynomial Relations. In: Kaliski Jr., B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 16–30. Springer, Heidelberg (1997)
48. Smart, N.P. (ed.): Final Report on Unified Theoretical Framework of Efficient Zero-Knowledge Proofs of Knowledge. CACE Project Deliverable (2009), http://www.cace-project.eu

49. Cramer, R., Damgård, I., Schoenmakers, B.: Proof of Partial Knowledge and Simplified Design of Witness Hiding Protocols. In: Desmedt, Y.G. (ed.) CRYPTO 1994. LNCS, vol. 839, pp. 174–187. Springer, Heidelberg (1994)
50. Paillier, P.: Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 223–238. Springer, Heidelberg (1999)
51. Rabin, M., Shallit, J.: Randomized Algorithms in Number Theory. Communications in Pure and Applied Mathematics 39(S1), 239–256 (1986)
52. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof-systems. In: STOC 1985, pp. 291–304. ACM Press (1985)
53. Camenisch, J., Lysyanskaya, A.: Signature Schemes and Anonymous Credentials from Bilinear Maps. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 56–72. Springer, Heidelberg (2004)