Full Proof Cryptography: Verifiable Compilation of Efficient Zero-Knowledge Protocols

José Bacelar Almeida
Universidade do Minho
jba@di.uminho.pt

Manuel Barbosa
Universidade do Minho
mbb@di.uminho.pt

Endre Bangerter
Bern Univ. of Appl. Sciences
endre.bangerter@bfh.ch

Gilles Barthe
IMDEA Software Institute
gilles.barthe@imdea.org

Stephan Krenn*
IST Austria
stephan.krenn@ist.ac.at

Santiago Zanella Béguelin
Microsoft Research
santiago@microsoft.com

Abstract

Developers building cryptography into security-sensitive applications face a daunting task. Not only must they understand the security guarantees delivered by the constructions they choose, they must also implement and combine them correctly and efficiently.

Cryptographic compilers free developers from having to implement cryptography on their own by turning high-level specifications of security goals into efficient implementations. Yet, trusting such tools is risky as they rely on complex mathematical machinery and claim security properties that are subtle and difficult to verify.

In this paper, we present ZKCrypt, an optimizing cryptographic compiler that achieves an unprecedented level of assurance without sacrificing practicality for a comprehensive class of cryptographic protocols, known as Zero-Knowledge Proofs of Knowledge. The pipeline of ZKCrypt tightly integrates purpose-built verified compilers and verifying compilers producing formal proofs in the CertiCrypt framework. By combining the guarantees delivered by each stage in the pipeline, ZKCrypt provides assurance that the implementation it outputs securely realizes the high-level proof goal given as input. We report on the main characteristics of ZKCrypt, highlight new definitions and concepts at its foundations, and illustrate its applicability through a representative example of an anonymous credential system.

1. INTRODUCTION

Zero-Knowledge Proofs of Knowledge (ZK-PoKs) [37, 36] are two-party protocols in which a prover convinces a verifier that it knows some secret piece of information satisfying some property without revealing anything except the correctness of this claim. ZK-PoKs allow obtaining assurance on a prover's honest behavior without compromising privacy, and are used in a number of practical systems, including *Direct Anonymous Attestation* (DAA) [17], a privacy-enhancing mechanism for remote authentication of computing platforms, the *identity mixer* [19], an anonymous credential system for user-centric identity management, *Off-the-Record messaging* [34, 15], a protocol enabling deniability in instant messaging protocols, and *privacy-friendly smart metering* [50], an emerging technology for smart meters. However, more than 25 years after their inception [35], the potential of ZK-PoKs has not yet been realized to its full extent, and many interesting applications of ZK-PoKs still only exist at the specification level. In our experience, one main hurdle towards a larger use of ZK-PoKs is the difficulty of designing and correctly implementing these protocols for custom proof goals.

Zero-knowledge compilers [7, 45] are domain-specific compilers that automatically generate ZK-PoKs for a large class of proof goals. They are a promising enabling technology for ZK-PoKs, because they allow developers to build cryptographic protocols that use them, without being experts in cryptography, and without the risk of introducing security flaws in their implementations.

Zero-knowledge compilers embed sophisticated mathematical machinery and as a consequence implementing them correctly can be difficult—arguably more difficult than implementing optimizing compilers. Moreover, this type of compilers cannot be tested and debugged because their purported correctness properties are formulated in the style of provable security, and testing such properties is out of reach of current methods. This state of affairs leaves practitioners with no other option than blindly trusting that the compiler is correct.

Contributions. We present ZKCrypt, a high-assurance zero-knowledge compiler that outputs *formally verified and optimized* implementations of ZK-PoKs for a comprehensive set of proof goals. We consider the class of Σ-protocols for proving knowledge of pre-images under group homomorphisms, which underly essentially all practically relevant applications of ZK-PoKs, including all applications mentioned above and the well-known identification schemes by Schnorr [53] and Guillou-Quisquater [39].

ZKCrypt achieves an unprecedented level of confidence among cryptographic compilers by leveraging and transposing to the realm of cryptography two recent breakthroughs: *verified compilation* [42], in which the correctness of a compiler is proved once and for all, and *verifying compilation* [47, 56], in which the correctness of the output of a compiler is proved for each run. Specifically, ZKCrypt implements a verified compiler that generates a reference implementation, and a verifying compiler that outputs an optimized implementation provably equivalent to the reference implementation. Taken together, the proofs output by the compilers

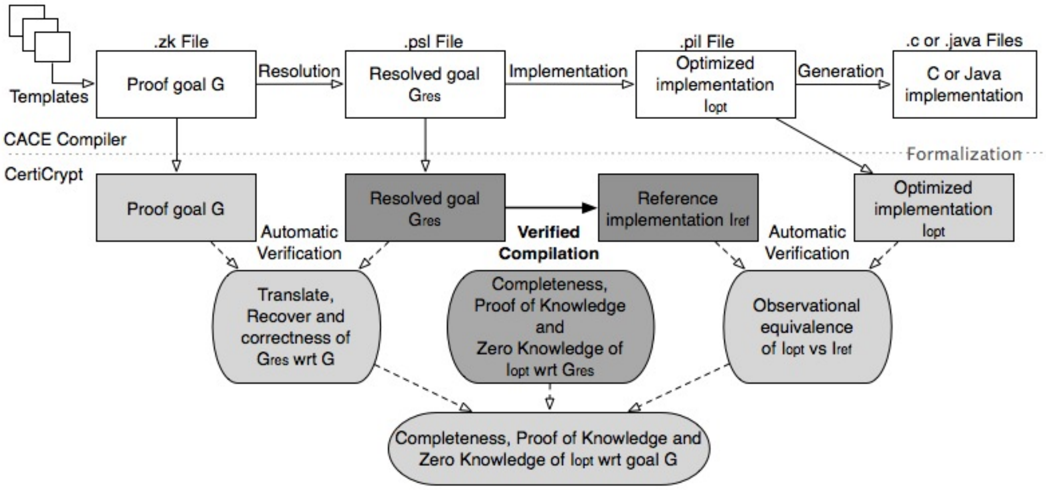

Figure 1: ZKCrypt architecture, depicting a verifying compiler that takes high-level proof goals G to optimized implementations (top), relying on a verified compiler implemented in Coq/CertiCrypt (center). Full lines denote compilation steps and translation over formalization boundary (i.e. the generation of code that can be fed into formal verification tools), dashed lines denote formal verification guarantees. Rectangular boxes denote code in various (intermediate) languages either stored in files or as data structures in memory. Rounded rectangles represent the main theorems that are generated and formally verified by ZKCrypt and which jointly yield the desired formal correctness and security guarantees.

establish that the reference and optimized implementations satisfy the following properties[1] (refer to Section 3.2 for the full definitions):

- *Completeness*: an honest prover can always convince an honest verifier;
- *Proof of knowledge*: a malicious prover cannot convince a verifier without actually knowing the secret, except with small probability;
- *Zero-knowledge*: a verifier following the protocol does not learn additional information about the secret of the prover.

The architecture of the compiler is shown in Figure 1. At the top level, ZKCrypt is composed of a chain of compilation components that generates C and Java implementations of ZK-PoKs; these implementations can be turned into executable binaries using general-purpose compilers. These top level compilation components are an extension of the CACE compiler [2] with support for user-defined templates and high-level proof goals. At the bottom level, ZKCrypt generates formal evidence in CertiCrypt of the correctness of each compilation step except code generation. The top level compiler is independent of the bottom level verification backend.

The main three verification phases in ZKCrypt are: resolution, verified compilation, and implementation. We briefly describe key aspects of each phase:

1. *Resolution* takes a (user-friendly) high-level goal G and outputs an equivalent goal $\mathsf{G}_{\mathsf{res}}$ only consisting of proofs of knowledge of pre-images under homomorphisms; such pre-image proofs constitute atomic building blocks that correspond to well known, concrete instances of ZK-PoK protocols. The correctness of resolution is captured by a transformation that provably converts ZK-PoK protocols for $\mathsf{G}_{\mathsf{res}}$ into ZK-PoK protocols for G. The compiler implements both the decomposition and the transformation, and we prove a set of sufficient conditions for correctness and security;
2. *Verified compilation* takes a resolved goal $\mathsf{G}_{\mathsf{res}}$ and outputs a reference implementation $\mathsf{I}_{\mathsf{ref}}$ in the embedded language of CertiCrypt.A once-and-for-all proof of correctness guarantees that this component only produces reference implementations that satisfy the relevant security properties, for all supported input goals. This result hinges on two contributions of independent interest: a unified treatment of the proof of knowledge property and a formalization of statistical zero-knowledge;
3. *Implementation* takes a resolved goal $\mathsf{G}_{\mathsf{res}}$ and outputs an optimized implementation $\mathsf{I}_{\mathsf{opt}}$. The correctness of this step is established, in the style of verifying compilation, using an equivalence checker proving semantic equivalence between the reference and optimized implementations $\mathsf{I}_{\mathsf{ref}}$ and $\mathsf{I}_{\mathsf{opt}}$.

Combining the correctness results for each phase yields a proof that the optimized implementation $\mathsf{I}_{\mathsf{opt}}$ satisfies the security properties of the original high-level goal G.

Limitations. Although the verification component of our

[1]In the remainder of the paper, when we refer to the (relevant) security properties of a ZK-PoK, we mean these three properties.

compiler is comprehensive, it currently has two limitations that we describe next.

First, ZKCrypt delivers formal guarantees about the correctness of the optimized implementation $\mathsf{I}_{\mathsf{opt}}$, but not for the last step in the compilation chain, namely the generation of Java or C code. Although we consider the verification of this last compilation step as an important direction for future development, we see this as an independent line of work, notably since the verification goals involved at this level are of a different nature to those presented in this paper. Specifically, the natural path to achieve correctness guarantees about binaries is to extend ZKCrypt with formal verification at the code generation level, and then to use a high-assurance compiler from C or Java to binaries. Given the characteristics of the programming language in which the optimized implementations $\mathsf{I}_{\mathsf{opt}}$ are described (cf. Appendix C), the key step to adding a formal verification back-end for code generation is to build a certified number theory library that matches the one provided with the CACE compiler. Then, the compilation from C to binaries can be certified directly using state-of-the-art verified compilers, such as CompCert [42].

Second, we do not prove completeness of verification, i.e., that all CACE compiler program can be verified by our component. There are two reasons for this. One is that there are some known sources of incompleteness, as some of the proof goals that can be handled by the CACE compiler are not yet supported. A more fundamental reason is that certifying compilation techniques, as used by part of the formal verification back-end, are seldom proved complete; instead, one validates the effectiveness of a technique by its ability to cover a wide range of examples. As we will demonstrate later in the paper, ZKCrypt satisfactorily complies with this more practical view, as the class of goals for which verification is available is already broad enough to cover most practical applications.

Paper organization. The applicability of ZKCrypt is illustrated in Section 2 through a use case from the *identity mixer* anonymous credential system [19]. Section 3 provides some necessary background material on CertiCrypt and ZK-PoKs. Sections 4–6 describe the resolution, verified compilation and implementation phases, respectively. Section 7 briefly reports on experimental results obtained by applying ZKCrypt to a wider range of examples. We give an overview of related work in Section 8 and briefly conclude in Section 9.

2. USE CASE

Anonymous credential systems [24, 25] are among the most practically relevant applications of ZK-PoKs; examples of prominent realizations include the IBM identity mixer library (Idemix) [23], the Microsoft U-Prove toolkit [46], as well as Trusted Platform Modules (TPMs), which implement the Direct Anonymous Attestation (DAA) protocol [17] and are widely built into consumer devices. In a further and coordinated effort to bring anonymous credential systems to practice, the ABC4Trust project [1] is working to deliver open reference implementations of attribute-based credential systems and to integrate them into real-world identity-management systems.

An anonymous credential system typically consists of a collection of protocols to issue, revoke, prove possession of credentials, etc. One key feature for ensuring anonymity is that users can selectively reveal certain identity attributes without disclosing anything else. Our running example is extracted from Idemix, and involves a user with a valid credential of his name m_1 and his birthdate m_2; following Camenisch et al. [23] we assume that the credential is a valid Camenisch-Lysanskaya (CL) signature [20] on m_1 and m_2 issued by some certification authority. Assume the user wants to authenticate to a server and is willing to reveal his name but not his birthdate. On the other hand, he is required to show that he was born after a certain date b. To achieve authentication agreeably to both parties, the user will reveal m_1 and give a ZK-PoK that z is a valid CL signature on m_1, m_2, where $m_2 \geq b$, without revealing m_2. Using the standard notation for ZK-PoK [22], this goal G is formally stated as:

$$\mathsf{ZPK}\Big[m_2 : z = \mathsf{CL}(m_1, m_2) \,\wedge\, m_2 \geq b\Big]$$

The convention in the formulation above is that knowledge of all values before the colon has to be proved, while all other values are assumed to be publicly known. Note that the first conjunct shows possession of a valid CL-signature z on m_1, m_2, and the second conjunct shows that $m_2 \geq b$, as required by the server policy.

ZKCrypt generates an optimized implementation of a ZK-PoK and machine-checkable proofs that it satisfies the relevant security properties. We give below an overview of the compilation process.

Resolution. ZKCrypt resolves the above proof goal G to the following goal $\mathsf{G}_{\mathsf{res}}$:

$$\begin{aligned}\mathsf{ZPK}\Big[\,&(e, m_2, v, r_\Delta, u_1, u_2, u_3, u_4, r_1, r_2, r_3, r_4, \alpha) : \\ &\frac{Z}{R_1^{m_1}} = A^e S^v R_2^{m_2} \,\wedge\, T_\Delta Z^b = Z^{m_2} S^{r_\Delta} \,\wedge \\ &\bigwedge_{i=1}^{4} T_i = Z^{u_i} S^{r_i} \,\wedge\, T_\Delta = T_1^{u_1} T_2^{u_2} T_3^{u_3} T_4^{u_4} S^{\alpha}\Big]\end{aligned}$$

The first conjunct is obtained by unfolding the definition of the CL predicate, making explicit the groups elements Z and S used in the signature. The remaining conjuncts are obtained by applying Lipmaa's technique [43] to resolve the goal $m_2 \geq b$ into equalities between exponentiations of Z and S. This compilation step and the formal verification of its correctness are described in detail in Section 4. In short, we formalize the sufficient conditions for correctness based on a procedure Translate for turning a witness for G into a witness for $\mathsf{G}_{\mathsf{res}}$ and a procedure Recover for computing a witness for $\mathsf{G}_{\mathsf{res}}$ from a witness for G.

Verified compilation This phase outputs a reference implementation $\mathsf{I}_{\mathsf{ref}}$ in the embedded language of CertiCrypt. This is done in two steps: first, ZKCrypt extends the base language of CertiCrypt by specifying and defining the necessary algebraic constructions, e.g., the underlying group with its operations and generators. Second, the compiler instantiates a CertiCrypt module for Σ-protocols with the resolved goal $\mathsf{G}_{\mathsf{res}}$ expressed using a single homomorphism Φ. In the use case we present, Φ is defined as:

$$\Phi\left(e, m_2, v, r_\Delta, u_1, u_2, u_3, u_4, r_1, r_2, r_3, r_4, \alpha\right) \doteq \\ \left(A^e S^v R_2^{m_2}, Z^{m_2} S^{r_\Delta}, Z^{u_1} S^{r_1}, \ldots, Z^{u_4} S^{r_4}, S^\alpha \ \prod_{i=1}^4 T_i^{u_i}\right)$$

This instantiation yields a reference implementation $\mathsf{I}_{\mathsf{ref}}$ of a ZK-PoK protocol, comprising four procedures that represent the computations performed by each party during a run of a Σ-protocol. Each procedure consists of either a random assignment followed by a deterministic assignment, or just a deterministic assignment. The completeness, proof of knowledge, and honest verifier zero-knowledge (HVZK) properties of $\mathsf{I}_{\mathsf{ref}}$ are a direct consequence of generic proofs in CertiCrypt. Interestingly, statistical HVZK is established using an approximate version of the Probabilistic Relational Hoare Logic of CertiCrypt, which has been recently developed for different purposes [11].

Implementation This phase outputs a representation $\mathsf{I}_{\mathsf{opt}}$ that can be used for code generation. Contrary to the reference implementation, $\mathsf{I}_{\mathsf{opt}}$ does not adhere to a constrained shape; in particular, it uses long sequences of instructions and branching statements. Also, algebraic expressions are re-arranged in order to enable optimizations during code generation. The equivalence of $\mathsf{I}_{\mathsf{opt}}$ and $\mathsf{I}_{\mathsf{ref}}$ is proved in two steps. First, ZKCrypt builds a representation of $\mathsf{I}_{\mathsf{opt}}$ in the embedded language of CertiCrypt. It then generates a proof that $\mathsf{I}_{\mathsf{opt}}$ satisfies the relevant security properties, by establishing that each algorithm in the protocol is observationally equivalent to the matching algorithm in the reference implementation.

By glueing together the correctness proofs of the different phases, one obtains the end-to-end guarantee that $\mathsf{I}_{\mathsf{opt}}$ is a correct implementation of a ZK-PoK for G.

3. PRELIMINARIES

3.1 An Overview of CertiCrypt

CertiCrypt [10, 12] is an automated toolset for proving the security of cryptographic constructions in the computational model. It builds upon state-of-the-art verification technologies to support code-based proofs, in which security is cast in terms of equivalence of probabilistic programs. The core of CertiCrypt is a rich set of verification techniques based on a Relational Hoare Logic for probabilistic programs [10]. A recent extension [11] supports reasoning about a broad range of quantitative properties, including statistical distance, which is crucial in our definition of zero-knowledge.

The CertiCrypt toolset consists of two main components. Both allow proving that the distributions generated by probabilistic experiments are identical or statistically close, but differ in their degree of automation, flexibility and formal guarantees. The first component, called CertiCrypt, excels in flexibility and is fully formalized in the Coq proof assistant; its verification methods are implemented in Coq and proved correct w.r.t. program semantics. The second component, EasyCrypt, delivers a higher degree of automation by relying on SMT solvers and automated theorem provers to discharge verification conditions arising in proofs. EasyCrypt generates proof certificates that can be mechanically checked in Coq, thus practically reducing the trusted computing base to that of the first component; however, it lacks on generality as it only exposes a limited set of proof methods. ZKCrypt takes advantage of both components: it uses the latter to check the correctness of goal resolution and the former for verifying the compiler for reference implementations and the equivalence of reference and optimized implementations. We outline below some of the essential features of both components.

Language. Programs are written in the procedural, probabilistic imperative language pWHILE. The statements of the language include deterministic and probabilistic assignments, conditional statements and loops, as given by the following grammar:

$\mathcal{C}$	$::=$	skip	nop
	$\mid$	$\mathcal{V} \leftarrow \mathcal{E}$	deterministic assignment
	$\mid$	$\mathcal{V} \xleftarrow{\$} \mathcal{DE}$	probabilistic assignment
	$\mid$	$\mathsf{if}\ \mathcal{E}\ \mathsf{then}\ \mathcal{C}\ \mathsf{else}\ \mathcal{C}$	conditional
	$\mid$	$\mathsf{while}\ \mathcal{E}\ \mathsf{do}\ \mathcal{C}$	while loop
	$\mid$	$\mathcal{V} \leftarrow \mathcal{P}(\mathcal{E}, \ldots, \mathcal{E})$	procedure call
	$\mid$	$\mathcal{C};\ \mathcal{C}$	sequence

where $\mathcal{V}$ is a set of variable identifiers, $\mathcal{P}$ a set of procedure names, $\mathcal{E}$ is a set of expressions, and $\mathcal{DE}$ is a set of distribution experssions.

This base language suffices to conveniently express a wide class of cryptographic experiments and security properties. However, to achieve greater flexibility, the language of deterministic and random expressions is user-extensible. A program $c \in \mathcal{C}$ in the language of CertiCrypt denotes a function $[\![c]\!]$ from an initial memory m (a mapping from program variables to values) to a distribution over final memories. We denote by $\Pr\left[c, S : m\right]$ the probability of event S w.r.t to the distribution $[\![c]\!]\ m$. We refer the reader to Barthe et al. [14] for a more detailed description of the language and its semantics.

Reasoning principles. Proving the (approximate) equivalence of the distributions generated by two probabilistic programs in CertiCrypt amounts to deriving valid judgments in an approximate Relational Hoare Logic (apRHL). We restrict our attention in this paper to a fragment of apRHL that captures both perfect and statistical indistinguishability of distributions generated by programs. We consider judgments of the form

$$c_1 \sim_\epsilon c_2 : \Psi \Rightarrow \Phi \ , \tag{1}$$

where c_1 and c_2 are probabilistic programs, Ψ, Φ binary relations over program memories and $\epsilon \in [0, 1]$. Taking Φ as the equality relation on a subset of observable program variables X, one recovers the usual definition of statistical indistinguishability. In particular, given an event A, represented as a predicate over memories, if A only depends on variables in X, one has

$$m_1\ \Psi\ m_2 \implies \left|\Pr\left[c_1, m_1 : A\right] - \Pr\left[c_2, m_2 : A\right]\right| \leq \epsilon \ .$$

We let $c_1 \approx_\epsilon^{\Psi, X} c_2$ denote the validity of (1) when Φ is the equality relation on variables in X; we omit Ψ when it is the total relation or can be inferred from the context.

3.2 Zero-Knowledge Proofs

All ZK-PoK protocols generated by ZKCrypt are Σ-protocols:

Definition 1 (Σ-protocol). *Let* R *denote a binary relation,* $(x, w) \in \mathsf{R}$*, and let* $\mathsf{P}_1, \mathsf{P}_2$*, and* V *denote arbitrary algorithms. A protocol between a prover* $\mathcal{P} \doteq \mathcal{P}(x, w)$ *and a probabilistic polynomial-time (PPT) verifier* $\mathcal{V} \doteq \mathcal{V}(x)$ *is called a* Σ-protocol *with* challenge set $\mathcal{C} = \{0, \ldots, c^+ - 1\}$*, if it satisfies the following conditions.*

3-move form. *The protocol is of the following form:*

- $\mathcal{P}$ *sets* $(r, \mathrm{st}) \leftarrow \mathsf{P}_1(x, w)$ *and sends* r *to* $\mathcal{V}$*;*
- $\mathcal{V}$ *sends a random challenge* $c \stackrel{\$}{\leftarrow} \mathcal{C}$ *to* $\mathcal{P}$*. We refer to the algorithm that samples the challenge as* V_c*;*
- $\mathcal{P}$ *sends* $s \leftarrow \mathsf{P}_2(x, w, \mathrm{st}, c)$ *to the verifier;*
- $\mathcal{V}$ *accepts if* $\mathsf{V}(x, r, c, s) = \mathrm{true}$*, otherwise* $\mathcal{V}$ *rejects.*

Completeness. *For an honest prover* $\mathcal{P}$*, the verifier* $\mathcal{V}$ *accepts on common input* x*, whenever* $(x, w) \in \mathsf{R}$*.*

A triple (r, c, s) for which $\mathsf{V}(x, r, c, s) = \mathrm{true}$ is called an *accepting conversation*.

Informally, a two-party protocol is a proof of knowledge if from every successful (potentially malicious) prover $\mathcal{P}^*$, a witness can be extracted within a certain time bound by a *knowledge extractor* algorithm. For all practically relevant Σ-protocols, the knowledge extractor works in two phases. First, using rewinding black-box access to $\mathcal{P}^*$, two accepting conversations (r, c, s) and (r, c', s') are extracted. Then, in a second step, a witness is computed from these conversations. The first part of the knowledge extractor is well-known to work for arbitrary Σ-protocols [28]. The second phase only works only under certain conditions, which are formalized next:

Definition 2 (Generalized Special Soundness). *A* Σ*-protocol satisfies* generalized special soundness *for a relation* R'*, if there is a PPT algorithm that, on input a relation* $\mathsf{R} \stackrel{\$}{\leftarrow} \mathcal{R}(1^\lambda)$*, a value* x *in the language defined by* R*, and any two accepting conversations,* (r, c, s) *and* (r, c', s') *satisfying* R'*, computes a valid witness* w *satisfying* $\mathsf{R}(x, w)$ *with overwhelming probability.*

Observe that a Σ-protocol satisfying this definition is a proof of knowledge for R if the conversations extracted by the first phase of the knowledge extractor satisfy R'.

Definition 2 is a generalization of the classical notion of special soundness found in the literature, e.g., Cramer [26], and enables a uniform formalization of the proof of knowledge property for all Σ-protocols supported by ZKCrypt. Roughly, the *special extractor* algorithm will only be able to recover a valid witness if the accepting conversations display specific properties captured by relation R'. Furthermore, in some cases, the existence of an algorithm that is able to extract these conversations relies on a computational assumption. To account for this, following Damgård and Fujisaki [29], we allow relation R to be sampled using an efficient algorithm $\mathcal{R}$. We will detail $\mathcal{R}$ and R' for each concrete instance of a Σ-protocol later[2].

The proof of knowledge property ensures a verifier that a convincing prover indeed knows the secret. On the other hand, the verifier should not be able to deduce any information about this witness. This is captured by the *zero-knowledge* property. In the following, we denote by $\mathrm{view}_{\mathcal{V}}^{\mathcal{P}}(x)$ the random variable describing the content of the random tape of $\mathcal{V}$ and the messages $\mathcal{V}$ receives from $\mathcal{P}$ during a successful protocol run on common input x.

Definition 3 (Honest Verifier Zero Knowledge). *A protocol* $(\mathcal{P}, \mathcal{V})$ *is perfectly (resp. statistically)* honest-verifier zero-knowledge *(HVZK), if there exists a PPT* simulator S *such that the distribution ensembles* $\{\mathsf{S}(x)\}_x$ *and* $\{\mathrm{view}_{\mathcal{V}}^{\mathcal{P}}(x)\}_x$ *are perfectly (resp. statistically) indistinguishable, for all inputs* x *in the language of* R*.*

Note that this definition only gives guarantees against verifiers that do not deviate from the protocol specification. Security against arbitrary verifiers can be realized by applying the Fiat-Shamir heuristic [32] to make the protocol non-interactive, which is also supported by our compiler (although, as we will explain later, this is currently outside of the scope of the verification back-end). Further, other standard techniques to solve this problem exist [31].

3.3 The Σ^{GSP}-Protocol

Almost all practical applications of ZK-PoKs are proofs for pre-images under a group homomorphism $\phi : \mathcal{G} \to \mathcal{H}$. Depending on whether $\mathcal{G}$ is finite or $\mathcal{G} \simeq \mathbb{Z}$, (typically) either the Σ^ϕ-, the Σ^{exp}-, or the Σ^{GSP}-protocol is used [2]. In the following we recapitulate the Σ^{GSP}-protocol, which is central for understanding the remainder, as it is used for the running example. For self-containment, the Σ^ϕ- and the Σ^{exp}-protocols (a far less prevalent version of the Σ^{GSP}-protocol) are included in Appendix A.1 and Appendix A.2.All the mentioned techniques are also incorporated in ZKCrypt.

The so called *Generalized Schnorr Protocol* (Σ^{GSP}-protocol) can be used to prove knowledge of pre-images under arbitrary exponentiation homomorphisms, in particular including such with a hidden-order co-domain. That is, it can be used for mappings of the form:

$$\phi : \mathbb{Z}^m \to \mathcal{H} : (w_1, \ldots, w_m) \mapsto \left(\prod_{i=1}^m g_{1i}^{w_i}, \ldots, \prod_{i=1}^m g_{ui}^{w_i} \right).$$

In the protocol, an upper bound T_i on the absolute value of each w_i needs to be known. These values can be chosen arbitrarily large, and required to assert that the protocol is (statistically) zero-knowledge for $w_i \in [-T_i, T_i]$. The protocol flow and the parties' algorithms are given in Figure 2, where ℓ is a security parameter regulating the tightness of the statistical zero-knowledge property.

[2]For the completeness and zero-knowledge properties, we quantify over all relations R in the range of $\mathcal{R}$.

$$
\begin{array}{lcr}
\mathsf{P}(x,(w_1,\ldots,w_m)) & & \mathsf{V}(x) \\
\hline
k_i \xleftarrow{\$} [-2T_i c^+ 2^\ell, 2T_i c^+ 2^\ell],\ 1 \le i \le m & & \\
r \leftarrow \phi(k_1,\ldots,k_m) & & \\
 & \xrightarrow{\quad r \quad} & \\
 & \xleftarrow{\quad c \quad} & c \xleftarrow{\$} \mathcal{C} = \{0,\ldots,c^+ - 1\} \\
s_i \leftarrow k_i + c(w_i + T_i),\ 1 \le i \le m & & \\
 & \xrightarrow{\ s_1,\ldots,s_m\ } & \\
 & & r x^c \stackrel{?}{=} \phi(s_1 - cT_1,\ldots,s_m - cT_m) \\
 & & s_i \stackrel{?}{\in} [-2T_i c^+ 2^\ell, 2T_i c^+ 2^\ell + 2T_i(c^+ - 1)],\ 1 \le i \le m
\end{array}
$$

Figure 2: Protocol flow of the Σ^{GSP}-protocol.

The Σ^{GSP}-protocol is statistically HVZK for arbitrary values of c^+ (the simulation error is upper-bounded by $m/2^\ell$)and is sound for $c^+ = 2$, i.e., for binary challenges. However, for larger challenge sets, which are are required for efficiency reasons, generalized special soundness can only be established under certain computational assumptions.

Concerning generalized special soundness, the relation generator $\mathcal{R}$ picks a group $\mathcal{H}$ in which the generalized strong RSA assumption [33] holds (i.e., given $x \xleftarrow{\$} \mathcal{H}$, it is hard to find $(w, e) \in \mathcal{H} \times \mathbb{Z} \setminus \{-1, 0, 1\}$ such that $x = w^e$), and defines:

$$\mathsf{R}(x, (\mu, w)) \doteq x = \mu\phi(w) \ \wedge \ \mu^d = 1,$$

Here, ϕ is as before, the g_{ij} are generators of a large subgroup of $\mathcal{H}$ with hidden order such that all relative discrete logarithms (i.e., all $\log_{g_{ij}} g_{uv}$) are unknown to $\mathcal{P}$, and d is the product of all primes smaller than c^+ dividing $\operatorname{ord}\mathcal{H}$.

The relation R' is defined as follows:

$$\mathsf{R}'((r, c, s), (r, c', s')) \doteq (c - c')|(s - s').$$

It can be shown [29] that the conversations extracted in the first step of the knowledge extractor satisfy R' with overwhelming probability, given that $\mathcal{H}$ satisfies the generalized RSA assumption and ϕ is defined as above. Thus, the Σ^{GSP}-protocol satisfies Definition 2 and is a ZK-PoK for R'.

For instance, if $n = pq$ is a safe RSA modulus (i.e., p, q, $(p - 1)/2$ and $(q - 1)/2$ are all prime), $\mathcal{H} = \mathbb{Z}_n^*$ and the g_{ij} all generate the quadratic residues modulo n, we get $d = 4$ and knowledge of a pre-image is proved up to a fourth root of unity.

3.4 Combination of Proof Goals

In practice one often has to prove knowledge of multiple, or one out of a set of secret values in one step. This can be achieved by And- and Or-compositions [27]. To support our description of the Idemix example in the rest of the paper we only require And-compositions of Σ^{GSP}-protocols for homomorphisms $\phi_1, \ldots, \phi_n$. This can be realized by running a single Σ^{GSP}-protocol for the homomorphism $\phi = \phi_1 \times \cdots \times \phi_n$. Generic constructions for Boolean And- and Or-compositions are given in Appendix A.3.

4. GOAL RESOLUTION

$\mathsf{ZKCrypt}$ generates implementations for arbitrary Boolean And- and Or-compositions of pre-image proofs under group homomorphisms and claims on the size of secrets. Formally, the class of proof goals supported by the $\mathsf{ZKCrypt}$ front-end can be defined as follows. Let $x_1, \ldots, x_n$ be public group elements, or expressions only containing public values, and let, for $1 \le i \le n$, $\phi_i : \mathcal{G}_{i1} \times \cdots \times \mathcal{G}_{im_i} \to \mathcal{H}_i$ be arbitrary and potentially different group homomorphisms, where the $\mathcal{G}_{ij}$ are either arbitrary finite groups $\mathbb{Z}$. Then every goal which can logically be rewritten to the following form is supported:

$$\mathsf{ZPK}\Big[(w_{11}, \ldots, w_{1m_1}, \ldots, w_{n1}, \ldots, w_{nm_n}) :$$
$$\bigvee_{G \in \Gamma} \bigwedge_{i \in G} x_i = \phi_i(w_{i1}, \ldots, w_{im_i}) \ \wedge \bigwedge_{(i,j) \in S} w_{ij} \in [L_{ij}, R_{ij}]\Big],$$

where $\Gamma \subseteq 2^{\{1,\ldots,n\}}$ and $\mathcal{G}_{ij} = \mathbb{Z}$ for all $(i, j) \in S$. Further, the L_{ij}, R_{ij} are public integers or $\pm\infty$. Note that $L_{ij} = -\infty$ is equivalent to $w_{ij} \le R_{ij}$ and similar for $R_{ij} = \infty$.

The first compilation step consists of rewriting all semantic expressions to pre-image proofs, i.e., every term $w \in [L, R]$ is rewritten to a proof specification of the following form [43, 41]:

$$\mathsf{ZPK}\Big[(w, r, w_1, \ldots, w_8, r_1, \ldots, r_8, r_w, r_L, r_R) :$$
$$x_w = g^w h^{r_w} \quad \wedge \quad \bigwedge_{i=1}^{8} x_i = h^{w_i} h^{r_i} \quad \wedge$$
$$x_w g^{-L} = \prod_{i=1}^{4} x_i^{w_i} h^{r_L} \quad \wedge \quad g^R x_w^{-1} = \prod_{i=5}^{8} x_i^{w_i} h^{r_w}\Big]. \qquad (2)$$

Here, g and h are both random generators of a group of hidden order (e.g. the (signed) quadratic residues modulo a safe RSA modulus n). Further, $\log_g h$ and $\log_h g$ must be hard to compute. If such a group is already used in the original proof goal, it can safely be reused.

4.1 A Cryptographic Perspective

We next describe how $\mathsf{ZKCrypt}$ deals with the formal verification of the goal translation stage.

The starting point in the goal resolution procedure is a (high-level) goal G associated to a relation generation algorithm $\mathcal{R}$, i.e., we aim to construct a Σ-protocol for proving (in zero-knowledge) knowledge of a witness w for a public input x such that $\mathsf{R}(x, w)$ holds, for R sampled from $\mathcal{R}$. The resolution procedure first defines a generator for a (lower-level) family of relations $\mathcal{R}_{\mathsf{res}}$ associated with a resolved goal $\mathsf{G}_{\mathsf{res}}$ and then defines a translation algorithm $\mathsf{Translate}(\mathsf{R}, x, w)$ which, on input a relation R and a pair (x, w), produces the description of a relation $\mathsf{R}_{\mathsf{res}}$ and a pair (x', w'). The following properties must be satisfied by the $\mathsf{Translate}$ algorithm:

1. **Completeness.** On a valid input (R, x, w), i.e., where R is in the range of $\mathcal{R}$ and $\mathsf{R}(x, w)$ holds, $\mathsf{Translate}$ outputs triples $(\mathsf{R}_{\mathsf{res}}, x', w')$ such that $\mathsf{R}_{\mathsf{res}}$ is in the range of $\mathcal{R}_{\mathsf{res}}$, and $\mathsf{R}_{\mathsf{res}}(x', w')$ holds.

2. **Soundness.** There is an efficient algorithm $\mathsf{Recover}$ such that, for all PPT adversaries $\mathcal{A}$, the following holds for $\mathsf{R} \xleftarrow{\$} \mathcal{R}$, $(x, w) \in \mathsf{R}$ and $(\mathsf{R}_{\mathsf{res}}, x', w') \leftarrow \mathcal{A}(\mathsf{R}, x, w)$: if $(\mathsf{R}_{\mathsf{res}}, x')$ are in the range of $\mathsf{Translate}(\mathsf{R}, x, w)$ and $\mathsf{R}_{\mathsf{res}}(x', w')$ holds, $\mathsf{Recover}(\mathsf{R}_{\mathsf{res}}, x', w')$ outputs $\tilde{w}$ such that $\mathsf{R}(x, \tilde{w})$ holds with overwhelming probability.

3. **Public verifiability.** The public outputs of $\mathsf{Translate}$, i.e., $\mathsf{R}_{\mathsf{res}}$ and x', can efficiently be checked to be in the correct range for all valid public inputs (R, x).

4. **Simulatability.** There exists an efficient simulator S which, on input R sampled from $\mathcal{R}$ and x in the language that it defines, outputs $(\mathsf{R}_{\mathsf{res}}, x')$ with a distribution identical (or statistically close) to that produced by $\mathsf{Translate}(\mathsf{R}, x, w)$ for a valid witness w.

Now, to construct a protocol for goal G, one first generates descriptions of algorithms P'_1, P'_2 and V' of a Σ-protocol for goal $\mathsf{G}_{\mathsf{res}}$, and then defines the procedures for the high-level protocol as follows:

- $\mathsf{P}_1(x, w)$ runs $\mathsf{Translate}(\mathsf{R}, x, w)$ to get $(\mathsf{R}_{\mathsf{res}}, x', w')$, and $\mathsf{P}'_1(x', w')$ to get (r', st'), and returns
$$(r, st) = ((\mathsf{R}_{\mathsf{res}}, x', r'), (\mathsf{R}_{\mathsf{res}}, x', w', st')).$$
- $\mathsf{P}_2(x, w, c, st)$ recovers $(\mathsf{R}_{\mathsf{res}}, x', w', st')$ from st. It then runs $\mathsf{P}'_2(x', w', st', c)$ to get s' and returns $s = s'$.
- $\mathsf{V}(x, r, c, s)$ recovers $(\mathsf{R}_{\mathsf{res}}, x', r')$ from r and checks that $\mathsf{R}_{\mathsf{res}}$ and x are in the correct range w.r.t. R and x. It then runs $\mathsf{V}'(x', r', c, s)$ and returns the result.

The correctness and security of the resulting protocol is established in the following theorem. We present a proof in Appendix B.

THEOREM 1. *Assume that algorithms P'_1, P'_2 and V' yield a Σ-protocol for $\mathcal{R}_{\mathsf{res}}$, which is complete, HVZK and satisfies generalized special soundness for relation $\mathsf{R}'_{\mathsf{res}}$. Then, if $\mathsf{Translate}$ satisfies the four properties listed above, algorithms P_1, P_2 and V yield a Σ-protocol for $\mathcal{R}$, which is complete, HVZK and satisfies generalized special soundness for relation*
$$\mathsf{R}'((\mathsf{R}_{\mathsf{res}}, x', r'), c, s), (\mathsf{R}_{\mathsf{res}}, x', r'), \hat{c}, \hat{s})) = \mathsf{R}'_{\mathsf{res}}((r', c, s), (r, \hat{c}, \hat{s}))$$

This result permits identifying precisely the proof obligations that suffice to formally verify that the resulting protocol is correct and secure. First of all, one needs to show that the low-level protocol is itself correct and secure for the relation generator $\mathcal{R}_{\mathsf{res}}$ (in ZKCrypt this maps to the formal verification of subsequent compilation steps, which we discuss later). Secondly, one needs to show that the $\mathsf{Translate}$ procedure has all properties described in the theorem.

Idemix *goal resolution.* To make things more concrete, let us go back to the goal resolution performed by ZKCrypt and recast the rewriting performed for a term of the form $w \geq b$ in the theoretical framework above (an upper bound on w can be treated analogously). This matches the resolution of the proof goal in our Idemix running example. From Section 2, and using $x = Z/R_1^{m_1}$, we can rewrite the relation corresponding to goal G as
$$\mathsf{R}(x, w) \ \doteq \ x = A^e S^v R_2^{m_2} \ \wedge \ m_2 \geq b \qquad (3)$$

```
Translate(R, x, w):
  Parse (A, S, R_2, Z) ← R
  Parse (e, m_2, v) ← w
  Find (u_1, u_2, u_3, u_4) s.t. m_2 − b = u_1^2 + u_2^2 + u_3^2 + u_4^2
  r_i ←$ [0..2^{|n|} 2^ℓ], for i = Δ, 1, 2, 3, 4
  α ← r_Δ − Σ_{i=1}^{4} u_i r_i
  T_i ← Z^{u_i} S^{r_i}, for i = 1, 2, 3, 4
  T_Δ ← T_1^{u_1} T_2^{u_2} T_3^{u_3} T_4^{u_4} S^α
  Y_1 ← T_Δ Z^b
  (g_1, g_2, g_3, g_4) ← (T_1, T_2, T_3, T_4)
  x' ← (x, Y_1, T_1, T_2, T_3, T_4, T_Δ)
  w' ← (e, m_2, v, r_Δ, u_1, u_2, u_3, u_4, r_1, r_2, r_3, r_4, α)
  return (R_res, x', w')
```

Figure 3: Translate algorithm for Idemix example. Relations R and $\mathsf{R}_{\mathsf{res}}$ are as in (3) and (4).

Here, we have $w = (e, m_2, v)$. Similarly, using $Y_1 = T_\Delta Z^b$, the relation associated with resolved goal $\mathsf{G}_{\mathsf{res}}$ is:
$$\mathsf{R}_{\mathsf{res}}(x', w') \ \doteq \ x = A^e S^v R_2^{m_2} \ \wedge Y_1 = Z^{m_2} S^{r_\Delta} \ \wedge \bigwedge_{i=1}^{4} T_i = Z^{u_i} S^{r_i} \ \wedge T_\Delta = g_1^{u_1} g_2^{u_2} g_3^{u_3} g_4^{u_4} S^\alpha \qquad (4)$$

Here,
$$w' = (e, m_2, v, r_\Delta, u_1, u_2, u_3, u_4, r_1, r_2, r_3, r_4, \alpha)$$
$$x' = (x, Y_1, T_1, T_2, T_3, T_4, T_\Delta)$$

We observe that implicit in the definition of these goals are the relation generators $\mathcal{R}$ and $\mathcal{R}_{\mathsf{res}}$ that produce descriptions of the (hidden order) groups and generators that are used in the protocol. Furthermore, note that the resolved goal $\mathsf{G}_{\mathsf{res}}$ can be handled by the Σ^{GSP} protocol, for which ZKCrypt can generate an implementation of a ZK-PoK protocol which is proven to display the relevant security properties.

Figures 3 and 4 provide the pseudo-code of the Translate and Recover algorithms for the Idemix example. Observe the dual role of the T_i values in $\mathsf{G}_{\mathsf{res}}$: these values appear both as generators in $\mathsf{R}_{\mathsf{res}}$, and as images in x'. As we will see, this is essential to guarantee that witnesses for the original goal G can be recovered. We will now discuss how we prove that these algorithms satisfy the hypotheses of Theorem 1, from which we can conclude that resolution is correct.

4.2 A Formal Verification Perspective

We illustrate our approach to verifying the goal translation step on the same running example from Idemix. We first show how one would use EasyCrypt [12] to prove, with minor user intervention, that the Translate algorithm from Figure 3 satisfies the completeness and soundness properties. Then we explain how this formal verification approach has been integrated into the compiler to obtain an automated solution. Finally, we discuss how the simulatability and public verifiability properties are handled with a once-and-for-all proof for supported goals.

***Completeness of* Translate.** The idea underlying the completeness property is the following. Assume a prover knows $w \geq b$. Then, by Lagrange's Four Square Theorem [41], she can find integers $u_1, \ldots, u_4$ such that $m - b = \sum_{i=1}^{4} u_i^2$. By choosing $r_\Delta, r_1, \ldots, r_4$ at random, and defining $\alpha =$

$$\begin{array}{l}\underline{\mathsf{Recover}(\mathsf{R}_{\mathsf{res}}, x', w'):}\\ (e, m_2, v, r_\Delta, u_1, u_2, u_3, u_4, r_1, r_2, r_3, r_4, \alpha) \leftarrow w' \\ w \leftarrow (e, m_2, v) \\ \text{return } w\end{array}$$

Figure 4: Recover **algorithm for** Idemix **example**

$r_\Delta - \sum_{i=1}^{4} u_i r_i$ she can now clearly perform the above proof. Formally verifying this property using EasyCrypt is achieved by proving that the following experiment always returns true for all R in the range of $\mathcal{R}$ and all pairs (x, w) such that $\mathsf{R}(x, w)$ holds:

$$(\mathsf{R}_{\mathsf{res}}, x', w') \xleftarrow{\$} \mathsf{Translate}(\mathsf{R}, x, w);\ \text{return } \mathsf{R}_{\mathsf{res}}(x', w')$$

The Translate algorithm is represented in EasyCrypt in a form very close to its description in Figure 3, with the sole difference that the decomposition of $m_2 - b$ as a sum of four squares is computed by applying a function assumed to correctly implement Lagrange's decomposition. The proof itself is written as a series of game transitions, where the initial experiment is gradually transformed until it is reduced to the trivial program that simply returns true. All transitions are proved automatically by the tool.

***Soundness of* Translate.** A detailed specification of the Recover algorithm for the Idemix example is given in Figure 4. Let $m_2' \doteq \sum_{i=1}^{4} u_i^2 + b$. Clearly $m_2' \geq b$ and, by the definition of $\mathsf{R}_{\mathsf{res}}$ associated with $\mathsf{G}_{\mathsf{res}}$, we have that $Z = A^e S^v R_1^{m_1} R_2^{m_2}$. The correctness of Recover thus hinges on the fact that $\mathcal{R}$ *computationally* guarantees $m_2 = m_2'$, down to the following assumption:

Definition 4 (Unique representation assumption). *Let $\mathcal{H}$ be as before, and let Z and S be generators of $\mathcal{H}$. If a PPT algorithm outputs (a, b), $(a', b') \in \mathbb{Z} \times \mathbb{Z}$ such that $Z^a S^b = Z^{a'} S^{b'}$ then, with overwhelming probability, we have that $a = a' \wedge b = b'$.*

Any witness w' given to Recover, satisfying $\mathsf{R}_{\mathsf{res}}(x', w')$, for publicly validated $\mathsf{R}_{\mathsf{res}}$ and x', can be expressed in the following form:

$$Z^{m_2 - b} S^{r_\Delta} = Z^{m_2' - b} S^{\alpha + \sum_{i=1}^{4} r_i u_i}$$

If $m_2 \neq m_2'$, the input witness would provide two alternative representations for the same value under generators Z and S, contradicting the unique representation assumption. Thus, necessarily with overwhelming probability $m_2 = m_2'$.

Formally, if the group parameters are generated in a particular way, the unique representation assumption holds for $\mathcal{H} = \mathbb{Z}_n^*$ if the factoring assumption holds for n [21]. The Idemix specification incorporates this method into its parameter generation procedure. We note that this computational assumption is used implicitly throughout relevant literature when dealing with such transformations. We believe that forcing such assumptions to be stated explicitly is one of the advantages of using mechanized support to validate security proofs for cryptographic protocols.

The correctness of this algorithm is again formally verified in EasyCrypt. The proof is more intricate than in the case of Translate, since we now must take into account the unique representation assumption. The proof is quantified for all relations R in the range of $\mathcal{R}$, and all x and w such that $\mathsf{R}(x, w)$ holds. Consistently with the definition of the soundness property, we begin by defining the following experiment in EasyCrypt:

$$\begin{array}{l}(\mathsf{R}_{\mathsf{res}}, x', w') \leftarrow \mathcal{A}(\mathsf{R}, x, w); \\ w^* \leftarrow \mathsf{Recover}(\mathsf{R}_{\mathsf{res}}, x', w'); \\ \text{if } \neg\mathsf{R}_{\mathsf{res}}(x', w') \ \vee\ \neg\mathsf{pubVerify}(\mathsf{R}, x, \mathsf{R}_{\mathsf{res}}, x') \text{ then return } \perp \\ \text{else return } \mathsf{R}(x, w^*)\end{array}$$

Here, an adversary (i.e., a malicious prover) $\mathcal{A}$ is given such an input (R, x, w), and outputs a tuple $(\mathsf{R}_{\mathsf{res}}, x', w')$. The Recover algorithm is then called to produce a high-level witness. The experiment output expresses that the event in which the adversary produces a valid tuple $(\mathsf{R}_{\mathsf{res}}, x', w')$ must imply that Recover succeeds in obtaining a valid high-level witness w^*. This is expressed as a disjunction where either the adversary fails to produce publicly verifiable $\mathsf{R}_{\mathsf{res}}$, x' and a witness w' such that $\mathsf{R}_{\mathsf{res}}(x', w')$ holds, or Recover must succeed. Public verifiability is captured by a predicate pubVerify imposing that $Y_1 = T_\Delta Z^b$, and $g_i = T_i$ for $i = 1, 2, 3, 4$.

The proof establishes that this experiment is identical to the trivial program that always returns true, except perhaps when both $\mathsf{R}_{\mathsf{res}}(x', w')$ and $\mathsf{pubVerif}(\mathsf{R}_{\mathsf{res}}, x')$ hold, but the witness w' satisfies the following Boolean test:

$$m_2 \neq u_1^2 + u_2^2 + u_3^2 + u_4^2 + b \ \vee \\ r_\Delta \neq r_1 u_1 + r_2 u_2 + r_3 u_3 + r_4 u_4 + \alpha$$

Intuitively, this *failure* condition can be triggered only if the adversary was able to recover a low level witness which contradicts the unique representation assumption: in the proof we show that the probability of failure is bounded by the probability that an adversary $\mathcal{B}$ finds two different representations for the same group element under generators Z and S. On the other hand, conditioning on the event that *failure* does not occur, and through a series of transformations involving algebraic manipulations, we show that Recover always succeeds. Again, the validity of all transformations is handled automatically by EasyCrypt.

Integration and automation The above approach to formally verifying the goal resolution procedure was not integrated in earlier versions of ZKCrypt. This meant that user intervention was required to deal with this compilation step, and hence the natural back-end to use was EasyCrypt. This was not a major limitation, as many practical applications are based on specifications that already include the result of the goal resolution stage (this is the case of Idemix). Nevertheless, the latest release of ZKCrypt has been extended with an extra component that deals with this compilation step without user intervention. The formal verification of proof goal resolution is now handled in two steps:

1. Given a high-level goal G, a certified goal resolution module implemented in Coq generates a description of a reference low-level goal $\mathsf{G}_{\mathsf{res}}^{\mathsf{ref}}$ and a description of

the translation and recovery procedures. A once-and-for-all proof in Coq then guarantees that translation is complete and sound with respect to the reference low-level goal. This is done using essentially the same formalization approach as described above, extended to handle the general case of Σ^{GSP}-type goals where each pre-image can be bounded with an arbitrary interval.

2. The resolved goal produced by the CACE compiler $\mathsf{G}_{\mathsf{res}}$ is then proven equivalent to $\mathsf{G}^{\mathsf{ref}}_{\mathsf{res}}$. This establishes that the goal resolution step carried out by the CACE compiler is indeed correct. We recall that the optimized implementation produced by the CACE compiler is generated from $\mathsf{G}_{\mathsf{res}}$ and that this compilation process is formally verified in ZKCrypt, as described in the following Sections. This means that ZKCrypt provides guarantees of correctness and security of the generated optimized protocol with respect to the original high-level goal.

Public verifiability and simulatability. We now briefly discuss why the resolution procedure used in Idemix can be easily shown to satisfy public verifiability and simulatability. This argument extends to all instances of the resolution step implemented in ZKCrypt.

Looking at (4), one can immediately see that the public outputs of Translate can be validated to be in the correct range assuming that group membership can be efficiently checked, and given the fixed structure of the low-level relation. For simulatability we observe that the description $(\mathsf{R}_{\mathsf{res}}, x')$ output by Translate comprises the values of the images and $x' = (x, Y_1, T_1, T_2, T_3, T_4, T_\Delta)$ and of the generators $(g_1, g_2, g_3, g_4) = (T_1, T_2, T_3, T_4)$ Since x and Y_1 are fully determined by public inputs and $(T_1, T_2, T_3, T_4, T_\Delta)$, all that remains to show is that the latter values can be efficiently simulated. Observing that the domain of r_i is sufficiently large for the distribution of S^{r_i} $(i \in \{1, 2, 3, 4, \Delta\})$ to be statistically close to uniform in $\langle S \rangle$, we conclude that the variables T_i $(i \in \{1, 2, 3, 4, \Delta\})$ are also statistically close to uniform (note that Z and S are both generators of the same group of hidden order). It follows that these values can be trivially simulated by sampling uniformly random elements in the target group.

5. VERIFIED COMPILATION

At the core of the formal verification tool of ZKCrypt sits a verified compiler that generates correct and secure reference implementations of ZK-PoK for the following class of resolved proof goals produced by the front end of the compiler:

(i) Atomic goals consisting of pre-image proofs under a homomorphism with a finite domain using the Σ^{ϕ}-protocol, including product homomorphisms where the co-domain is a tuple of images in the range of the group operation.

(ii) Atomic goals consisting of pre-image proofs under an exponentiation homomorphism in a hidden-order group using the Σ^{GSP}-protocol, including product homomorphisms where the co-domain is a tuple of images in the range of the group operation. In particular, these include goals resulting from the resolution of interval proofs as described in Section 4.

(iii) Arbitrary, possibly nested, AND- and OR-compositions of proof goals as in point (i).

We stress that, although the code generation component of ZKCrypt addresses an even broader set of proof goals, this class was selected to cover essentially all practical applications.

Given a resolved proof goal $\mathsf{G}_{\mathsf{res}}$, the verified compiler is able to generate a description of a reference implementation for a suitable Σ-protocol, consisting of CertiCrypt programs corresponding to algorithms P_1, P_2, V and V_{c}. The generated descriptions of these algorithms follow a carefully designed structure, tailored to facilitate the formal proof that, for any goal, the (therefore) verified compiler produces correct and secure reference implementations.

The challenge here was to find the best balance between the level of abstraction at which the formalization is performed in CertiCrypt, and our goal to give formal verification guarantees over the optimized implementations generated by ZKCrypt. On the one hand, proving that the generated reference implementations meet the prescribed correctness and security requirements is much easier if one can reason abstractly about homomorphisms, group operations, etc. On the other hand, we wish to prove observational equivalence to programs produced by ZKCrypt in what is essentially pseudo-code of an imperative language very close to common programming languages. We achieve a compromise between these two aspects, which enables us to reach both objectives simultaneously.

Concretely, we construct each algorithm as shown in Figure 5. Note that all algorithms have at most two statements: a random assignment that samples all necessary random values up-front, and a deterministic assignment that computes the output in terms of the input of the algorithm and the sampled random values. For example, in algorithm $\mathsf{P}_1\langle \mathsf{G}_{\mathsf{res}} \rangle$, the first operation corresponds to sampling a tuple uniformly at random from the set $\mathsf{stType}\langle \mathsf{G}_{\mathsf{res}} \rangle$, which corresponds to a cartesian product of sets derived from the proof goal $\mathsf{G}_{\mathsf{res}}$. The second statement consists of a single assignment that evaluates a function of the inputs of the algorithm and the randomly sampled tuple; this is typically a huge expression performing all the necessary parsing and algebraic computations. More precisely, functions $\mathsf{Prover}_1\langle \mathsf{G}_{\mathsf{res}} \rangle$, $\mathsf{Prover}_2\langle \mathsf{G}_{\mathsf{res}} \rangle$ and $\mathsf{Verifier}\langle \mathsf{G}_{\mathsf{res}} \rangle$ may map to arbitrarily complex CertiCrypt expressions.

We observe that by restricting the reference implementation of protocols to the form shown in Figure 5 we do not lose generality. Indeed, this form is achievable for all goals, including those comprising arbitrary (possibly nested) Boolean compositions of atomic goals, which is a non-trivial aspect of the formalization approach adopted in ZKCrypt. Intuitively, for atomic goals, the computations performed by the reference implementation correspond to those described in Section 3.2

$\underline{\mathsf{P}_1\langle\mathsf{G}_{\mathsf{res}}\rangle(x,w)}$:
  $\mathsf{st}_{\mathcal{P}} \xleftarrow{\$} \mathsf{stType}\langle\mathsf{G}_{\mathsf{res}}\rangle$
  $r \leftarrow \mathsf{Prover}_1\langle\mathsf{G}_{\mathsf{res}}\rangle(x,w,\mathsf{st}_{\mathcal{P}})$
  return $(r,\mathsf{st}_{\mathcal{P}})$

$\underline{\mathsf{V}_{\mathsf{c}}\langle\mathsf{G}_{\mathsf{res}}\rangle()}$:
  $c \xleftarrow{\$} \mathsf{cType}\langle\mathsf{G}_{\mathsf{res}}\rangle$
  return c

$\underline{\mathsf{P}_2\langle\mathsf{G}_{\mathsf{res}}\rangle(x,w,c,\mathsf{st}_{\mathcal{P}})}$:
  $s \leftarrow \mathsf{Prover}_2\langle\mathsf{G}_{\mathsf{res}}\rangle(x,w,c,\mathsf{st}_{\mathcal{P}})$
  return s

$\underline{\mathsf{V}\langle\mathsf{G}_{\mathsf{res}}\rangle(x,r,c,s)}$:
  $a \leftarrow \mathsf{Verifier}\langle\mathsf{G}_{\mathsf{res}}\rangle(x,r,c,s)$
  return a

Figure 5: Descriptions of reference implementations.

for the Σ^{ϕ}- and Σ^{GSP}- protocols. For Boolean combinations of Σ^{ϕ}-protocols, the reference implementation is generated recursively by unfolding the inductively defined proof goal according to the standard procedures for Boolean composition described in Section 3.2. This is made possible by our approach to isolating random sampling operations from other computations. For illustrative purposes we present a short excerpt of the definition of the prover function Prover_1 in Listing 1. The excerpt corresponds to the case of Boolean compositions of proof goals that can be handled using the Σ^{ϕ}-protocol, and takes as input a pair (x,w) and the value $\mathsf{st}_{\mathcal{P}}$ comprising all values randomly sampled by the $\mathsf{P}_1\langle\mathsf{G}_{\mathsf{res}}\rangle$ algorithm. The base case maps to a concrete homomorphism, whereas recursive calls construct homomorphisms for And and Or combinations.

In addition to the descriptions of reference implementations for the algorithms of Σ-protocols, the compiler also generates the auxiliary algorithms that are required to establish security. In particular, for each goal, the compiler generates definitions of a suitable simulator and special extractor that can be used in the theorem statements that capture the zero-knowledge and proof of knowledge properties. The ability to generate suitable simulators is also an essential part of generating ZK-PoK protocols for Or compositions of Σ^{ϕ}-protocols. Indeed, the definitions of algorithms P_1 and P_2 explicitly rely on the simulator descriptions as part of their code, as can be seen in the snippet in Listing 1: in Or-compositions, the prover uses as a sub-procedure the simulator of the protocol for which it does not know a witness.

Listing 1: Definition of algorithm Prover_1 in Coq.

```
Fixpoint prover_phi g :
 expr (DomType g) → expr (CodomType g) →
 expr (TGtype (RandTG g)) → expr (CodomType g) :=
 match g with
 | Hom (PhiHom A B h) ⇒ fun w x ⇒ phiHom h
 | And g1 g2 ⇒ fun w x ps ⇒
   ( prover_phi (Fst w) (Fst x) (Fst ps) |
     prover_phi (Snd w) (Snd x) (Snd ps) )
 | Or g1 g2 ⇒ fun w x ps ⇒
   IF IsL(w) THEN
   ( prover_phi (ProjL w) (Fst x) (Fst (Snd ps)) |
     sim_phi (Snd x) (Fst ps) (Snd (Snd ps)) )
   ELSE
   ( sim_phi (Fst x) (Fst ps) (Fst (Snd ps)) |
     prover_phi (Projr w) (Snd x) (Snd (Snd ps)) )
 end
```

We discuss next how we prove the correctness and security properties of reference implementations for all supported proof goals.

Completeness. Completeness of reference implementations is given by the CertiCrypt theorem below.

Theorem 2 (Completeness). *For all supported goals $\mathsf{G}_{\mathsf{res}}$, and all pairs (x,w) satisfying the associated relation, we prove*

$$\begin{array}{l} c \xleftarrow{\$} \mathsf{V}_{\mathsf{c}}\langle\mathsf{G}_{\mathsf{res}}\rangle(); \\ (r,\mathsf{st}_{\mathcal{P}}) \xleftarrow{\$} \mathsf{P}_1\langle\mathsf{G}_{\mathsf{res}}\rangle(x,w); \\ s \leftarrow \mathsf{P}_2\langle\mathsf{G}_{\mathsf{res}}\rangle(x,w,c,\mathsf{st}_{\mathcal{P}}); \\ a \leftarrow \mathsf{V}\langle\mathsf{G}_{\mathsf{res}}\rangle(x,r,c,s) \end{array} \approx_0^{\{a\}} \quad a \leftarrow \mathrm{true}$$

Intuitively, this formalization states that in an honest execution, the verifier always will accept. Observe that, also in the protocol definition, the challenge generation is hoisted to the beginning of the protocol, as this facilitates proving equivalence claims. This is a valid transformation because we only have to prove that properties hold for an honest verifier that does not deviate from the protocol.

The proof of this theorem requires combined reasoning about the algebraic manipulations performed by the protocol parties. This is particularly challenging in the case of goals based on Σ^{ϕ}, for which the proof is by induction on the structure of the goal, dealing with the recursive definitions of the algorithms themselves. For example, in the case of Or-compositions, one needs to deal with the rearrangement of recursive invocations, by establishing intermediate results of the form:

$$\begin{array}{r} (r,c,s,a) \xleftarrow{\$} \mathsf{Protocol}\langle\mathsf{G}_1 \vee \mathsf{G}_2\rangle(x,\iota_1(w_1)) \\ \approx_0^{\{r,c,s,a\}} \\ (r_1,c_1,s_1,a_1) \xleftarrow{\$} \mathsf{Protocol}\langle\mathsf{G}_1\rangle(\pi_1(x),w_1); \\ c \xleftarrow{\$} \mathsf{cType}\langle\mathsf{G}_{\mathsf{res}}\rangle;\ c_2 \leftarrow c - c_1; \\ (r_2,c_2,s_2,a_2) \xleftarrow{\$} \mathsf{S}\langle\mathsf{G}_2\rangle(\pi_2(x),c_2); \\ r \leftarrow (r_1,r_2);\ a \leftarrow a_1 \wedge a_2 \end{array}$$

This result essentially states that the behavior of the protocol, when run on the prover side with the witnesses corresponding to goal G_1 is identical to that of another procedure which explicitly relies on a protocol for goal G_1 and a simulator for goal G_2. The proof of this equivalence must then make use of the recursive definition of the protocol (itself based on the prover and verifier algorithms presented in Figure 5) and of the simulator, and requires proving that the needed code rearrangements do not modify the semantics of the experiments.

5.1 Honest Verifier Zero Knowledge Property

Honest verifier ZK of reference implementations is given by the CertiCrypt theorem below.

Theorem 3 (HVZK). *For all supported goals $\mathsf{G}_{\mathsf{res}}$, and for all pairs (x,w) satisfying the relation associated with $\mathsf{G}_{\mathsf{res}}$, we prove the following statistical equivalence:*

$$\begin{array}{l} c \xleftarrow{\$} \mathsf{V}_{\mathsf{c}}\langle\mathsf{G}_{\mathsf{res}}\rangle(); \\ (r,\mathsf{st}_{\mathcal{P}}) \xleftarrow{\$} \mathsf{P}_1\langle\mathsf{G}_{\mathsf{res}}\rangle(x,w); \\ s \leftarrow \mathsf{P}_2\langle\mathsf{G}_{\mathsf{res}}\rangle(x,w,c,\mathsf{st}_{\mathcal{P}}); \\ a \leftarrow \mathsf{V}\langle\mathsf{G}_{\mathsf{res}}\rangle(x,r,c,s) \end{array} \approx_{\epsilon\langle\mathsf{G}_{\mathsf{res}}\rangle}^{\{r,c,s\}} \quad (r,c,s) \xleftarrow{\$} \mathsf{S}\langle\mathsf{G}_{\mathsf{res}}\rangle(x)$$

Here, $\mathsf{S}\langle\mathsf{G}_{\mathsf{res}}\rangle$ is the simulator algorithm generated by ZKCrypt for goal $\mathsf{G}_{\mathsf{res}}$. The concrete value of the statistical distance between the distributions depends on the goal. For the particular case of Σ^{ϕ}-protocols and Boolean combinations thereof, this is actually 0, and so proving this property corresponds to showing that the distributions are identical, implying perfect HVZK. In this case, the type of reasoning required to construct the proof is very similar to that described for completeness.

On the contrary, proving the zero knowledge property of Σ^{GSP}-protocols constitutes a significant challenge because it requires reasoning about statistical distance. Given any Σ^{GSP} goal $\mathsf{G}_{\mathsf{res}}$ defined over a homomorphism where the codomain is a tuple of arbitrary size m, we bound the statistical distance in the statement above by $\epsilon\langle\mathsf{G}_{\mathsf{res}}\rangle = m/2^{\ell}$, where ℓ is a concrete security parameter given as input to the compiler along with the goal specification (see Section 3.2). Establishing this result for arbitrary homomorphisms required reasoning about the number of points contained in hypercubes in $\mathbb{Z}^m$, and proving the upper bound using Bernoulli's inequality.

5.2 Proof of Knowledge

The following CertiCrypt theorem ensures that all generated reference implementations satisfy the Generalized Special Soundness introduced in Section 3.2.

Theorem 4 (Proof of Knowledge). *For every supported* valid *goals* $\mathsf{G}_{\mathsf{res}}$*, for all* (x, w) *satisfying the relation associated with* $\mathsf{G}_{\mathsf{res}}$*, and for any two accepting conversations* (r, c, s) *and* (r, c', s') *satisfying relation* $\mathsf{R}'\langle\mathsf{G}_{\mathsf{res}}\rangle$*,*

$$\mathsf{R}\langle\mathsf{G}_{\mathsf{res}}\rangle(x, \mathsf{E}\langle\mathsf{G}_{\mathsf{res}}\rangle(r, c, c', s, s')) = \text{true}\,.$$

The theorem statement nicely matches Definition 2, where relation $\mathsf{R}'\langle\mathsf{G}_{\mathsf{res}}\rangle$ expresses the restriction on traces described in Section 3.2 for either Σ^{ϕ}- or Σ^{GSP}- protocols. However, the theorem includes an additional *validity* restriction on proof goals that we now explain. Referring to Section 3.2, recall that for $c^+ > 2$ both the Σ^{ϕ}-protocol and the Σ^{GSP}-protocol can only be proven to satisfy Definition 2 if the underlying homomorphisms satisfy an additional property. Our notion of proof goal validity captures these extra restrictions. Concretely, the validity requirement for Σ^{ϕ} goals implies that all prime factors of special exponents for homomorphisms are greater than c^+. For the Σ^{GSP}-protocol, the validity requirement is as follows. Recall, from Section 3.2 that $\mathcal{R}$ typically samples an RSA modulus n and defines a relation R as

$$\mathsf{R}(x, (\mu, w)) \doteq x = \mu\phi(w) \;\wedge\; \mu^d = 1 \pmod{n}$$

Here, d is the product of the primes dividing the order of the multiplicative group modulo n, which are less than or equal to c^+. We require for validity that d satisfies this property.

Proving this theorem in CertiCrypt posed a different sort of challenge when compared to the previous ones, as it is not formulated in the form of a program equivalence statement. Essentially, it translates into a proof goal formulated over the semantics of the underlying algebraic constructions. Here we make critical use of the extensive Coq library that is included in ZKCrypt and that was developed to support the semantics of the data types included in the necessary CertiCrypt extensions. In turn, this library makes intensive use of SSReflect [38] and its comprehensive Coq library on algebraic and number theoretic results.

6. IMPLEMENTATION

To establish our ultimate verification goal, we translate the optimized implementations of protocols generated by ZKCrypt to the language of CertiCrypt. By taking advantage of the convenient notation that ZKCrypt automatically sets up in CertiCrypt, this translation step is straightforward and essentially corresponds to pretty-printing the output implementation files. Our strategy to formally verify these optimized implementations is to first establish an intermediate result stating that these are correct with respect to a reference implementation. More precisely, we establish that each of the algorithms in the implementation file, namely P_1, P_2, V, and V_{c}, are observationally equivalent to the corresponding algorithms in the reference implementation. These results are formalized in CertiCrypt by lemmas that typically look as the one below.

Lemma 1 (Correctness of P_1). *For all* (x, w) *in the domain of relation* R*, associated with resolved goal* $\mathsf{G}_{\mathsf{res}}$*, the following equivalence holds:*

$$(r, \mathsf{st}_{\mathcal{P}}) \xleftarrow{\$} \mathsf{P}_1(x, w) \approx_0^{\{r, \mathsf{st}_{\mathcal{P}}\}} (r, \mathsf{st}_{\mathcal{P}}) \xleftarrow{\$} \mathsf{P}_1^{\mathsf{ref}}\langle\mathsf{G}_{\mathsf{res}}\rangle(x, w)$$

Here $\mathsf{P}_1^{\mathsf{ref}}$ refers to the reference implementation for algorithm P_1. Equivalence is formalized by imposing that, for any possible fixed input, the outputs of both algorithms are identically distributed. Several aspects make proving these lemmas non trivial:

1. The reference implementation is expressed at a slightly higher level of abstraction than the optimized implementation. In particular, the reference implementation expresses homomorphism computations as native operations in the CertiCrypt language, whereas these are expanded as lower-level operations over the underlying algebraic groups in the optimized protocol implementation.
2. The reference implementation typically uses different language constructions than the optimized protocol. In particular, the reference implementation uses a minimum number of statements and local variables, in exchange for more elaborate expressions. For example, expressions in the reference implementation pack program variables into product data types, and contain conditional expressions in order to eliminate the need for if-then-else statements.
3. The ZK-compiler implementation may rearrange algebraic expressions in order to enable the generation of optimized implementations by the lower-level code generators or back-ends.

These differences are clearly visible in Listing 2, where we show an example of an observational equivalence proof goal as it appears in CertiCrypt, extracted from the deniable authentication example we include in the next section. The reference and optimized implementation respectively sit at the bottom and top of the listing.

Listing 2: Equivalence proof goal in CertiCrypt.

```
EqObs {x,w} {r,st,x,w}
 [if IsL(w) then [
    r1 <$- Gs;
    t1 ← [g] [^H] ([Gto_nat] r1);
    t2 ← [h] [^H] ([Gto_nat] r1) ]
  else [
    c1 <$- cs; s1 <$- Gs;
    t1 ← [g] [^H] ([Gto_nat] s1) [/H]
         Fst (Fst _x) [^H] ([cto_nat] c1);
    t2 ← [h] [^H] ([Gto_nat] s1) [/H]
         Snd (Fst _x) [^H] ([cto_nat] c1) ];
  If !IsL(w) then [
    r2 <$- Gs; t3 ← [g] [^H] ([Gto_nat] r2) ]
  else [
    c2 <$- cs; s2 <$- Gs;
    t3 ← [g] [^H] ([Gto_nat] s2) [/H]
         Snd _x [^H] ([cto_nat] c2)    ];
  r ← ((t1 | t2) | t3);
  st ← IF IsL(w) THEN (c2|(r1|s2))
                 ELSE (c1|(s1|r2)); ]
 [st <$- E.Dprod cs (E.Dprod Gs Gs);
  r ← IF IsL(w) THEN
        ([phi] Fst (Snd st) |
         [psi] Snd (Snd st) [/H]
          Snd _x [^H] ([cto_nat] (Fst st)))
       ELSE
        ((Fst ([phi] Fst (Snd st)) [/H]
          Fst (Fst _x) [^H] ([cto_nat] (Fst st)) |
          Snd ([phi] Fst (Snd st)) [/H]
          Snd (Fst _x) [^H] ([cto_nat] (Fst st))) |
          [psi] Snd (Snd st) ) ]
```

Pleasingly, our automation approach performed well in handling such equivalence proofs, both for this example and for the ones described in Section 7. Specifically, we have found that tactics already implemented in CertiCrypt are ideally suited to reduce proof goals as the one in Listing 2 to lower-level verification conditions over the semantics of operators used to implement the algorithms. Thanks to this, the problem of automation becomes one of constructing Coq tactics that can solve these lower level goals. To do this, we combine the powerful decision procedures ring and omega built into Coq with customized tactics that handle patterns observed in a comprehensive set of practical examples.

Listing 3: Protocol correctness lemma in CertiCrypt.

```
Lemma protocol_correct: ∀ g E P1 V1 P2 V2,
 EqObs {vW g,vX g} {vT g,vPS g,vW g,vX g}
  (P1 ++
  [vPS g ←Fst (vP1Res g); vT g ← Snd (vP1Res g)])
  (Prover1 g) →
 EqObs {vT g,vPS g,vW g,vX g}
       {vVS g,vC,vT g,vPS g,vW g,vX g}
  (V1 ++
  [vVS g ← Fst (vV1Res g); vC ← Snd (vV1Res g)])
  (verifier1 ++ [vVS g ← (vC | vT g)]) →
 EqObs {vVS g,vC,vT g,vPS g,vW g,vX g}
       {vS g,vVS g,vC,vT g,vW g,vX g})
  P2 (Prover2 g)  →
 EqObs {vS g,vVS g,vC,vT g,vW g,vX g}
       {vA,vS g,vC,vT g,vW g,vX g}
  V2 (Verifier2 g) →
 EqObs {vW g,vX g} {vA,vS g,vC,vT g,vW g,vX g}
  (P1 ++
   [vPS g ←Fst (vP1Res g); vT g ← Snd (vP1Res g)]
   ++ V1 ++
   [vVS g ← Fst (vV1Res g); vC ← Snd (vV1Res g)]
   ++ P2 ++ V2)
  (Verifier1 ++ Prover1 g ++ Prover2 g ++ Verifier2 g)
```

Combining the results. Once the equivalence lemmas above are established, generic proof scripts can be used to discharge the proof obligations associated with completeness, honest verifier, and generalized special soundness of optimized implementations. The theorem statements themselves are identical to those described in Section 5 for the reference implementations produced by ZKCrypt; however, their proofs are esentially different.

We rely on a general lemma stating that any given algorithms P_1, P_2, V, and V_c observationally equivalent to the respective algorithms in a reference implementation, lead to a protocol whose transcripts are distributed exactly as in the reference implementation. The statement of this lemma in CertiCrypt appears in Listing 3.

Proving the completeness and honest verifier zero knowledge properties of the optimized protocol then amounts to arguing that these results are directly implied by the identical distributions displayed by reference and optimized protocol implementations. For the soundness property, one appeals directly to the correctness of the optimized V_2 algorithm, which implies that an accepting trace for the optimized protocol is a valid input to the knowledge extractor that is proven to exist for the reference implementation.

7. MORE EXPERIMENTS AND RESULTS

Besides the running example presented in the previous sections, we also tested and verified the functionality of ZKCrypt based on a representative set of proof goals of academic and practical interest. We briefly report on some of these applications to illustrate the capabilities of ZKCrypt. We provide benchmarking results in Table 1 in terms of lines of code of the implementations output by the compiler and verification time of formal proofs. We note that the formal verification component of ZKCrypt described in this paper was developed in a way that is totally non-intrusive to the original CACE compiler that generates the executable implementations, and hence the efficiency of the generated C- and Java-implementations remains unaffected.

Electronic Cash. Electronic payment systems realize fully digital analogues of classical cash systems involving bills and coins. Besides high security and privacy guarantees, real-world usability requires that they work off-line, i.e., the bank must not be required to participate in transactions. One of the first schemes satisfying this condition was suggested by Brands [16]. All phases of his scheme use ZK-PoKs as sub-protocols. For instance, when withdrawing money from a bank account, a user has to prove its identity by proving possession of a secret key. The respective proof goal is given as follows:

$$\mathsf{ZPK}\Big[(u_1,u_2): I = g_1^{u_1} g_2^{u_2}\Big].$$

Here, $I, g_1, g_2 \in \mathbb{Z}_p^*$ such that $\operatorname{ord} g_1 = \operatorname{ord} g_2 = q$, where $q|(p-1)$ and $p, q \in \mathbb{P}$. The secrets u_1, u_2 are elements of $\mathbb{Z}_q$. This proof goal can be realized by a single instance of the Σ^ϕ-protocol(see Appendix A.1).

Deniable Authentication. Any Σ-protocol can be transformed into a non-interactive protocol using the Fiat-Shamir heuristic [32]. The idea is to substitute the verifier's first algorithm by a cryptographic hash function: Instead of relying on $\mathcal{V}$ to choose the challenge c uniformly at random,

	Type	Compositions	HLL (loc)	PIL (loc)	CertiCrypt (loc)	Verification
Electronic Cash	Σ^{ϕ}	None	23	59	1288	< 2m
Deniable Authentication	Σ^{ϕ}	And, Or	31	89	1383	< 3m
Ring Signatures	Σ^{ϕ}	Or	37	110	1384	< 4m
Identity Mixer	Σ^{GSP}	And	23	134	1515	< 25m

Table 1: Benchmarking results for representative applications of ZKCrypt. Type **and** Composition **describe the type and complexity of the protocol required to realize the proof goal.** HLL, PIL **and CertiCrypt denote the lines of code of the high-level input file, the generated protocol and the generated formal proof.** Verification **denotes the duration of generating the proofs and verifying them in CertiCrypt.**

the prover computes c itself as $c \leftarrow \mathsf{H}(r)$, where r is the commitment computed in its first step. It then computes its response s as in the original protocol. Upon receiving (r, c, s), the verifier checks whether the triple is an accepting conversation, and whether $c = \mathsf{H}(r)$.[3]

Clearly, proofs obtained in this way are not deniable. Namely, the verifier can convince a third party that it knows the prover's secret by just forwarding (r, c, s). This problem can be solved by migrating to *designated verifier ZK proofs*. There, one assumes a public key infrastructure (PKI), where each party deposits a public key. The prover then shows that it either knows the secret key for its own public key, or the secret key of the verifier. In this way the authentication scheme becomes deniable, as $\mathcal{V}$ could simulate proofs using its own secret key.

To make things concrete, we briefly recap the scheme of Wang and Song [55] here. A party A holds a secret key $x_{\mathsf{A}} \in \mathbb{Z}_q$, and publishes the corresponding public key $y_{\mathsf{A}} = (y_{1\mathsf{A}}, y_{2\mathsf{A}}) = (g^{x_{\mathsf{A}}}, h^{x_{\mathsf{A}}})$, where $q \in \mathbb{P}$ and g, h are elements of $\mathbb{Z}_p^*$ with order q. Now, authenticating $\mathcal{P}$ towards $\mathcal{V}$ boils down to the following proof goal:

$$\mathsf{ZPK}\Big[\,(x_{\mathcal{P}}, x_{\mathcal{V}}) : (y_{1\mathcal{P}} = g^{x_{\mathcal{P}}} \wedge y_{2\mathcal{P}} = h^{x_{\mathcal{P}}}) \vee y_{1\mathcal{V}} = g^{x_{\mathcal{V}}}\Big].$$

As the order q of g, h is known, this proof goal can be realized using the Σ^{ϕ}-protocol and the composition rules stated in Section 3.4 and Appendix A.3.

Ring Signatures. A ring signature scheme allows a set of parties to sign documents on behalf of the whole group [52], without revealing the identities of the signers. Such schemes are often realized by modifying the Fiat-Shamir transformation as follows: instead of setting $c \leftarrow \mathsf{H}(r)$, the prover sets $c \leftarrow \mathsf{H}(r, m)$, hashing the pair (m, r) where m is the message to be signed.

In a very basic scenario one wants to allow each member of the group to issue signatures on behalf of the group. Let therefore be given a PKI containing public keys $(y_{\mathsf{A}}, e_{\mathsf{A}}) \in \mathbb{Z}_{n_{\mathsf{A}}}^* \times \mathbb{Z}$ for safe RSA moduli n_{A}, and let each party A hold its secret key x_{A} satisfying $y_{\mathsf{A}} = x_{\mathsf{A}}^{e_{\mathsf{A}}}$. For simplicity, assume further that the group consists of only three parties. Then the proof goal is given by:

$$\mathsf{ZPK}\Big[\,(x_1, x_2, x_3) : y_1 = x_1^{e_1} \;\vee\; y_2 = x_2^{e_2} \;\vee\; y_3 = x_3^{e_3}\Big].$$

[3]Currently, the Fiat-Shamir heuristic is supported by the code-generation component of ZKCrypt only, so the formal verification tool currently only verifies the original Σ-protocol.

Again, as the domain of each mapping $x \mapsto x^{e_i}$ is finite, realization is done using the Σ^{ϕ}-protocol and the techniques stated in Section 3.4 and Appendix A.3.

Summary. Our experimental results illustrate that ZKCrypt is flexible enough to generate and verify implementations for a large set of proof goals occurring in practically relevant applications. We observe that, although proof verification is performed automatically, the performance of the developed Coq/Certicrypt tactics degrades significantly for proof goals based on the Σ^{GSP}-protocol. This is due to the complexity of the formalization of the underlying algebraic structures, which involve the definition of product homomorphisms with a large number of inputs and outputs.

8. RELATED WORK

Cryptographic compilers for ZK-PoK were studied before in two different lines of work; in the setting of the CACE project [18, 6, 7, 2] and for e-cash applications [45].

The CACE compiler is a certifying compiler that generates efficient implementations of zero-knowledge protocols. The compiler takes moderately abstract specifications of proof goals as input and generates C or Java implementations. The core compilation steps (i.e., all but the backends) are certifying in the sense that they generate an Isabelle [48] proof of the existence of a knowledge extractor guaranteeing special soundness. However, neither the fundamental zero-knowledge property nor completeness are addressed by the compiler, and the verification component only supports a very limited set of proof goals, not including the Σ^{GSP}-protocol. ZKCrypt builds on the compilation functionality of the CACE compiler, adding a new front-end and a completely reengineered verification component. Moreover, it solves several minor bugs, some of which were uncovered as a direct consequence of the new formal verification back-end development.

The ZKPDL compiler generates efficient distributed implementations of ZK-PoKs from high-level goals [45]. It has been used to build a realistic e-cash library. ZKPDL offers a level of abstraction similar to ours, but foregoes any attempt to verify the generated code and supports a more restricted set of proof goals.

Besides tools for ZK-PoK, there exists a large variety of other domain specific compilers, including Fairplay [44] and VIFF [30] for generating implementations of secure two-party computations. Also, generic cryptographic compilers offering differently abstract input languages have been proposed, e.g., [4, 54, 9, 40]. However, none of these tools supports formal verification.

A number of works have considered applications of formal verification to zero-knowledge proofs. Barthe et al. [13] use CertiCrypt to prove soundness, completeness, and zero-knowledge of Σ^{ϕ}-protocols and simple And/Or-compositions thereof. Although these results were constructed by hand and needed to be extended for a wider range of proof goals and arbitrary Boolean compositions, they are at the genesis of the formal verification infrastructure of ZKCrypt. Backes et al. [3] propose a method for checking that zero-knowledge proofs are adequately used, and apply their method to the DAA protocol.

9. CONCLUSIONS

ZKCrypt is an experimental high-assurance zero-knowledge compiler that applies to the realm of cryptography state-of-the-art approaches in verified and verifying compilation. It achieves an unprecedented level of confidence among cryptographic compilers. The verification infrastructure of ZKCrypt is based on the CertiCrypt platform, and relies on a set of carefully isolated concepts, including a new unified approach to special soundness and a novel formal treatment of goal resolution as a compilation step. We demonstrated that the compiler and the verification component are able to handle a large number of applications using ZK-PoKs.

There are plenty of avenues for future research in the field of cryptographic compilation and verification in general, and for the class of ZK-PoKs in particular. One future task is to verify the last stage of the compiler chain, code generation, to cover the entire compilation process. An interesting question is how far verified compilation can be extended beyond ZK-PoKs.

10. REFERENCES

[1] ABC4Trust EU Project. Official Website. `https://abc4trust.eu/`, 2011.

[2] Almeida, J. B., Bangerter, E., Barbosa, M., Krenn, S., Sadeghi, A.-R., and Schneider, T. A Certifying Compiler for Zero-Knowledge Proofs of Knowledge Based on Σ-Protocols. In *ESORICS '10* (2010), vol. 6345 of *LNCS*, Springer.

[3] Backes, M., Hritcu, C., and Maffei, M. Type-Checking Zero-Knowledge. In *ACM CCS 08* (2008), ACM, pp. 357–370.

[4] Bain, A., Mitchell, J. C., Sharma, R., Stefan, D., and Zimmerman, J. A Domain-Specific Language for Computing on Encrypted Data (Invited Talk). In *FSTTCS 2011* (2011), vol. 13 of *LIPIcs*, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, pp. 6–24.

[5] Bangerter, E. *Efficient Zero-Knowledge Proofs of Knowledge for Homomorphisms.* PhD thesis, Ruhr-University Bochum, 2005.

[6] Bangerter, E., Barzan, S., Krenn, S., Sadeghi, A.-R., Schneider, T., and Tsay, J.-K. Bringing zero-knowledge proofs of knowledge to practice. In *SPW 09* (2009).

[7] Bangerter, E., Briner, T., Heneka, W., Krenn, S., Sadeghi, A.-R., and Schneider, T. Automatic generation of Σ-protocols. In *EuroPKI 09* (2009).

[8] Bangerter, E., Camenisch, J., and Maurer, U. Efficient proofs of knowledge of discrete logarithms and representations in groups with hidden order. In *PKC 05* (2005), vol. 3386 of *LNCS*, Springer, pp. 154–171.

[9] Bangerter, E., Krenn, S., Seifriz, M., and Ultes-Nitsche, U. cPLC - A Cryptographic Programming Language and Compiler. In *ISSA 2011* (2011), IEEE Computer Society.

[10] Barthe, G., Grégoire, B., and Béguelin, S. Formal certification of code-based cryptographic proofs. In *POPL 09* (2009), pp. 90–101.

[11] Barthe, G., Grégoire, B., Heraud, S., Olmedo, F., and Zanella Béguelin, S. Verified indifferentiable hashing into elliptic curves. In *POST 2012* (Heidelberg, 2012), LNCS, Springer.

[12] Barthe, G., Grégoire, B., Heraud, S., and Zanella Béguelin, S. Computer-aided security proofs for the working cryptographer. In *CRYPTO 2011* (Heidelberg, 2011), vol. 6841 of *LNCS*, Springer, pp. 71–90.

[13] Barthe, G., Hedin, D., Zanella Béguelin, S., Grégoire, B., and Heraud, S. A machine-checked formalization of Σ-protocols. In *CSF 2010* (2010), IEEE.

[14] Barthe, G., Köpf, B., Olmedo, F., and Zanella Béguelin, S. Probabilistic reasoning for differential privacy. In *POPL 2012* (2012), ACM.

[15] Borisov, N., Goldberg, I., and Brewer, E. Off-the-Record Communication, or, why not to use PGP. In *WPES 2004* (2004), ACM, pp. 77–84.

[16] Brands, S. An Efficient Off-line Electronic Cash System Based on the Representation Problem. Tech. Rep. CS-R9323, CWI, 1993.

[17] Brickell, E. F., Camenisch, J., and Chen, L. Direct Anonymous Attestation. In *ACM CCS 04* (2004), ACM Press, pp. 132–145.

[18] Briner, T. Compiler for zero-knowledge proof-of-knowledge protocols. Master's thesis, ETH Zurich, 2004.

[19] Camenisch, J., and Herreweghen, E. V. Design and Implementation of the idemix Anonymous Credential System. In *ACM CCS 02* (2002), ACM Press, pp. 21–30.

[20] Camenisch, J., and Lysyanskaya, A. A Signature Scheme with Efficient Protocols. In *SCN 02* (2002), vol. 2576 of *LNCS*, Springer, pp. 268–289.

[21] Camenisch, J., and Shoup, V. Practical Verifiable Encryption and Decryption of Discrete Logarithms. In *CRYPTO 03* (2003), vol. 2729 of *LNCS*, Springer, pp. 126–144.

[22] Camenisch, J., and Stadler, M. Efficient group signature schemes for large groups (extended abstract). In *CRYPTO 97* (1997), vol. 1294 of *LNCS*, Springer, pp. 410–424.

[23] Camenisch *et al.*, J. Specification of the Identity Mixer Cryptographic Library (Version 2.3.0). Research Report RZ 3730 (#99740), IBM Research, 2010.

[24] Chaum, D. Security without identification: Transaction systems to make big brother obsolete. *Commun. ACM 28*, 10 (1985), 1030–1044.

[25] Chaum, D., and Evertse, J.-H. A secure and

privacy-protecting protocol for transmitting personal information between organizations. In *CRYPTO* (1986), vol. 263 of *LNCS*, Springer, pp. 118–167.

[26] Cramer, R. *Modular Design of Secure yet Practical Cryptographic Protocols*. PhD thesis, CWI and University of Amsterdam, 1997.

[27] Cramer, R., Damgård, I., and Schoenmakers, B. Proofs of partial knowledge and simplified design of witness hiding protocols. In *CRYPTO 94* (1994), vol. 839 of *LNCS*, Springer, pp. 174–187.

[28] Damgård, I. On Σ-protocols, 2004. Lecture on Cryptologic Protocol Theory; Faculty of Science, University of Aarhus.

[29] Damgård, I., and Fujisaki, E. A statistically-hiding integer commitment scheme based on groups with hidden order. In *ASIACRYPT 02* (2002), vol. 2501 of *LNCS*, Springer, pp. 77–85.

[30] Damgård, I., Geisler, M., Krøigaard, M., and Nielsen, J. B. Asynchronous Multiparty Computation: Theory and Implementation. In *PKC 09* (2009), vol. 5443 of *LNCS*, Springer, pp. 160–179.

[31] Damgård, I., Goldreich, O., Okamoto, T., and Wigderson, A. Honest Verifier vs Dishonest Verifier in Public Coin Zero-Knowledge Proofs. In *CRYPTO 95* (1995), vol. 963 of *LNCS*, Springer, pp. 325–338.

[32] Fiat, A., and Shamir, A. How to prove yourself: practical solutions to identification and signature problems. In *CRYPTO 86* (1987), vol. 263 of *LNCS*, Springer, pp. 186–194.

[33] Fujisaki, E., and Okamoto, T. Statistical zero knowledge protocols to prove modular polynomial relations. In *CRYPTO 97* (1997), vol. 1294 of *LNCS*, Springer, pp. 16–30.

[34] Goldberg, I., Ustaoglu, B., Gundy, M. V., and Chen, H. Multi-party off-the-record messaging. In *ACM CCS 09* (2009), ACM, pp. 358–368.

[35] Goldreich, O. Zero-knowledge twenty years after its invention. Tech. Rep. TR02-063, Electronic Colloquium on Computational Complexity, 2002.

[36] Goldreich, O., Micali, S., and Wigderson, A. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the ACM 38*, 1 (1991), 691–729. Preliminary version in 27th FOCS, 1986.

[37] Goldwasser, S., Micali, S., and Rackoff, C. The knowledge complexity of interactive proof-systems. In *STOC 85* (1985), ACM, pp. 291–304.

[38] Gonthier, G., Mahboubi, A., and Tassi, E. A Small Scale Reflection Extension for the Coq system. Rapport de recherche RR-6455, INRIA, 2008.

[39] Guillou, L., and Quisquater, J.-J. A "paradoxical" identity-based signature scheme resulting from zero-knowledge. In *CRYPTO 88* (1990), vol. 403 of *LNCS*, Springer, pp. 216–231.

[40] Kiyomoto, S., Ota, H., and Tanaka, T. A Security Protocol Compiler Generating C Source Codes. In *ISA 08* (2008), IEEE Computer Society, pp. 20–25.

[41] Lagrange, J. L. Œuvres, 1770.

[42] Leroy, X. Formal certification of a compiler back-end or: programming a compiler with a proof assistant. In *POPL 06* (2006), ACM Press, pp. 42–54.

[43] Lipmaa, H. On diophantine complexity and statistical zeroknowledge arguments. In *ASIACRYPT 03* (2003), vol. 2894 of *LNCS*, Springer, pp. 398–415.

[44] Malkhi, D., Nisan, N., Pinkas, B., and Sella, Y. Fairplay – Secure two-party computation system. In *USENIX Security Symposium* (2004), USENIX Association, pp. 287–302.

[45] Meiklejohn, S., Erway, C., Küpçü, A., Hinkle, T., and Lysyanskaya, A. ZKPDL: A Language-Based System for Efficient Zero-Knowledge Proofs and Electronic Cash. In *USENIX Security Symposium* (2010), USENIX Association, pp. 193–206.

[46] Microsoft. U-Prove. `http://www.microsoft.com/u-prove`, 2011.

[47] Necula, G. C., and Lee, P. The design and implementation of a certifying compiler. In *PLDI* (New York, NY, USA, 1998), vol. 33, PUB-ACM, pp. 333–344.

[48] Nipkow, T., and Paulson, L. Isabelle web site. `http://isabelle.in.tun.de`, 2010.

[49] Paillier, P. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT 99* (1999), vol. 1592 of *LNCS*, Springer, pp. 223–238.

[50] Rial, A., and Danezis, G. Privacy-preserving smart metering, 2011.

[51] Rivest, R., Shamir, A., and Adleman, L. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM 21*, 2 (1978), 120–126.

[52] Rivest, R., Shamir, A., and Tauman, Y. How to Leak a Secret - Theory and Applications of Ring Signatures. In *ASIACRYPT 01* (2001), vol. 2248 of *LNCS*, Springer, pp. 552–565.

[53] Schnorr, C. Efficient signature generation by smart cards. *Journal of Cryptology 4*, 3 (1991), 161–174.

[54] Schröpfer, A., Kerschbaum, F., Biswas, D., Geissinger, S., and Schütz, C. L1 - Faster Development and Benchmarking of Cryptographic Protocols. In *SPEED-CC 09* (2009).

[55] Wang, B., and Song, Z. A Non-Interactive Deniable Authentication Scheme Based on Designated Verifier Proofs. *Information Sciences 179*, 6 (2009), 858–865.

[56] Zuck, L. D., Pnueli, A., Goldberg, B., Barrett, C. W., Fang, Y., and Hu, Y. Translation and run-time validation of loop transformations. *Formal Methods in System Design 27*, 3 (2005), 335–360.

APPENDIX

A. CRYPTOGRAPHIC BACKGROUND

A.1 The Σ^ϕ-Protocol

The Σ^ϕ-protocol allows one to efficiently prove knowledge of preimages under homomorphisms $\phi : \mathcal{G} \to \mathcal{H}$ with a finite domain [53, 39]. For instance, it can be used to prove knowledge of the content of ciphertexts under the Pailler [49] or RSA [51] encryption schemes. Also, it can be used for all homomorphisms mapping into a group over elliptic curves. The protocol flow and the prover's and verifier's algorithms are depicted in Figure 6.

It is well known that the protocol is perfectly honest-verifier zero-knowledge for arbitrary ϕ and c^+, and that it satisfies

$\mathsf{P}(x, w)$		$\mathsf{V}(x)$
$k \xleftarrow{\$} \mathcal{G}$		
$r \leftarrow \phi(k)$	$\xrightarrow{r}$	
	$\xleftarrow{c}$	$c \xleftarrow{\$} \mathcal{C} = \{0, \ldots, c^+ - 1\}$
$s \leftarrow k + cw$	$\xrightarrow{s}$	$rx^c \stackrel{?}{=} \phi(s)$

Figure 6: Protocol flow of the Σ^ϕ-protocol.

the special soundness property whenever $c^+ = 2$. However, for many homomorphisms occurring in applied cryptography a much larger challenge set can be used, resulting in a much smaller cheating probability of the prover. Namely, this is the case if a preimage of a known and fixed power of x can be computed efficiently given only ϕ and x, i.e., if a pair $(u, v) \in \mathcal{G} \times \mathbb{Z} \setminus \{0\}$ satisfying $x^v = \phi(u)$ can be found in PPT (homomorphisms with this property are calld *special*, and (u, v) is a *pseudo preimage* of x under ϕ [5]). In this case, the special soundness property is known to hold for every c^+ smaller or equal to the smallest prime dividing v. For instance this condition is satisfied by homomorphisms mapping into a group $\mathcal{H}$ of known order q: there we have $x^q = \phi(0)$. Similarly, if $\phi(w_1, w_2) = w_1^e \psi(w_2)$, where $e \in \mathbb{Z}$ and ψ is a homomorphism, we have $x^e = \phi(x, 0)$.

For the Σ^ϕ-protocol, the relation generator $\mathcal{R}$ required for Definition 2 is given by the algorithm that always outputs the fixed relation $\mathsf{R} = \{(x, w) : x = \phi(w)\}$, and relation R' only requires that $c \neq c'$ in the accepting conversations. This corresponds to the classic notion of special soundness, in which no computational assumption is necessary for the success of the knowledge extractor.

A.2 The Σ^{exp}-Protocol

The efficient usage of the Σ^{GSP}-protocol depends on the given assumptions on the homomorphism ϕ. The Σ^{exp}-*protocol* can be used if those are not satisfied. Without going into details, the idea is to assume the availability of a common reference string containing a homomorphism ψ which satisfies the assumptions for the Σ^{GSP}-protocol. One then computes a commitment to all secret values under ψ, and runs instances of the Σ^{GSP}-protocol for ϕ and ψ in parallel. The generalized special soundness property of the overall protocol can then be inferred from that for ψ. For details we refer to Bangerter *et al.* [5, 8].

A.3 Boolean Compositions

Combining n Σ-protocols by a Boolean And, i.e., proving knowledge of witnesses for all protocols simultaneously, can easily be achieved by running the protocols in parallel, but letting the verifier only choose a single challenge c, which is then used for all protocols to combine.

Combining n predicates by a Boolean Or is slightly more involved. The prover is allowed to simulate all-but-one accepting conversations (for the predicate i^* it does not know the secrets for) by allowing it to choose the corresponding c_i. The remaining challenge c_{i^*} is then fixed such that $\sum_{i=1}^n c_i \equiv c \mod c^+$. To ensure this, the response is now given by $((s_1, c_1), \ldots, (s_n, c_n))$, where s_i is the response of the i^{th} predicate. In addition to running all verification algorithms, the verifier also checks that the c_i add up to the challenge c.

B. PROOF OF THEOREM 1

The three move form of the protocol follows directly from the specification of the algorithms for the high-level protocol. For the completeness property, one combines the completeness of the $\mathsf{Translate}$ algorithm with that of the underlying protocol for the resolved goal. Given that $\mathsf{Translate}$ always succeeds in producing a relation $\mathsf{R}_{\mathsf{res}}$ for which the low level protocol is complete and also a pair (x', w') satisfying this relation, this implies that an honest verifier will always accept on an interaction with an honest prover.

For the honest verifier zero-knowledge property, one needs to rely on both the completeness and the simulatability of $\mathsf{Translate}$. Simulatability ensures that $\mathsf{R}_{\mathsf{res}}$ and x' can be sampled statistically close to the output of $\mathsf{Translate}$, even without knowing the witness. Furthermore, from completeness, we know that $\mathsf{Translate}$ always produces a relation $\mathsf{R}_{\mathsf{res}}$ in the range of $\mathcal{R}_{\mathsf{res}}$, and a pair (x', w') satisfying this relation. On the other hand, the fact that the low level protocol is HVZK implies that, for all relations $\mathsf{R}_{\mathsf{res}}$ in the range of $\mathcal{R}_{\mathsf{res}}$, there exists a simulator that can generate a valid trace for this protocol from $\mathsf{R}_{\mathsf{res}}$ and x'. By combining the trace for the low-level protocol with the simulated $\mathsf{R}_{\mathsf{res}}$ and x', we get a valid simulated trace for the high level protocol.

Finally, the completeness and soundness conditions for $\mathsf{Translate}$, combined with the generalized special soundness for the low-level ZK-PoK protocol, imply that it is possible to construct a suitable generalized special extractor for the high-level protocol. Note that, in this case the prover is not assumed to be honest, which means that it might not be running the $\mathsf{Translate}$ algorithm correctly. Hence, for this to be possible, it is crucial that the verifier is able to use the public verifiability property, to check that the low level relation and public input are in the range of $\mathsf{Translate}$, and hence in the range of $\mathsf{R}_{\mathsf{res}}$. Observe also that the generalized special soundness for the high level protocol is actually defined with respect to a relation on traces that reflects the restrictions required for the low level protocol. In practice, this means that the proof of knowledge property of the high-level protocol will only be guaranteed if the computational hardness assumptions underlying $\mathcal{R}_{\mathsf{res}}$ can be publicly verified to hold in an instance $\mathsf{R}_{\mathsf{res}}$, derived from an honestly sampled high-level relation R. For our purposes, this means that the traces provided to the generalized special extractor we need to construct can be passed directly to the generalized special extractor for the low-level protocol, to recover a valid low-level witness w'. We now appeal to the soundness property of $\mathsf{Translate}$, which guarantees that this low level witness can be translated back into the required witness for the original goal G using the $\mathsf{Recover}$ protocol, with overwhelming probability.

C. INPUT AND OUTPUT FILES OF THE USE CASE

In the following we give the input files specifying the proof goal of our use case in Section 2, as well as the outputs of the goal resolution and the implementation phases, respectively. We omit giving outputs of the backends here, as these are essentially standard C- and Java-programs without further interest for this paper.

C.1 The Proof Goal G

Figure 7 shows the input (a .zk-file) for our running example. It can easily be obtained by instantiating the template $\mathsf{CL}(m_1, m_2)$ by the mapping underlying the CL-signature scheme [20], as is already done in the identity mixer specification as well. The rest of the file essentially only describes the algebraic setting, and the required security properties.

```
Declarations {
 Int(2048)  n;
 Zmod*(n)   z, R_1, R_2, A, S;
 Int(1000)  m_1, m_2, e, v, b;
}
Inputs {
 Public          := n, z, R_1, A, S, R_2, b;
 ProverPrivate  := e, m_2, v;
}
Properties {
 KnowledgeError      := 80;
 SZKParameter        := 80;
 ProtocolComposition := P_0;
}
SigmaGSP P_0 {
 Homomorphism(phi: Z^3 -> Zmod*(n):
              (e,m_2,v) |-> (A^e*S^v*R_2^m_2));
 ChallengeLength := 80;
 Relation(
   (z*R_1^(-m_1)) = phi(e,m_2,v) And m_2 >= b);
}
```

Figure 7: .zk-file specifying Idemix proof goal G.

The first two blocks, `Declarations` and `Inputs`, are almost self-explanatory. First, all variables used in the protocol are declared. The compiler natively supports several data types, including integers, additive and multiplicative residue groups as well as certain groups over elliptic curves. If other algebraic structures are required, users can specify abstract group types, and only have to supply implementations in the target language (e.g., Java) at compilation time. Then, the variables are passed as public or private inputs to the parties. Typically, variables will be declared as private if and only if knowledge of these values has to be proved.

The `Properties` block specifies the intended security properties and the proof goal of the protocol. In our example, the `KnowledgeError` of the generated protocol shall be at most 2^{-80}, and the statistical distance of simulated from real protocol runs must be at most $2^{-\mathtt{SZKParameter}}$. Inside ZKCrypt, the `KnowledgeError` parameter is translated onto a concrete challenge length, and `SZKParameter` gives the security parameter controlling the tightness of the HVZK property. Note that the specified values are consistent with the specifications of the identity mixer [23].

Finally, the proof goal only consists of a single predicate. This predicate is then specified in detail. It shall be proved using a `SigmaGSP`-protocol, and the relation to be proved is that of our running example. The maximum `ChallengeLength` that may safely be used for the homomorphism at hand is given by 80 (this cannot be computed from `phi` as it would require to compute the order of `Zmod*(n)`, which is infeasible; it must thus be specified by the user). We observe that, for concrete values of n, i.e., strong RSA moduli, this parameter implicitly gives the concrete value of d (the product of all primes smaller than c^+ dividing $\operatorname{ord}\mathcal{H}$) for which the proof of knowledge property will hold.

C.2 The Resolved Proof Goal $\mathsf{G}_{\mathsf{res}}$

As discussed in detail in Section 2 and Section 4, ZKCrypt resolves all interval claims in the proof goal G in a first step, resulting in a resolved proof goal $\mathsf{G}_{\mathsf{res}}$ only containing preimage claims under group homomorphisms. The output of this goal resolution phase is the .psl-file given in Figure 8. The

```
Declarations {
 Int(2048) n;
 Int(1000) b, m[1..2], e, v, r_Delta, r[1..4],
           u[1..4], alpha;
 Zmod*(n)  z, R[1..2], A, S, T_Delta, T[1..4],
           x_1 := z*R_1Ãă(-m_1), x_2 := T_Delta*z^b;
}
Inputs {
 Public        := n, b, m_1, z, R[1..2], A, S,
                  T_Delta, T[1..4];
 ProverPrivate := e, m_2, v, r_Delta, r[1..4],
                  u[1..4], alpha;
}
Properties {
 KnowledgeError      := 80;
 SZKParameter        := 80;
 ProtocolComposition := P_0;
}
SigmaGSP P_0 {
 Homomorphism(phi: Z^13 -> Zmod*(n)^7:
  (e,m_2,v,r_Delta,u_1,u_2,u_3,u_4,r_1,r_2,r_3,
   r_4,alpha) |->
  (A^e*S^v*R_2^m_2,z^m_2*S^r_Delta,z^u_1*S^r_1,
   z^u_2*S^r_2,z^u_3*S^r_3,z^u_4*S^r_4,
   T_1^u_1*T_2^u_2*T_3^u_3*T_4^u_4*S^alpha));
 ChallengeLength := 80;
 Relation((x_1,x_2,T_1,T_2,T_3,T_4,T_Delta) =
          phi(e,m_2,v,r_Delta,u_1,u_2,
              u_3,u_4,r_1,r_2,r_3,r_4,alpha));
}
```

Figure 8: .psl-file with resolved Idemix proof goal $\mathsf{G}_{\mathsf{res}}$.

syntax and semantics of the single blocks are identical to .zk-files, with the only exception that no interval claims are allowed any more in .psl-files. The assignments to `x_1` and `x_2` in the `Declarations`-block define relations that have to be satisfied by the inputs and which are later verified at runtime. It can be seen that up to minor syntactical changes the proof goal is exactly that specified in (3).

C.3 The Optimized Implementation $\mathsf{I}_{\mathsf{opt}}$

As explained in Section 2, the implementation phase transforms the resolved proof goal into an optimized implementation $\mathsf{I}_{\mathsf{opt}}$. This .pil-file can be thought of as some kind of pseudocode, which makes all algebraic operations and messages to be sent explicit.

On a high level, the .pil-file consists of two sets of algorithms for the prover and the verifier, respectively, which are then compiled into C- and Java-code by the respective backends, which essentially only need to perform syntactic rewriting. For illustration purposes, we show the verifier's algorithm of our running example in Figure 9. The `...`-blocks substitute complex algebraic expressions, which were removed due to space constraints. The semantics of the single commands is straightforward: `CheckMembership` checks whether a variable lies in a given interval or group. The `Verify` command checks whether or not the condition the specified algebraic relation is satisfied. If either of these checks fails, the verifier aborts and rejects the protocol run. It accepts, if and only if all checks were successful.

```
Def (Void): Round2([...,...] _s_1, ... , _s_13) {
  CheckMembership(_s_1, [...,...]);
  CheckMembership(_s_2, [...,...]);
  CheckMembership(_s_3, [...,...]);
  CheckMembership(_s_4, [...,...]);
  CheckMembership(_s_5, [...,...]);
  CheckMembership(_s_6, [...,...]);
  CheckMembership(_s_7, [...,...]);
  CheckMembership(_s_8, [...,...])
  CheckMembership(_s_9, [...,...]);
  CheckMembership(_s_10, [...,...]);
  CheckMembership(_s_11, [...,...]);
  CheckMembership(_s_12, [...,...]);
  CheckMembership(_s_13, [...,...]);
  Verify(x_1 == (z*(R_1^(-m_1));
  Verify(x_2 == (((T_Delta)*(z^b)));
  Verify((_t_1*(x_1^_c)) == ...);
  Verify((_t_2*(x_2^_c)) == ...);
  Verify((_t_3*(T_1^_c)) == ((z^_s_5)*(S^_s_9)));
  Verify((_t_4*(T_2^_c)) == ((z^_s_6)*(S^_s_10)));
  Verify((_t_5*(T_3^_c)) == ((z^_s_7)*(S^_s_11)));
  Verify((_t_6*(T_4^_c)) == ((z^_s_8)*(S^_s_12)));
  Verify((_t_7*(T_Delta^_c)) == T_1^_s_5* ...);
}
```

Figure 9: .pil-file with the Idemix implementation I_{opt}.

More generally, the implementation language supports a broad set of operations, including assignments, random choices in groups and intervals, `if-then-else` branches checking whether certain values are known to the prover, loops and native commands for certain further cryptographic schemes which are required to efficiently realize certain Boolean compositions of proof goals.