# On the Design and Implementation of Efficient Zero-Knowledge Proofs of Knowledge*

Endre Bangerter[1], Stephan Krenn[2],
Ahmad-Reza Sadeghi[3], Thomas Schneider[3], and Joe-Kai Tsay[4]

[1] Bern University of Applied Sciences, Biel-Bienne, Switzerland
endre.bangerter@bfh.ch
[2] Bern University of Applied Sciences, Biel-Bienne, and University of Fribourg, Switzerland
stephan.krenn@bfh.ch
[3] Horst Görtz Institute for IT Security, Ruhr-University Bochum, Germany
{ahmad.sadeghi,thomas.schneider}@trust.rub.de
[4] LSV, ENS Cachan, and CNRS, and INRIA, France
tsay@lsv.ens-cachan.fr

**Abstract.** Zero-knowledge proofs of knowledge (ZK-PoK) play an important role in many cryptographic applications. Direct anonymous attestation (DAA) and the identity mixer anonymous authentication system are first real world applications using ZK-PoK as building blocks. But although having been used for many years now, design and implementation of sound ZK-PoK remains challenging. In fact, there are security flaws in various protocols found in literatur. Especially for non-experts in the field it is often hard to design ZK-PoK, since a unified and easy to use theoretical framework on ZK-PoK is missing.

With this paper we overcome important challenges and facilitate the design and implementation of efficient and sound ZK-PoK in practice. First, Camenisch et al. have presented at EUROCRYPT 2009 a first unified and modular theoretical framework for ZK-PoK. This is compelling, but makes use of a rather inefficient 6-move protocol. We extend and improve their framework in terms of efficiency and show how to realize it using efficient 3-move $\Sigma$-protocols. Second, we perform an exact security and efficiency analysis for our new protocol and various protocols found in the literature. The analysis yields novel - and perhaps surprising - results and insights. It reveals for instance that using a 2048 bit RSA modulus, as specified in the DAA standard, only guarantees an upper bound on the success probability of a malicious prover between $1/2^4$ and $1/2^{24}$. Also, based on that analysis we show how to select the most efficient protocol to realize a given proof goal. Finally, we also provide low-level support to a designer by presenting a compiler realizing our framework and optimization techniques, allowing easy implementation of *efficient and sound* protocols.

**Keywords**: Zero knowledge; Proof of knowledge; Protocol design

## 1 Introduction

Zero-knowledge proofs and arguments of knowledge (ZK-PoK) are main building blocks of many higher level cryptographic protocols and applications. Examples include secure watermark detection [1], group signatures [21], interactive verifiable computation [26], efficient secure multiparty computation [53], and identification schemes [65] – just to name a few.

* This work is being performed within the FP7 EU project CACE (Computer Aided Cryptography Engineering). This work was done while Tsay was at the Chair for System Security at Ruhr-University Bochum.

Almost all ZK-PoK protocols being used in practice today are based on so called $\Sigma$-protocols [34], which are 3-move protocols. Basic $\Sigma$-protocols are used to prove knowledge of a preimage under a homomorphism (e.g., a secret discrete logarithm). More complex variations and compositions of these preimage proofs allow to perfom, e.g., AND- and OR-proofs, proofs of linear relations among preimages, etc..

While many applications using ZK-PoK only exist on a specification level, a direction of applied research has produced first such applications that are deployed in the real world. The probably most prominent example is *Direct Anonymous Attestation (DAA)* [19]. Another example is the *identity mixer* anonymous credential system [22], which was released by IBM into the "Higgins" open source project[5].

Yet, while they are important and widely used, design and implementation of sound and efficient ZK-PoK remains challenging. This has certainly to do with the fact that ZK-PoK are significantly more complex than other, non-interactive, crypto primitives. The main problem is the lack of a unified, modular, and easy to understand theoretical framework underlying the various ZK-PoK protocols and proof techniques. As a result there is no methodological formal way to guide cryptographic protocol designers. In fact, there is a large number of tricks and techniques "to prove this and that", yet combining various tricks and preserving the security properties is neither modular nor straightforward. The composition of techniques often requires intricate knowledge, and may also require modifications of the techniques being used. As a result, finding and designing ZK-PoK protocols for a specific purpose is a heuristic process based on experience and a detailed understanding of the techniques being used.

Also experts in the field of ZK-PoK are not immune to protocol design errors. In fact, minor flaws in protocol designs [3,8,41] (which can be fixed easily) can lead to serious security issues [31,37,49]. Moreover, experiences made while implementing, e.g., a prototype of the identity mixer [22,24] protocols have shown that implementing ZK-PoK is tedious and error prone. Minor changes in the protocol design can lead to substantial changes in the implementation. In particular, it is hard for software engineers who are non-experts in the field of ZK-PoK to correctly implement specifications, since there are many intricate choices and dependencies concerning protocol parameters.

### 1.1 Our Contribution

In this paper we describe several theoretical and practical contributions that overcome the challenges mentioned above and facilitate the design and implementation of efficient and sound ZK-PoK in practice. The three following paragraphs summarize our main contributions.

**Efficient and comprehensive framework for ZK-PoK.** At EUROCRYPT 2009 Camenisch et al [23] have presented a compelling unified modular framework (henceforth called *CKY-framework*) for what they call unconditionally portable ZK-PoK for exponentiation homomorphisms. The framework overcomes many of the challenges concerning ZK-PoK protocol design mentioned before.

We extend and improve the CKY-framework towards further practicability and efficiency. To this end we replace the unconditionally portable 6-move $\Sigma_{\tau}^{ext,+}$-protocol from [23] by an also unconditionally portable 3-move $\Sigma$-protocol that we call the $\Sigma^{\mathsf{exp}}$-protocol. The $\Sigma^{\mathsf{exp}}$-protocol will mainly be employed to obtain ZK-PoK for exponentiation homomorphisms in hidden order groups (e.g., RSA groups).

[5] See http://www-03.ibm.com/press/us/en/pressrelease/20974.wss

Using the $\Sigma^{\mathsf{exp}}$-protocol and other existing protocols we describe a framework of unconditionally portable ZK-PoK for homomorphisms that is exclusively based on 3-move $\Sigma$-protocols. Therefore our framework leverages in a straightforward manner existing proof techniques for so called AND-, OR-proofs as well as various techniques to prove linear relations between secret preimages [17,18,30,35]. Also, all protocols in our framework can be turned into non-interactive zero-knowledge (NIZK) or signature proofs of knowledge using the Fiat-Shamir heuristic [40]. To the best of our knowledge it is not known how to obtain practical NIZK proofs for the 6-move protocol used by the CKY-framework. We also describe some other, relatively minor, improvements over the CKY-framework. For instance, our framework additionally supports ZK-PoK for power homomorphisms as used in the RSA system [64] or the mixed exponentiation-power homomorphisms of the Paillier homomorphic encryption scheme [38,59].

Our $\Sigma^{\mathsf{exp}}$-protocol is an argument of knowledge in the auxiliary string model [36]. From a theoretical point of view, PoK in the auxiliary string model are less desirable than standard PoK. The reason is that the dependency on an auxiliary string might result in less modular security proofs, and setting up and distributing the auxiliary string might be inefficient or impractical. We shall see that these problems can be avoided for the $\Sigma^{\mathsf{exp}}$-protocol. In fact, it is often sufficient to have a single, systems-wide certified RSA modulus for the $\Sigma^{\mathsf{exp}}$-protocol to work.

**Concrete security analysis and efficiency optimizations.** The $\Sigma^{\mathsf{exp}}$-protocol and a series of published ZK-PoK protocols [5,8,23,37,41] are all based on the same basic idea originally proposed by Fujisaki and Okamoto [41]. In the following we refer to this class of protocols as *FO-based* protocols. The main application area of these protocols are ZK-PoK for exponentiation homomorphisms in hidden order groups. FO-based protocols play an important role in many practical applications such as DAA and identity mixer. All FO-based protocols are *arguments of knowledge under the Strong RSA assumption*, i.e., their soundness is based on the Strong RSA assumption. In this paper, we present the first exact security analysis of the soundness for the class of FO-based protocols. The analysis shows that even the best currently known reduction is not tight, which leads to - perhaps surprising - results and insights. The analysis reveals for instance that, using a 2048 bit RSA modulus, as specified in the DAA standard, yields an upper bound on the cheating probability of a malicious prover of *only* $1/2^4$ - $1/2^{24}$, depending on the computational power attributed to a malicious prover. These security guarantees are considered to be insecure by many. To reduce the cheating probability to $1/2^{80}$, the RSA modulus used in FO-based protocols needs to be approximately $10'000$ - $15'000$ bits long (again depending on the computational power of the malicious prover). This raises the question in how far FO-based protocols are computationally efficient.

We show that the answer to this question is multi-faceted. The folklore saying that FO-based protocols are the most efficient technique to carry out ZK-PoK in hidden order groups is not generally true. In fact, our analysis shows that sequentially repeating[6] a variant of the Schnorr protocol using binary challenges (i.e., a challenge set $\mathcal{C} = \{0, 1\}$) is sometimes more efficient than using a FO-based protocol.

**Automated implementation of sound protocols.** We have designed and implemented a language for high-level specification of proof goals together with a compiler that outputs the protocol implementation in the $C$ language together with a human-readable documentation in

[6] We recall that sequentially repeating a ZK-PoK with knowledge error $\kappa$, $r$-times reduces the knowledge error to $\kappa^r$.

LaTeX. The language and compiler realize our protocol framework and optimization strategies based on the efficiency considerations mentioned above. They also support protocol designers in getting sound protocols in various ways. For instance, given an overall security parameter for the protocols to be generated, the compiler will appropriately choose protocol parameters (e.g., interval sizes, number of parallel executions of protocols, etc.) to assert that the knowledge error and the (statistical) zero-knowledge property meet the specified security level. Also, we have extensively tested the correctness of the compiler using conventional software testing techniques. We plan to tackle the formal verification in future work. Due to lack of space, we give only an overview of the compiler in this paper, the full version of this paper will contain a detailed coverage.

### 1.2 Related Work

ZK-PoK were first introduced in the seminal work of Goldwasser et al. in [43]. The definitional setting of proofs of knowledge was later refined [10], resulting in what is regarded the standard definition today. The first efficient $\Sigma$-protocols for proving knowledge of discrete logarithms in known order groups and roots in RSA groups were given by Schnorr [65] and Guillou and Quisquater [45]. There are various works describing so called "AND-", "OR-"compositions, for proving knowledge of linear relations between secret pre-images etc., and also frameworks for combining these techniques [17,18,30,35]. All these frameworks consider homomorphisms with finite domain. A profound analysis of the $\Sigma$-protocols was performed by Cramer [34]. The first efficient ZK-PoK protocol for exponentiation homomorphisms in hidden order groups was given in [41] and has been corrected later by Damgård and Fujisaki [37]. Subsequently, other variants overcoming some limitations have been proposed [5,7,8].

In very recent work at EUROCRYPT 2009, a long overdue unified framework for exponentiation homomorphisms in hidden and known order groups was given by Camenisch et al. [23]. As discussed above in more detail, we present a framework extending that in [23]. Our framework also incorporates many of the proof techniques and protocols mentioned above. We believe that our framework is the most comprehensive and efficient framework for homomorphisms in known and hidden order groups.

Our concrete security analysis of FO-based protocols, makes use of the analysis of the knowledge extractor described by Damgård and Fujisaki [37]. The results of our analysis concern the entire class of what we call FO-based protocols [5,8,23,37,41]. Otherwise our analysis and related insights are novel.

A first prototype of a zero-knowledge compiler was started in [20,27], and was later extended in [7]. This compiler only supports a very basic framework of proofs in known order groups and it lacks the underlying unified framework and efficiency optimizations used by our compiler. Similar work to ours was performed in the field of secure function evaluation [56,57]. Their compilers allow to specify the function to be evaluated in a high-level language, and output executable code. In principle, zero-knowledge proofs could be realized by secure function evaluation [46]. Yet, the resulting protocols are significantly less efficient than those generated by our compiler.

## 2 An Efficient Three-Move Framework

In very recent work, Camenisch et al. gave a first unified framework for exponentiation homomorphisms in groups of known *and* hidden order [23]. We refer to this framework as the *CKY-framework*. However, for many real-world applications, their results are not adequate

because of the large computational and communicational overhead. In the following we propose a framework improving over the CKY-framework, whilst keeping its universality. We first give an efficient protocol realizing its theory, and then extend its applicability by combining our protocol with other well known techniques [34,35].

In §2.1, we introduce the novel $\Sigma^{\mathsf{exp}}$-protocol. The ZK property of this protocol holds in the plain model, whereas it is proven to be sound in the auxiliary string model. For a formal treatment of arguments and proofs of knowledge in this model we refer to [36]. Using the notion of [23], it is an unconditionally portable protocol. This means, that it can be used for preimage proofs under arbitrary exponentiation homomorphisms, independent of the distribution of the inputs to the prover and the verifier. Besides reducing the computational costs compared to the protocols proposed in [8,23], the $\Sigma^{\mathsf{exp}}$-protocol also reduces the number of moves from 6 to 3. We want to stress that the protocol was invented independently from, and earlier than, the protocol in [23] already in [5].

After giving a short discussion of the auxiliary string model and a comparison to the protocols of [8,23] in §2.2, we extend the scope of our framework also to power-homomorphisms in §2.3. Those are used in various schemes such as RSA or the Damgård-Jurik encryption scheme [38,59,64]. This is achieved by also including the well known $\Sigma^{\phi}$-protocol [34,45,65] into our framework. Additionally, doing so allows to further increase the efficiency for certain proof goals. Finally, we give an easy-to-use recipe guiding a cryptographic designer in finding an efficient protocol for a given proof goal.

### 2.1 $\Sigma^{\mathsf{exp}}$ - An Efficient, Unconditionally Portable Protocol

Before we describe the $\Sigma^{\mathsf{exp}}$-protocol we state the underlying intractability assumption, namely the Strong RSA assumption. This also underlies previous protocols for proofs in hidden order groups [8,23,37,41].

**Definition 1 (Strong RSA assumption [41]).** *The* Strong RSA problem *is defined as follows: given an RSA modulus $n$ and $y \in \mathbb{Z}_n$, compute $m \in \mathbb{Z}_n$ and $e \geq 2$, such that $y = m^e \bmod n$. The* Strong RSA assumption *then states, that there is no probabilistic polynomial-time algorithm, that on input $(n, y \in_R \mathbb{Z}_n)$ succeeds in solving the Strong RSA problem with non-negligible probability.*

Next, we describe how to generate the auxiliary string $\vartheta$ that we use in the description of the $\Sigma^{\mathsf{exp}}$-protocol. Given the security parameter $k$ and the number $m$ of base elements, one chooses a safe RSA modulus $n$ (i.e. $n = pq$, where $p, q, \frac{p-1}{2}, \frac{q-1}{2}$ are large primes) of length $k$, and a random element $g \in_R \mathbb{Z}_n^*$ generating the quadratic residues modulo $n$, i.e. $\langle g \rangle = QR_n$. Further, one chooses $g_i \in_R \langle g \rangle$ for $i = 1, \ldots, m$. Then the auxiliary string is given by the homomorphism $\vartheta(x_1, \ldots, x_m, \mathsf{x}) := g_1^{x_1} \ldots g_m^{x_m} g^{\mathsf{x}} \bmod n$.

In the following we set $\vartheta_u(x_1, \ldots, x_u, \mathsf{x}) := \vartheta(x_1, \ldots, x_u, 0, \ldots, 0, \mathsf{x})$ for any $u \leq m$.

We now explain knowledge of the preimage of a public value $x := \phi_E(w_1, \ldots, w_u)$ for some exponentiation homomorphism $\phi_E$ with $u \leq m$ can be proven. For each $1 \leq i \leq u$, let $L_i, R_i$ be given such that $w_i \in [L_i, R_i]$. (Throughout the whole paper, we always consider *integer intervals*.) These intervals can be chosen arbitrarily, and are only needed to control the provers random choices to make the protocol statistical honest-verifier zero-knowledge (HVZK) whenever $w_i \in [L_i, R_i]$ for all $i$. Further, we set $m_i := R_i - L_i$ to the length of the interval $w_i$ lies in.

The tightness of the statistical ZK property is controlled by the additional security parameter $l$.

**Definition 2 ($\Sigma^{\mathsf{exp}}$-protocol).** *Let $\vartheta$ be generated as above, and let $\phi_E$ be an exponentiation homomorphism with domain $\mathbb{Z}^u$ ($u \leq m$), and $x := \phi_E(w_1, \ldots, w_u)$ with $w_i \in [L_i, R_i]$. Then the* $\Sigma^{\mathsf{exp}}$-protocol *with challenge set $\mathcal{C} := [0, c^+]$ and auxiliary string $\vartheta$ consists of two parties, $\mathsf{P}$ and $\mathsf{V}$, performing the joint computation described in Fig. 1.*

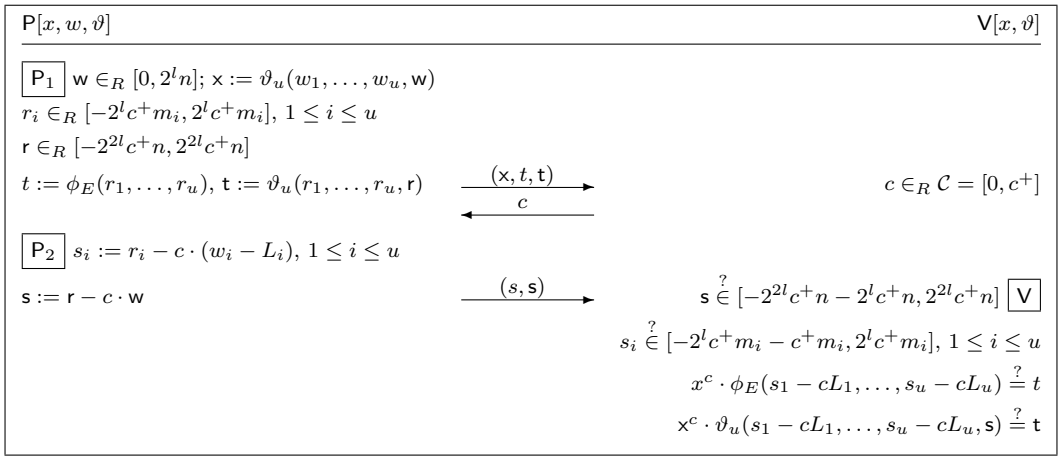


**Fig. 1.** The $\Sigma^{exp}$-protocol

By a slight modification the restriction $u \leq m$ can be overcome. The idea is to split the set of preimages into subsets of at most $m$ elements each, and committing to each of these subsets seperately by a pair $(\mathsf{x}, \mathsf{t})$. We will examine this idea in detail in the full version of this paper.

The following theorem states sufficient conditions for the $\Sigma^{\mathsf{exp}}$-protocol to be an argument of knowledge.

**Theorem 3.** *Let $c^+$ be smaller than any divisor of the order of the image of $\phi_E$. Under the Strong RSA assumption, the $\Sigma^{\mathsf{exp}}$-protocol as defined above then is a statistical HVZK, unconditionally portable argument of knowledge in the auxiliary string model.*

We note that this theorem does not make any statement about the knowledge error of the resulting protocol. This will be discussed in detail in §3. Due to space limitations we omit a proof here. It works analogously to that of the six-move protocols proposed in [8,23] and will be given in the full version of this paper.

We want to make one more remark on the $\Sigma^{\mathsf{exp}}$-protocol. By repeating an argument (or proof) of knowledge with knowledge error $\kappa$, say, $r$ times, the success probability of a malicious prover can be reduced to $\kappa^r$. Note that for the $\Sigma^{\mathsf{exp}}$-protocol this can only be guaranteed by using a *different* auxiliary string in each repetition. If otherwise a malicious prover manages to factor $n$ in the first execution of the protocol, the Strong RSA assumption does not hold any more in the subsequent repetitions. In similar protocols [5,8,23] this problem is solved inherently by the protocol: there, $\mathsf{V}$ chooses a new random modulus and base elements in its first step in each repetition.

**Minimizing the auxiliary string.** In the following we show how to reduce the auxiliary string to the modulus only: the idea is that the prover and the verifier both compute the bases themselves. Yet, this must happen in a way that guarantees $g_i \in \langle g \rangle$ (for the protocol to be ZK), without the prover learning either the discrete logarithms of $g_i$ in base $g$ for any $i$, or any non-trivial roots of a $g_i$ or $g$ (for the protocol to be sound).

This can be achieved by requiring the modulus to be of the form $n = p^2q$ for two safe primes $p, q$. As such moduli are hard to factor, the Strong RSA problem is still believed to be hard [62,63]. Yet, we want to mention that the length of such moduli has to be higher than that of safe RSA moduli for the Strong Root problem to have the same complexity, which results in less efficient protocols.

Because of the multiplicativity of the Jacobi symbol we have $\left(\frac{x}{n}\right) = \left(\frac{x}{p}\right)^2\left(\frac{x}{q}\right) = \left(\frac{x}{q}\right)$ for all $x \in \mathbb{Z}_n^*$. Hence, $x$ is a quadratic residue modulo $q$, if and only if $\left(\frac{x}{n}\right) = 1$. This check can be performed efficiently without knowing the factorization of $n$ [39].

Let additionally $\mathcal{H}$ be a hash-function with domain $\mathbb{Z}_n$, i.e., $\mathcal{H} : \{0,1\}^* \to \mathbb{Z}_n$. Note that such *full domain hashes* can be obtained efficiently from existing hash-functions [11]. The bases of the auxiliary string are then set to the first $m+1$ randomly chosen quadratic residues modulo $q$ in $\mathbb{Z}_n^*$:

$$g := \mathcal{H}(j_0), \quad g_i := \mathcal{H}(j_i) \quad \text{with} \quad j_i := \min\{k > j_{i-1} : \left(\frac{\mathcal{H}(k)}{n}\right) = 1\} \text{ and } j_{-1} := 0$$

As the probability of a random $g$ to be a quadratic residue modulo $q$ is about $1/2$, on average $1.5(m+1)$ evaluations of $\mathcal{H}$ are necessary to obtain the elements. Because of $q$ being a safe prime, for any $x \in QR_q$ either $\langle x \rangle = QR_q$ or $x \equiv 1 \mod q$ holds. The probability for the latter is $1/q$ and therefore negligible; thus the statistical ZK-property can be followed from that of the $\Sigma^{\mathsf{exp}}$-protocol. Soundness of the protocol can be seen by using the same reduction proof as for the $\Sigma^{\mathsf{exp}}$-protocol, e.g., [25,37].

### 2.2 Discussion of the Auxiliary String Model and Related Protocols

The $\Sigma^{\mathsf{exp}}$-protocol described before is an argument of knowledge in the *auxiliary string model* (often also referred to as *common reference string model*). The model was first introduced in [14], and a formal definition of arguments and proofs of knowledge in that model was given by Damgård [36].

The idea of the auxiliary string model is that all parties in a protocol have access to a common *auxiliary* string which was chosen following some predefined distribution. Besides distributing the auxiliary string, the main disadvantage of this approach is that often no information on the auxiliary string except the string itself must be leaked to one or more of the participating parties. For instance, in the $\Sigma^{\mathsf{exp}}$-protocol the prover must neither learn anything about the factorization of the modulus, nor discrete logarithms or roots of the base elements. Yet, in our case this is mainly a theoretical problem. In practice, this can be guaranteed if a trusted third party (TTP) generates the auxiliary string once together with a certificate vouching its correctness. Alternatively, it can be chosen by the verifier and proven to be correct in zero-knowledge at the beginning of the system's lifetime. Additionally, in most applications, the advantages of this model outweigh the disadvantages. Especially, computationally or communicationally expensive setup phases can be avoided or moved forward to the beginning of the system's lifetime.

We believe that the $\Sigma^{\mathsf{exp}}$-protocol is the best compromise between the protocols in [8,23] on the one hand and that in [37] on the other. By moving to the auxiliary string model,

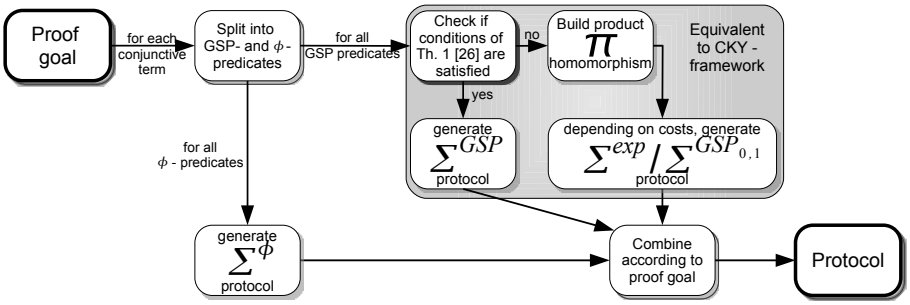


**Fig. 2.** How to use our Framework; the decision whether to generate a GSP with binary challenge set, denoted $\Sigma^{\mathsf{GSP}_{0,1}}$, or generating $\Sigma^{\mathsf{exp}}$, will be made according to the efficiency analysis of §3.2.

we keep the universality of the former. That is, the $\Sigma^{\mathsf{exp}}$-protocol can be used for arbitrary preimage proofs of exponentiation homomorphisms, not depending on the distribution of the input as is the case in [37]. Yet, as in [37] we shift the setup phase to the beginning of the system's lifetime (or to a TTP), and therefore save three moves and several computations at runtime compared to [8,23].

### 2.3 How to Use Our Framework

In the following we explain how a cryptographic protocol designer can obtain an efficient protocol for a given proof goal containing special homomorphisms (see App. A), and exponentiation homomorphisms in groups of known and unknown order, potentially along with optional interval claims for the preimages. Consider for a moment a proof goal of the form

$$\mathsf{ZPK}\left[(\omega_1,\ldots,\omega_u) : \bigwedge_{i=1}^{r} x_i = \phi_i(\omega_{i_1},\ldots,\omega_{i_{u_i}}) \quad \wedge \quad \bigwedge_{j=1}^{u} \omega_j \in [A_i, B_i]\right].$$

By a *predicate* we then understand a term $x_i = \phi_i(w_{i_1},\ldots,w_{i_{u_i}})$ together with all interval claims associated to the preimages occurring in $\phi_i$. A predicate is called *special*, if $\phi_i$ is a special homomorphism and there are no interval claims. Predicates involving exponentiation homomorphisms in groups of unknown order, or such with non-trivial interval claims are called *GSP-predicates*.

The key ingredients of our framework are the $\Sigma^{\mathsf{exp}}$-protocol and the well known $\Sigma^{\phi}$-and $\Sigma^{\mathsf{GSP}}$-protocols. The latter two protocols are briefly recapitulated in App. A. Further we make use of efficient techniques to compose $\Sigma$-protocols [35]. Using the notion introduced above, Fig. 2 gives an easy-to-grasp recipe how to obtain an efficient protocol for a given proof goal.

The class of proof goals that can be proven by restricting Fig. 2 to the part within the gray shape only, is equivalent to that considered by the framework proposed in [23]. For this class, using the $\Sigma^{\mathsf{exp}}$-protocol improves over the protocols proposed in [8,23] in terms of efficiency. Yet, by also considering the $\Sigma^{\phi}$-protocol, we extend this class to power-homomorphisms as well.

All protocols generated in this way are at least *statistical honest verifier zero knowledge arguments of knowledge in the auxiliary string model*. Yet, there are cases where perfect HVZK respectively the proof of knowledge property in the plain model can be inferred.

## 3 Exact Security Analysis of FO-based Protocols

As stated in §2, the $\Sigma^{\mathsf{exp}}$-protocol and the protocols proposed in [5,8,23,37] only yield *arguments of knowledge* (often also referred to as *computational proofs of knowledge*) [10], with the underlying intractability assumption being the Strong RSA assumption. All those protocols build on the same idea as the protocol proposed by Fujisaki and Okamoto [41], and we call them *FO-based* henceforth. As any malicious prover can solve the Strong RSA problem with probability strictly larger than zero (e.g. by trying to factor the modulus), the security of FO-based protocols does not only depend on the size of the challenge set $|\mathcal{C}|$ but also on the length of the modulus $|n|$. That is, both these parameters have to be increased simultaneously to strengthen the security of the protocol.

Although FO-based protocols have been widely used for years [1,13,19,22,44], an exact security analysis for the necessary length of the modulus is still outstanding. We perform this analysis in §3.1, obtaining surprising results. To limit the prover's cheating probability to $2^{-80}$, the modulus has to have between $10'000$ and $15'000$ bits for provers computationally limited to $2^{40}$ respectively $2^{80}$ steps. Alternatively, if moduli of a fixed length have to be used, a large number of repetitions might be necessary (remember that $r$ repetitions of a PoK decrease the knowledge error to $\kappa^r$). For instance, the DAA protocol using 2048-bit moduli as specified in [19] only guarantees a cheating probability between $2^{-24}$ and $2^{-4}$, again depending on the prover's power. This means, that 4 to 23 repetitions are necessary to achieve a cheating probability of at most $2^{-80}$.

Using very long moduli as well as repeating the protocol often causes high computational costs. Yet, decreasing the length of the modulus in a FO-based protocols increases the number of necessary repetitions, and vice versa. Hence some questions arise: What is the optimum in this efficiency tradeoff? Are FO-based protocols practical at all? Or is the $\Sigma^{\mathsf{GSP}_{0,1}}$-protocol (i.e. the $\Sigma^{\mathsf{GSP}}$-protocol with binary challenge set $\mathcal{C} = \{0,1\}$) more efficient, although often being decried to be inefficient [5,7,8]? In §3.2 we discuss those questions for the $\Sigma^{\mathsf{exp}}$-protocol in detail. We show that they cannot be answered in general, but that the answers very much depend on, e.g., the concrete proof goal and the algebraic setting.

### 3.1 Determining the Size of the Modulus

For arguments of knowledge we follow the definition given in [37], and split the cheating probability of the malicious prover into two parts: $\kappa$ and $\nu$. Informally, the former indicates the "standard" knowledge error, whereas the latter denotes the probability that the knowledge-extractor fails for a prover convincing the verifier with probability larger than $\kappa$. For further discussion, we refer to [37].

**Definition 4 (Argument of knowledge [37]).** *A two-party protocol between a prover and a verifier is a* $(\nu, \tau, \kappa)$-convincing argument of knowledge*, if the verifier accepts for an honest prover, and if there exists a polynomial* $p(\cdot)$ *and an extractor* $M$*, such that the following holds: for every prover that is limited to at most* $\tau$ *steps, and convinces the verifier with probability* $\varepsilon$*,* $M$ *computes a witness in at most* $\frac{p(\cdot)}{\varepsilon - \kappa}$ *steps with probability at least* $(1 - \nu)$*. We refer to* $\kappa + \nu$ *as the* computational knowledge error*.*

We note that the computational knowledge error is an upper bound on the success probability of a malicious prover. One of the differences to the widely used definition given in [10] is that latter requires the knowledge extractor to run within a strict time bound instead of an

expected one. This is worse suited in our context, because the Strong RSA assumption does not consider algorithms with a bound on their expected runtime either.

Let $k$ denote the length of the modulus in a FO-based protocol, and let $L(k)$ denote the complexity of solving the Strong RSA problem for such a modulus. Further, let $s(k)$ the necessary key length to achieve the same security in a symmetric encryption scheme. The following theorem then describes the relation between the computational knowledge error and $l$. (For the proof see App. B.)

**Theorem 5.** *Let $2^a$ be the number of steps a malicious prover can perform. Then a FO-based protocol using $k$-bit safe RSA moduli is an argument of knowledge with computational knowledge error $36 \cdot 2^v$, where $v = \frac{1}{2}(\log_2 448 - \log_2 18 + a - s(k))$, if $\lceil v + \log_2 \frac{9}{2} \rceil$-bit challenges are used.*

Ensuring a FO-based protocol to be an argument of knowledge with computational knowledge error less than $2^{-b}$ can now be achieved via two approaches: in the first, the length of the modulus is fixed, e.g., by the scope of the application, and the computational knowledge error is decreased by sequentially repeating the protocol. For example, this is the case for DAA and other protocols using Trusted Platform Modules (TPMs), which, according to their specification, have to support moduli with a length of up to 2048-bit only. In the second, the modulus is chosen large enough to ensure the required computational knowledge error in only one repetition of the protocol. This minimizes the length of the proof, which is important for signature proofs of knowledge.

Those two approaches are captured by the following corollary, the proof of which is in App. B:

**Corollary 6.** *Using $v$ from Th. 5, a FO-based protocol using $k$-bit safe RSA moduli has to be repeated $r = \lceil \frac{-b}{v + \log_2 36} \rceil$ times using at least $\lceil \frac{b}{r} + 3 \rceil$-bit challenges to achieve a computational knowledge error of at most $2^{-b}$ against a prover limited to $2^a$ steps. Especially, if $s(k) \geq 15 + a + 2b$, one repetition of the protocol using $b + 3$-bit challenges is sufficient.*

**Applying results to practice.** At the time of writing, all known algorithms for solving the Strong RSA problem use the factorization of the modulus. The fastest algorithm to factor an $k$-bit modulus currently is the general number field sieve (GNFS) [51], for which $L(k)$ is then given by:

$$L(k) = \exp\big((\alpha + o(1)) \cdot (k \ln 2)^{1/3} \cdot (\ln(k \ln 2))^{2/3}\big).$$

The factor $\alpha$ depends on the form of the modulus to factor. For arbitrary moduli, the best results can be obtained using the modification proposed by Coppersmith [33], having $\alpha \approx 1.90$, while for moduli close to a power of a small integer this can be improved to $\alpha \approx 1.53$. Hence such moduli should be avoided in practice.

We use that $L(k)/L(k_0) = 2^{s(k) - s(k_0)}$. This is easy to see, as doubling the complexity of solving the Strong RSA problem is equivalent to increasing the keylength in a symmetric scheme by 1. Thus, knowing $s(k_0)$ for some $k_0$ allows to estimate $s(k)$ for arbitrary $k$. Today's recommendations are to use an 1248-bit RSA modulus to achieve a security level equivalent to an 80-bit symmetric key [9,42,50,52,58]. Hence, we have $k_0 = 1248$ and $s(k_0) = 80$. Further we assume the $o(1)$-term to be negligible for $k \geq 1248$ and set it to zero. Note that this approach to estimate the security of a modulus is almost identical to that in [58].

Table 1 shows that very large moduli are needed to perform the proof within only one protocol run, or that the protocol has to be repeated sequentially, if the modulus is restricted

| Computational power of P | Computational Knowledge Error | Length of modulus | Repetitions needed for ... $\|n\| = 2048$ | $\|n\| = 4096$ |
|---|---|---|---|---|
| $2^{40}$ | $2^{-80}$ | 10479 | 4 | 2 |
| $2^{60}$ | $2^{-80}$ | 12852 | 6 | 3 |
| $2^{80}$ | $2^{-80}$ | 15528 | 23 | 4 |
| $2^{40}$ | $2^{-100}$ | 15528 | 5 | 3 |
| $2^{60}$ | $2^{-100}$ | 18522 | 8 | 4 |
| $2^{80}$ | $2^{-100}$ | 21847 | 29 | 5 |

**Table 1.** Either very large moduli or many repetitions are necessary to achieve reasonable security.

to 2048 or 4096 bits. We stress that a computational bound of $2^{80}$ steps yields security against any malicious prover, whilst $2^{40}$ steps is only reasonable against computationally weak provers.

Let us conclude this section with a short discussion of the previous results. On the one hand, the length of the moduli will increase if a more efficient algorithm for factoring is found, or if the Strong RSA problem turns out to be less complex than factoring. Yet, recent results have given new evidence that these problems might be equally hard to solve [2]. On the other hand, finding a tighter reduction proof than that in [37] could decrease the length of the moduli significantly. There are at least two points where such an improvement could take place. First, a more efficient rewinding algorithm (having black-box access to the cheating prover) to extract two accepting communication triples having the same first message and different challenges could be found for FO-based protocols. Second, the algebraic reduction from a cheating prover to an algorithm solving the Strong RSA problem could be tightened.

### 3.2 Minimizing Computational Costs

In the following we show how to minimize the computational costs of a proof by choosing the optimal length of the auxiliary modulus in the $\Sigma^{\mathsf{exp}}$-protocol, and by picking the $\Sigma^{\mathsf{GSP}_{0,1}}$-protocol in certain cases. This optimization is essential when running the protocol on low-cost devices such as smart-cards. In many applications the prover authenticates itself to a computationally powerful server acting as verifier. Hence, we perform our analysis for P. If the computational costs of V are of main interest, the analysis can easily be adopted, yielding similar results.

We first estimate the computational costs of the $\Sigma^{\mathsf{exp}}$- and the $\Sigma^{\mathsf{GSP}_{0,1}}$-protocols. These costs are mainly determined by the number of modular multiplications, and the length of the moduli therein. Thus, we neglect the costs of other operations such as additions over the integers.

To ease the presentation we restrict the form of the co-domain of the homomorphism for which knowledge of preimages has to be proven. That is, we assume the homomorphism is given by:

$$\phi : \mathbb{Z}^u \rightarrow (\mathbb{Z}^*_{n_0})^d : (x_1, \ldots, x_u) \mapsto (\prod_{i \in S_1} g_{1i}^{x_i}, \ldots, \prod_{i \in S_d} g_{di}^{x_i}) \qquad \text{with} \qquad S_1, \ldots, S_d \subseteq \{1, \ldots, u\}.$$

Note that this restriction on the co-domain is satisfied for many protocols proposed in literature.

**Multiplication proof**

| $\lvert n_0\rvert$ | $\mathsf{Costs}(\Sigma^{\mathsf{exp}},\lvert n\rvert)/\mathsf{Costs}(\Sigma^{\mathsf{GSP}_{0,1}})$ | | |
|---|---|---|---|
| | $\lvert n\rvert = 15528$ | $\lvert n\rvert = 2048$ | optimal $\lvert n\rvert$ |
| 1024 | 42.68 | 2.73 | 1.92 ($\lvert n\rvert = 2610$) |
| 1280 | 24.00 | 1.66 | 1.12 ($\lvert n\rvert = 2610$) |
| 1536 | 13.08 | 1.03 | 0.65 ($\lvert n\rvert = 2748$) |
| 2048 | 5.63 | 0.60 | 0.33 ($\lvert n\rvert = 2939$) |

**Lipmaa's interval proof**

| $\lvert n_0\rvert$ | $\mathsf{Costs}(\Sigma^{\mathsf{exp}},\lvert n\rvert)/\mathsf{Costs}(\Sigma^{\mathsf{GSP}_{0,1}})$ | | |
|---|---|---|---|
| | $\lvert n\rvert = 15528$ | $\lvert n\rvert = 2048$ | optimal $\lvert n\rvert$ |
| 1024 | 14.77 | 1.20 | 0.77 ($\lvert n\rvert = 2748$) |
| 1280 | 8.33 | 0.80 | 0.47 ($\lvert n\rvert = 2748$) |
| 1536 | 4.56 | 0.56 | 0.30 ($\lvert n\rvert = 2939$) |
| 2048 | 1.97 | 0.40 | 0.17 ($\lvert n\rvert = 3221$) |

**Table 2.** Relative computational costs of $\Sigma^{\mathsf{exp}}$ and $\Sigma^{\mathsf{GSP}_{0,1}}$, when aiming to reach a computational knowledge error of $2^{-80}$ for a prover limited to $2^{80}$ steps.

**Costs of the $\Sigma^{\mathsf{GSP}_{0,1}}$-protocol.** We start by estimating the computational costs when performing a proof using the $\Sigma^{\mathsf{GSP}_{0,1}}$-protocol. For efficient algorithms for multi-exponentiation, i.e. evaluation of a term like $g_1^{x_1} \cdot \ldots \cdot g_t^{x_t}$, the number of multiplications is mainly determined by the length of the largest exponent [4,15]. More precisely, it can be estimated by $(1+\frac{t}{\log_2 \ell})\ell$, where $\ell = \max_i |x_i|$. Using the notation from Def. 2, the costs of running the $\Sigma^{\mathsf{GSP}_{0,1}}$-protocol $b$ times, can now be estimated by:

$$\mathsf{Costs}(\Sigma^{\mathsf{GSP}_{0,1}}) := b \cdot \sum_{i=1}^{d}(1+\frac{|S_i|}{\log_2 \ell_i})\ell_i \qquad \text{for} \qquad \ell_i := \max_{j\in S_j}|m_j| + l + 1.$$

**Costs of the $\Sigma^{\mathsf{exp}}$-protocol.** Similarly, for the $\Sigma^{\mathsf{exp}}$-protocol using the auxiliary modulus $n$, the number of multiplications in $\mathbb{Z}_{n_0}^*$ respectively $\mathbb{Z}_n^*$ can be estimated. Using Cor. 6 to obtain the number $r$ of repetitions, and the challenge length $l_c$, we get the costs for the $\Sigma^{\mathsf{exp}}$-protocol:

$$\mathsf{Costs}(\Sigma^{\mathsf{exp}},|n|) := r \cdot \left(\sum_{i=1}^{d}(1+\frac{|S_i|}{\log_2 \ell_i})\ell_i + f\Big((1+\frac{u+1}{\log_2 \ell'})\ell' + (1+\frac{u+1}{\log_2 \ell''})\ell''\Big)\right).$$

Here we have $\ell_i := \max_{j\in S_j}|m_j| + l + l_c$, $\ell' := |n| + l$, respectively $\ell'' := |n| + 2l + l_c$ for the lengths of the exponents occurring in the protocol. Further, $f$ describes the relative costs of modular multiplications in $\mathbb{Z}_n^*$ compared to $\mathbb{Z}_{n_0}^*$. We assume that these costs grow quadratically in the length of the modulus, and therefore have $f = (\frac{|n|}{|n_0|})^2$. This can be reached by using efficient implementations of multiplication and modular reduction, for which the costs grow (almost) linear in the length of the modulus, e.g. [61,66] respectively [48].

**Minimizing computational costs.** If the length of the modulus is not restricted by the scope of the application, the costs of the $\Sigma^{\mathsf{exp}}$-protocol can be seen as a function in $|n|$. This can be minimized numerically, because with the notation from §3.1 only $|n|$ with $15 + a \leq s(|n|) \leq 15+a+2b$ have to be considered. Smaller $|n|$ yield a computational knowledge error of 1 (cf. Th. 5), while larger $|n|$ guarantee a smaller computational knowledge error than required and therefore cause unnecessary effort. These costs can then be compared to the costs of the $\Sigma^{\mathsf{GSP}_{0,1}}$-protocol, and the protocol yielding the minimal costs is chosen.

Table 2 illustrates these relative costs for a simple multiplication proof and Lipmaa's interval proof [54] for different values of $|n_0|$. Both these examples are integral building blocks of

various applications [1,19,22,44]. The table gives the relative costs of the $\Sigma^{\mathsf{exp}}$-protocol compared to the $\Sigma^{\mathsf{GSP}_{0,1}}$-protocol for a prover limited to $2^{80}$ steps, and a required computational knowledge error of $2^{-80}$, for three different situations. In the first, a $15'528$-bit modulus is used to perform the proof in one run. In the second, the modulus has 2048 bits, as is the case when using TPMs, e.g. [19]. In the third, there is no restriction on the choice of $|n|$. Values larger than 1 give priority to the $\Sigma^{\mathsf{GSP}_{0,1}}$-protocol. It can be seen that the relative costs of the $\Sigma^{\mathsf{exp}}$-protocol decrease with increasing complexity of the proof goal and the value of $|n_0|$. On the contrary, restricting the length of the modulus speaks against the usage of $\Sigma^{\mathsf{exp}}$. Especially, we see that the relative efficiency varies considerably for different proof goals.

**Parameters determining relative costs.** In the following we point out parameters the relative efficiency of $\Sigma^{\mathsf{GSP}_{0,1}}$ and $\Sigma^{\mathsf{exp}}$ depends on. These do not allow to make a general statement about which protocol is more efficient, but show that this *very much depends on the concrete proof*. Yet, we can identify the relevant parameters and qualitatively describe their influence on the relative costs:

- **Proof goal**: An increase in the number of images for which knowledge of the preimages has to be proven increases the costs of the $\Sigma^{\mathsf{GSP}_{0,1}}$-protocol faster than those of the $\Sigma^{\mathsf{exp}}$-protocol. Especially, if the length of the modulus does not underly any restrictions, the $\Sigma^{\mathsf{exp}}$-protocol is more efficient for all values of $|n_0| \geq 1024$, if knowledge for at least 5 images is proven.
- **Algebraic setting**: Larger values of $|n_0|$, i.e. an increasing size of the co-domain of the homomorphism $\phi$, decrease the relative costs of computations in $\mathbb{Z}_n^*$, and thus make the $\Sigma^{\mathsf{exp}}$-protocol be applied more often.
- **Underlying mathematical libraries**: If the costs for multiplication or modular reduction grow superlinear in the length of the modulus, as is the case for Karatsuba's multiplication algorithm [47], the relative costs of operations in $\mathbb{Z}_n^*$ grow. Further, using parallelized algorithms for multi-exponentiation [12,60] can decrease the complexity of $\Sigma^{\mathsf{GSP}_{0,1}}$ significantly. Especially in the latter case the $\Sigma^{\mathsf{exp}}$-protocol will only be given priority for very complex proof goals.
- **Restriction of the proof length**: Especially when used for signature proofs of knowledge, the length of the proof is essential. This is minimal for the $\Sigma^{\mathsf{exp}}$-protocol using one large modulus, at the cost of high computational costs. Yet, the $\Sigma^{\mathsf{GSP}_{0,1}}$-protocol is not an alternative in this case.

## 4 Compiler Framework

We present a toolbox, consisting of tools and languages, for automatic generation of sound ZK-PoK protocols which realizes the efficient three-move framework presented in §2 including the optimizations described in §3. How such a toolbox can overcome several challenges in the design and implementation of ZK-PoK protocols is discussed in [6]. We give an overview of the architecture in §4.1 and details on the input language in §4.2.

### 4.1 Architecture of Toolbox

Our toolbox, as shown in Fig. 3, is modularly composed from multiple components to allow for easy extension (e.g., via new plugins or backends), exchange (e.g., cryptographic libraries used in C backend), and re-use of components.

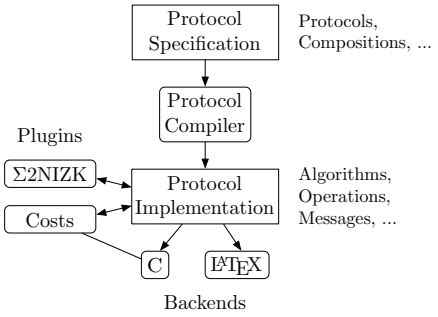


**Fig. 3.** Architecture

```
Declarations {
  RSA(1248) n';
  Int(1328) r, r';
  Unsigned T:=0xC0FFEE; [-T;T] m;
  H=Zmod*(n') g, h, x, x';
}
Inputs {
  ProverPrivate := m, r, r';
  Public        := n', g, h, x, x';
}
Protocol {
  KnowledgeError      := 80;
  SZKParameter        := 80;
  Composition := P_1;
}
SigmaPort(opt_computation) P_1 {
  Homomorphism (phi : Z^3 -> H^2 :
                (a,b,c) |-> (g^a * h^b, y^a * h^c));
  Relation ((x,x') = phi(m, r, r' - m*r));
}
```

**Fig. 4.** PSL example for $\mathsf{ZPK}[(m, r, r') : x = g^m \cdot h^r \mod n' \quad \wedge \quad x' = g^{m^2} \cdot h^{r'} \mod n']$.

**Protocol Specification.** The user formulates the specification of the intended protocol in our *Protocol Specification Language* (PSL). On a high level, this language concretizes the Camenisch-Stadler notation [29] and allows to specify which kind of $\Sigma$-protocols with which input parameters in which algebraic structures should be composed in which way (see example in Fig. 4). In particular, PSL supports the $\Sigma$-protocols described in §2 and App. A: $\Sigma^{\phi}$, $\Sigma^{\mathsf{GSP}}$, and $\Sigma^{\mathsf{exp}}$ with arbitrary compositions via AND, OR, and $n$-out-of-$k$-threshold composition [35] using Shamir's secret sharing [67]. Details on PSL together with a detailed description of the example PSL file shown in Fig. 4 are given in §4.2.

**Protocol Compiler.** The Protocol Compiler takes the protocol specification and automatically chooses the size of the $\Sigma^{\mathsf{exp}}$ moduli to achieve a given knowledge error which guarantees the soundness property of the generated protocol as described in §3. All three optimizations for fixed length moduli, optimal communication complexity, or optimal computation complexity are supported (see §4.2 for details). The output is the **Protocol Implementation** which consists of the different rounds of the protocol and the operations (e.g., group operations, random choice of elements, ...) performed therein. Our *Protocol Implementation Language (PIL)* describes the protocol implementation in pseudo-code.

**Plugins.** The protocol implementation of the generated $\Sigma$-protocols can then be transformed with several plugins: The **Σ2NIZK** plugin transforms the generated $\Sigma$-protocol into a Non-Interactive Zero-Knowledge protocol (NIZK) respectively a Signature-Proof of Knowledge by applying the Fiat-Shamir transformation [40]. The **Costs** plugin counts the *abstract costs* of the generated protocol, i.e., the communication complexity and the number of operations in each group. In order to automatically determine the *real costs* of the protocol on a specific platform (in kBytes, CPU cycles and milliseconds), this plugin is coupled to the code generation backend.

**Backends.** Multiple backends allow to transform the protocol implementation into source code in various output languages: The **C Backend** generates source code in the C program-

ming language for prover and verifier. The **LaTeX Backend** generates a human-readable documentation of the protocol in LaTeX. An example output will be given in the full version of this paper.

### 4.2 Protocol Specification Language (PSL)

Here, we describe the rationale underlying our Protocol Specification Language (PSL) and show how to formulate a given proof goal in this language using the following example:

We want to prove that a committed integer value lies in an interval. This need arises in many applications such as electronic cash systems [32], group signature schemes [21], watermarking [1], or verifiable encryption [28]. Such efficient interval proof techniques were given by Boudot [16] and Lipmaa [55]. The core of both techniques is to prove that a number committed within a Damgård-Fujisaki integer commitment [37] is a square. Recall, the Damgård-Fujisaki integer commitment scheme is $Com(m,r) = g^m \cdot h^r \mod n'$, where $n'$ is an RSA modulus, $g$ is a generator of the quadratic residues modulo $n'$, $h \in \langle g \rangle$ and the factorization of $n'$ as well as the discrete logarithm between $g$ and $h$ are unknown to $\mathsf{P}$. The corresponding proof goal can be formulated in Camenisch-Stadler notation [29] as

$$\mathsf{ZPK}\left[(m,r') : x' = g^{m^2} \cdot h^{r'} \mod n'\right].$$

This can be rewritten algebraically to the following:

$$\mathsf{ZPK}\left[(m,r,r') : x = g^m \cdot h^r \mod n' \quad \wedge \quad x' = x^m \cdot h^{r'-m\cdot r} \mod n'\right].$$

Using the homomorphism $\phi : \mathbb{Z}^3 \to {\mathbb{Z}_{n'}^*}^2 : (a,b,c) \mapsto (g^a \cdot h^b \mod n', x^a \cdot h^c \mod n')$ we get:

$$\mathsf{ZPK}\left[(m,r,r') : (x,x') = \phi(m,r,r'-m\cdot r)\right].$$

This proof goal together with the underlying algebraic setting can directly be formulated in PSL as shown in Fig. 4 and described next. A PSL file consists of the following four sections:

*Declarations.* All variables used in the protocol must be declared in this section first. PSL supports several data types with a given bitlength such as signed integers, unsigned integers, RSA moduli, or primes. Also intervals and predefined multiplicative and additive groups such as $(\mathbb{Z}_n^*, *)$ or $(\mathbb{Z}_p, +)$ are supported. During declaration, an identifier can be assigned to a group and constants can be predefined. The compiler can easily be extended by implementing abstract groups with one's favorite group (e.g., to support elliptic curve groups).

*Inputs.* In this section, the inputs of the protocol are assigned to prover and verifier by specifying which variables are private to the prover and which are publicly known to both parties. All protocol inputs must have been declared before.

*Protocol.* In the protocol section, the properties of the protocol to be generated are specified. For instance, `KnowledgeError := 80` describes an intended knowledge error $\kappa$ of $2^{-80}$. Analogously, the `SZKParameter` specificies the intended tightness of the statistical ZK property. Afterwards, the protocol is composed by combining several $\Sigma$-protocols which prove a given predicate with a monotone boolean formula consisting of arbitrary combinations of AND, OR, and $n$-out-of-$k$-threshold composition [35] implemented over Shamir's secret sharing scheme [67].

*Predicates.* Finally, the predicates used in the protocol composition are specified. Each predicate is proven with a $\Sigma$-protocol, i.e., either `SigmaPhi` (cf. §A), `SigmaGSP` or `SigmaPort` for the optimal unconditionally portable $\Sigma$-protocol (see below). For each $\Sigma$-protocol, the underlying homomorphism and the relation between public images and private preimages under this homomorphism are specified. Linear relations between preimages can be proven explicitly as shown in the example.

**Automatic Choice of Unconditionally Portable Protocols.** The compiler is capable to automatically determine the optimal unconditionally portable $\Sigma$-protocol by choosing between the $\Sigma^{\mathsf{exp}}$-protocol (cf. §2.1) and the $\Sigma^{\mathsf{GSP}}$-protocol with binary challenges (cf. App. A) and optimizing the modulus size according to §3 as follows:

`SigmaPort(T=1248,t=80)` uses $T = 1248$ bit RSA moduli which correspond to $t = 80$ bit symmetric security and computes the number of parallel executions of the protocol to achieve the intended knowledge error as described in §3.1. Recommended sizes for $T$ and $t$ can be found in [9,42,50,52,58].

`SigmaPort(opt_communication)` minimizes the communication complexity of the protocol by automatically choosing the bitlength of the single RSA modulus to achieve the intended knowledge error in one repetition using the techniques described in §3.1.

`SigmaPort(opt_computation)` optimizes for minimum computation complexity of `P` by choosing the optimal modul length: Currently, this optimization uses the formulas given in §3.2. Future versions of the compiler could additionally use the code generation backend to determine the parameters for minimal costs of a given protocol on a specific platform.

## References


1. A. Adelsbach and A.-R. Sadeghi. Zero-knowledge watermark detection and proof of ownership. In *Information Hiding*, volume 2137 of *LNCS*, pages 273–288. Springer, 2001.
2. D. Aggarwal and U. Maurer. Breaking RSA generically is equivalent to factoring. In *EUROCRYPT 09*, volume 5479 of *LNCS*, pages 36–53. Springer, 2009.
3. G. Ateniese, J. Camenisch, M. Joye, and G. Tsudik. A practical and provably secure coalition-resistant group signature scheme. In *CRYPTO 00*, volume 1880, pages 255–270. Springer, 2000.
4. R. Avanzi. On multi-exponentiation in cryptography, 2002.
5. E. Bangerter. *Efficient Zero-Knowledge Proofs of Knowledge for Homomorphisms.* PhD thesis, Ruhr-University Bochum, 2005.
6. E. Bangerter, S. Barzan, S. Krenn, A.-R. Sadeghi, T. Schneider, and J.-K. Tsay. Bringing zero-knowledge proofs of knowledge to practice. In *International Workshop on Security Protocols – SPW 09*, LNCS. Springer, 2009. Preliminary version available at `http://eprint.iacr.org/2009/211`.
7. E. Bangerter, J. Camenisch, S. Krenn, A.-R. Sadeghi, and T. Schneider. Automatic generation of sound zero-knowledge protocols. In poster session of *EUROCRYPT 09*, 2009.
8. E. Bangerter, J. Camenisch, and U. Maurer. Efficient proofs of knowledge of discrete logarithms and representations in groups with hidden order. In *PKC 05*, volume 1992 of *LNCS*, pages 154–171. Springer, 2005.
9. E. Barker, W. Barker, W. Burr, W. Polk, and M. Smid. NIST SP800-57: Recommendation for key management – part 1: General(revised). Technical report, NIST, March 2007.
10. M. Bellare and O. Goldreich. On defining proofs of knowledge. In *CRYPTO 92*, volume 740 of *LNCS*, pages 390–420. Springer, 1993.
11. M. Bellare and P. Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. In *ACM CCS 93*, pages 62–73. ACM, 1993.

12. D. Bernstein. Pippenger's exponentiation algorithm. `http://cr.yp.to/papers/pippenger.pdf`, 2002.
13. A. Bhargav-Spantzel, A. C. Squicciarini, S. Modi, M. Young, E. Bertino, and S. J. Elliott. Privacy preserving multi-factor authentication with biometrics. *Journal of Computer Security*, 15(5):529–560, 2007.
14. M. Blum, P. Feldman, and S. Micali. Non-interactive zero-knowledge and its applications. In *ACM STOC 88*, pages 103–112. ACM, 1988.
15. J. Bos and M. Coster. Addition chain heuristics. In *CRYPTO 89*, volume 435 of *LNCS*, pages 400–407, New York, NY, USA, 1989. Springer.
16. F. Boudot. Efficient proofs that a committed number lies in an interval. In *EUROCRYPT 00*, volume 1807 of *LNCS*, pages 431–444. Springer, 2000.
17. S. Brands. Rapid demonstration of linear relations connected by boolean operators. In *EUROCRYPT 97*, volume 1233 of *LNCS*, pages 318–333. Springer, 1997.
18. E. Bresson and J. Stern. Proofs of knowledge for non-monotone discrete-log formulae and applications. In *ISC 2002*, pages 272–288. Springer, 2002.
19. E. Brickell, J. Camenisch, and L. Chen. Direct anonymous attestation. In *ACM CCS 2004*, pages 132–145. ACM, 2004.
20. T. Briner. Compiler for zero-knowledge proof-of-knowledge protocols. Master's thesis, ETH Zurich, 2004.
21. J. Camenisch. *Group Signature Schemes and Payment Systems Based on the Discrete Logarithm Problem*. PhD thesis, ETH Zurich, Konstanz, 1998.
22. J. Camenisch and E. V. Herreweghen. Design and implementation of the idemix anonymous credential system. In *ACM CCS 2002*, pages 21–30. ACM, 2002.
23. J. Camenisch, A. Kiayias, and M. Yung. On the portability of generalized schnorr proofs. In *EUROCRYPT 09*, volume 5479 of *LNCS*, pages 425–442. Springer, 2009.
24. J. Camenisch and A. Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *EUROCRYPT 01*, volume 2045 of *LNCS*, pages 93–118. Springer, 2001.
25. J. Camenisch and M. Michels. A group signature scheme with improved efficiency. In *ASIACRYPT 98*, volume 1514 of *LNCS*, pages 160–174. Springer, 1998.
26. J. Camenisch and M. Michels. Proving in zero-knowledge that a number is the product of two safe primes. In *EUROCRYPT 99*, volume 1592 of *LNCS*, pages 107–122. Springer, 1999.
27. J. Camenisch, M. Rohe, and A.-R. Sadeghi. Sokrates - a compiler framework for zero-knowledge protocols. In *WEWoRC 05*, 2005.
28. J. Camenisch and V. Shoup. Practical verifiable encryption and decryption of discrete logarithms. In *CRYPTO 03*, volume 2729 of *LNCS*, pages 126–144. Springer, 2003.
29. J. Camenisch and M. Stadler. Efficient group signature schemes for large groups (extended abstract). In *CRYPTO 97*, volume 1294, pages 410–424. Springer, 1997.
30. J. Camenisch and M. Stadler. Proof systems for general statements about discrete logarithms. Technical Report 260, Institute for Theoretical Computer Science, ETH Zürich, 1997.
31. Z. Cao. Analysis of one popular group signature scheme. In *ASIACRYPT 06*, volume 4284 of *LNCS*, page 460466. Springer, 2006.
32. A. Chan, Y. Frankel, and Y. Tsiounis. Easy come - easy go divisible cash. In *EUROCRYPT 98*, volume 1403 of *LNCS*, pages 561–575. Springer, 1998.
33. D. Coppersmith. Modifications to the number field sieve. *Journal Of Cryptology*, 6(3):169–180, 1993.
34. R. Cramer. *Modular Design of Secure yet Practical Cryptographic Protocols*. PhD thesis, CWI and University of Amsterdam, 1996.
35. R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *CRYPTO 94*, volume 839 of *LNCS*, pages 174–187. Springer, 1994.
36. I. Damgård. Efficient concurrent zero-knowledge in the auxiliary string model. In *EUROCRYPT 00*, volume 1807 of *LNCS*, pages 418–430. Springer, 2000.

37. I. Damgård and E. Fujisaki. A statistically-hiding integer commitment scheme based on groups with hidden order. In *ASIACRYPT 02*, volume 2501 of *LNCS*, pages 77–85. Springer, 2002.
38. I. Damgård and M. Jurik. A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In *PKC 01*, LNCS, pages 119–136. Springer, 2001.
39. S. Eikenberry and J. Sorenson. Efficient algorithms for computing the jacobi symbol. In *ANTS-II*, volume 1122 of *LNCS*, pages 225–239. Springer, 1996.
40. A. Fiat and A. Shamir. How to prove yourself: practical solutions to identification and signature problems. In *CRYPTO 86*, volume 263 of *LNCS*, pages 186–194. Springer, 1987.
41. E. Fujisaki and T. Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In *CRYPTO 97*, volume 1294, pages 16–30. Springer, 1997.
42. D. Giry. Keylength – Cryptographic key length recommendation. `http://www.keylength.com`, 2009.
43. S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof-systems. In *ACM STOC 85*, pages 291–304, New York, NY, USA, 1985. ACM.
44. J. Groth. Non-interactive zero-knowledge arguments for voting. In *ACNS 2005*, volume 3531 of *LNCS*, pages 467–482. Springer, 2005.
45. L. Guillou and J. Quisquater. A "paradoxical" identity-based signature scheme resulting from zero-knowledge. In *CRYPTO 88*, volume 403 of *LNCS*, pages 216–231. Springer, 1990.
46. Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. Zero-knowledge from secure multiparty computation. In *ACM STOC 07*, pages 21–30. ACM, 2007.
47. A. A. Karacuba. Berechnungen und Kompliziertheit von Beziehungen. *Elektronische Informationsverarbeitung Kybernetik*, 11:603–606, 1975.
48. C. Koc and C. Hung. Fast algorithm for modular reduction. In *IEE Computers and Techniques – IEEPCDT 98*, page 145. IEEE, 1998.
49. S. Kunz-Jacques, G. Martinet, G. Poupard, and J. Stern. Cryptanalysis of an efficient proof of knowledge of discrete logarithm. In *PKC 06*, volume 3958 of *LNCS*, pages 27–43. Springer, 2006.
50. A. Lenstra. Keylength. In *Handbook of Information Security*, pages 617 – 635. Wiley, 2006.
51. A. Lenstra and H. Lenstra, editors. *The development of the number field sieve*, volume 1554 of *LNM*. Springer, Berlin, 1993.
52. A. K. Lenstra and E. R. Verheul. Selecting cryptographic key sizes. In *PKC 00*, volume 1751 of *LNCS*, pages 446–465. Springer, 2000.
53. Y. Lindell, B. Pinkas, and N. Smart. Implementing two-party computation efficiently with security against malicious adversaries. In *SCN 08*, volume 5229 of *LNCS*, pages 2–20. Springer, 2008.
54. H. Lipmaa. Statistical zero-knowledge proofs from diophantine equations. Technical Report 86, Laboratory for Theoretical Computer Science, Helsinki University of Technology, 2001.
55. H. Lipmaa. On diophantine complexity and statistical zeroknowledge arguments. In *ASIACRYPT 03*, volume 2894 of *LNCS*. Springer, 2003.
56. P. MacKenzie, A. Oprea, and M. K. Reiter. Automatic generation of two-party computations. In *ACM CCS 2003*, pages 210–219. ACM, 2003.
57. D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay — a secure two-party computation system. In *USENIX Security 04*, 2004.
58. M. Näslund. ECRYPT yearly report on algorithms and keysizes (2007-2008). Technical report, ECRYPT, 2008.
59. P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT 99*, volume 1592 of *LNCS*, pages 223–238. Springer, 1999.
60. N. Pippenger. On the evaluation of powers and related problems (preliminary version). In *FOCS 76*, pages 258–263. IEEE, 1976.
61. J. M. Pollard. The fast fourier transform in a finite field. *Mathematics of Computation*, 25(114):365–374, 1971.
62. R. Rivest. Personal communication, 2009.
63. R. Rivest and B. Kaliski Jr. RSA problem. In *Encyclopedia of Cryptography and Security*. Springer, 2005.

64. R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.
65. C. Schnorr. Efficient signature generation by smart cards. *Journal Of Cryptology*, 4(3):161–174, 1991.
66. A. Schönhage and V. Strassen. Schnelle Multiplikation großer Zahlen. *Computing*, 7:281–292, 1971.
67. A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

# A The $\Sigma^{\mathsf{GSP}}$- and $\Sigma^{\phi}$-Protocol

In this section, we briefly recall the Generalized Schnorr Protocol $\Sigma^{\mathsf{GSP}}$, and the $\Sigma^{\phi}$-protocol for special homomorphisms.

**The $\Sigma^{\mathsf{GSP}}$-protocol.** The $\Sigma^{\mathsf{GSP}}$-protocol is a generalization of the well known protocol proposed by Schnorr [65]. It is defined by a prover and a verifier performing the joint computation given in Fig. 5. It can be used for arbitrary exponentiation homomorphisms in groups of known and hidden order. Especially in the case $\mathcal{C} = \{0, 1\}$, i.e. $c^+ = 1$, the protocol is always known to be a statistical HVZK proof of knowledge, not relying on any computational assumptions. We refer to this special case as the $\Sigma^{\mathsf{GSP}_{0,1}}$-protocol. Other conditions for the protocol to be a ZK-PoK are given in Theorem 1 of [23].

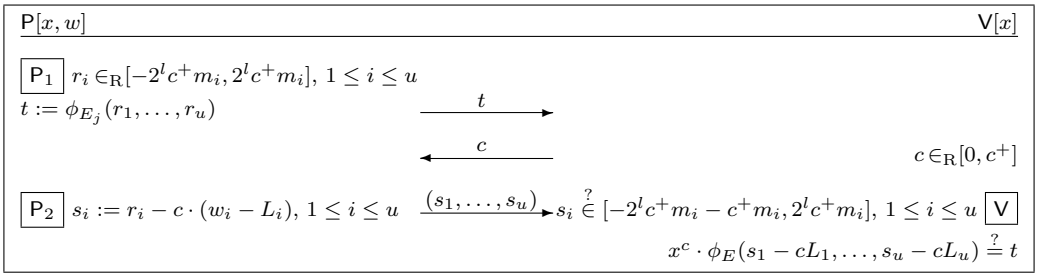


**Fig. 5.** The $\Sigma^{\mathsf{GSP}}$-protocol to prove knowledge of $w = (w_1, \ldots, w_u)$ with $w_i \in [L_i, R_i]$, and $m_i := R_i - L_i$.

As for the $\Sigma^{\mathsf{exp}}$-protocol, each secret witness is associated with an interval it lies in, i.e. $w_i \in [L_i, R_i]$ for all $w_i$. As for the $\Sigma^{\mathsf{exp}}$-protocol, the intervals are only needed to guarantee the protocol to be HVZK.

**The $\Sigma^{\phi}$-protocol.** The $\Sigma^{\phi}$-protocol is given by two parties, $\mathsf{P}$ and $\mathsf{V}$, performing the joint computation depicted in Fig. 6. To be able to give sufficient conditions for the protocol to be HVZK we have to recap the following definition due to Cramer [34]: a homomorphism $\phi : \mathcal{G} \to \mathcal{H}$ is called *special*, if there is a probabilistic polynomial-time algorithm that on input $\phi$ and $x \in \mathrm{Im}\ \phi$ outputs $(u, v) \in \mathcal{G} \times \mathbb{Z} \setminus \{0\}$, such that $x^v = \phi(u)$. For a given $\phi$, the *special exponent* $v$ being output has to be the same for all $x$.

Many homomorphisms used in cryptography are special. For example every homomorphism $\phi$, for which a non-zero multiple $v'$ of the order of $\mathrm{Im}\ \phi$ is known, is special, since $x^{v'} = 1 = \phi(0)$ for all $x \in \mathrm{Im}\ \phi$. Especially, if the order of $\mathcal{H}$ is known, one can set $v' := \mathrm{ord}(\mathcal{H})$. Further,

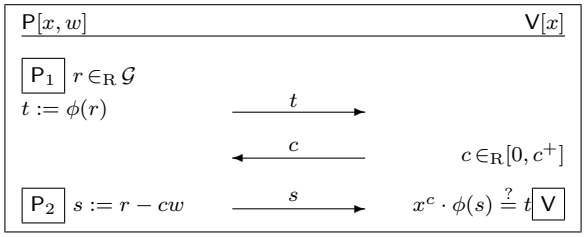


**Fig. 6.** The $\Sigma^\phi$-protocol for a homomorphism $\phi : \mathcal{G} \to \mathcal{H}$.

power-homomorphisms $\phi(w) := w^e$ are special, since for $x = \phi(w)$ we always have $x^e = \phi(x)$. Those are important in many cryptographic schemes such as RSA or the Paillier encryption scheme [38,59,64].

As shown in [34], the $\Sigma^\phi$-protocol is a perfect HVZK proof of knowledge with knowledge error $1/|\mathcal{C}|$, if $\phi$ is a special homomorphism, and $c^+$ is smaller than any prime divisor of its special exponent $v$.

Note, that the $\Sigma^\phi$-protocol especially subsumes the protocols proposed by Schnorr [65] and Guillou-Quisquater [45].

## B Proofs

We call that the Strong Root problem is $(\tau, \varepsilon)$-*secure*, if there is no algorithm solving it in at most $\tau$ steps with probability larger than $\varepsilon$.

The next corollary is directly derived from the results in [37].

**Corollary 7.** *If the Strong Root Problem is $(\tau, \varepsilon)$-secure, and $\tau' \leq \frac{\tau}{448}$, then a DF-based protocol using a safe RSA modulus is a $(18\varepsilon, \tau', \max(\frac{4}{|\mathcal{C}|}, \frac{448\tau'}{\tau}))$ -convincing argument of knowledge.*

*Proof (of Th. 5).* Given a $k$-bit modulus, and $v' \in [0, s(k)]$, the Strong Root problem is $(2^{v'}, 2^{v'-s(k)})$-secure by the definition of $s(k)$. Hence, according to Cor. 7, the protocol is a $(\nu, 2^a, \kappa)$-convincing argument of knowledge, where $\nu = 18 \cdot 2^{v'-s(k)}$ and $\kappa = \max(\frac{4}{|\mathcal{C}|}, 448 \cdot 2^{a-v'})$. As $|\mathcal{C}|$ can be chosen arbitrarily large, we assume $\kappa = 448 \cdot 2^{a-v'}$ for a moment. Minimizing the computational knowledge error $\kappa + \nu$ in $v'$ then yields a minimum of $36 \cdot 2^v$ at $v := \frac{\log_2 448 - \log_2 18 + a - s(k)}{2}$.

Here we used that $\kappa = 18 \cdot 2^v$. To ensure that, we also need that $4/|\mathcal{C}| \leq 18 \cdot 2^v$. This is reached by using at least $\lceil v + \log_2 \frac{9}{2} \rceil$ bit challenges. □

*Proof (of Cor. 6).* The computational knowledge error can now be reduced by repeating the protocol sequentially, each repetition decreasing the success probability of a malicious prover by a factor $36 \cdot 2^v$. Thus, to reach a computational knowledge error of at most $2^{-b}$, the protocol has to be repeated $r$ times, where $r$ satisfies $(36 \cdot 2^v)^r \leq 2^{-b}$. Accordingly we get $r := \lceil -\frac{b}{v + \log_2 36} \rceil$.

Having $s(k) \geq 15 + a + 2b$ directly yields $r = 1$, and hence $b + 3$ bit challenges. □