Automatic Generation of Sound Zero-Knowledge Protocols

Endre Bangerter[1], Jan Camenisch[2], Stephan Krenn[3], Ahmad-Reza Sadeghi[4], and Thomas Schneider[4⋆]

[1] Bern University of Applied Sciences, Biel-Bienne, Switzerland
endre.bangerter@bfh.ch
[2] IBM Research, Zurich Research Lab, Rüschlikon, Switzerland
jca@zurich.ibm.com
[3] Bern University of Applied Sciences, Biel-Bienne and University of Fribourg, Switzerland
stephan.krenn@bfh.ch
[4] Horst Görtz Institute for IT Security, Ruhr-University Bochum, Germany
{ahmad.sadeghi,thomas.schneider}@trust.rub.de

(Extended Poster Abstract)

Abstract. Efficient zero-knowledge proofs of knowledge (ZK-PoK) are basic building blocks of many practical cryptographic applications such as identification schemes, group signatures, and secure multiparty computation. Currently, first applications that critically rely on ZK-PoKs are being deployed in the real world. The most prominent example is Direct Anonymous Attestation (DAA), which was adopted by the Trusted Computing Group (TCG) and implemented as one of the functionalities of the cryptographic chip Trusted Platform Module (TPM).
Implementing systems using ZK-PoK turns out to be challenging, since ZK-PoK are, loosely speaking, significantly more complex than standard crypto primitives, such as encryption and signature schemes. As a result, implementation cycles of ZK-PoK are time-consuming and error-prone, in particular for developers with minor or no cryptographic skills.
In this paper we report on our ongoing and future research vision with the goal to bring ZK-PoK to practice by automatically generating sound ZK-PoK protocols and make them accessible to crypto and security engineers. To this end we are developing protocols and compilers that support and automate the design and generation of secure and efficient implementation of ZK-PoK protocols.

1 Introduction

A zero-knowledge proof of knowledge (ZK-PoK) is a two-party protocol between a prover and a verifier, which allows the prover to convince the verifier that he knows some secret values that satisfy a given relation (proof of knowledge property), without the verifier being able to learn anything about them (zero-knowledge property). There are fundamental results showing that all relations in NP have ZK-PoK [GMW91,DFK+93,PRS02,IKOS07]. The corresponding protocols are of theoretical relevance, but much too inefficient to be used in practical applications.

In contrast to these generic protocols, there are various protocols which are efficient enough to be used in real world applications. Essentially, all ZK-PoK protocols being used in practice today are based on so called Σ-protocols. What is typically being proved using basic Σ-protocols is the knowledge of a preimage under a homomorphism (e.g., a secret discrete logarithm). Yet, there are numerous considerably more complex variations of these preimage proofs. These ZK-PoK proof techniques play an important role in applied cryptography. In fact, many practically oriented applications use such proofs as basic building blocks. Examples include identification schemes [Sch91], interactive verifiable computation [CM99], group signatures [Cam98], secure watermark detection [ARS05], and efficient secure multiparty computation [LPS08] - just to name a few.

⋆ This work is supported by the EU under FP7 project CACE (Computer Aided Cryptography Engineering).

While many of these applications typically only exist on a specification level, a direction of applied research has produced first applications using ZK-PoKs that are deployed in the real world. The probably most prominent example is Direct Anonymous Attestation (DAA) [BCC04], which was adopted by the Trusted Computing Group (TCG), an industry consortium of many IT enterprises, as a privacy enhancing mechanism for remote authentication of computing platforms. Another example is the *identity mixer* anonymous credential system [CH02], which was released by IBM into the Eclipse Higgins project, an open source effort dedicated to developing software for "user-centric" identity management.

Up to now, design and implementation of ZK-PoK protocols is done "by hand". In fact, past experiences, e.g., during the design and implementation of the preceding two examples, have shown that this is a time consuming and error prone task. This has certainly to do with the fact that ZK-PoK protocols are relatively new primitives (e.g., there are no pre-built libraries and other tools supporting implementation) but also with the fact that ZK-PoK are considerably more complex than other existing crypto primitives (e.g., signatures, encryption, etc.) which are widely deployed.

The goal of our ongoing and future research is to bring ZK-PoK to practice by making them accessible to crypto and security engineers. To this end we are working on compilers and related tools that support and automate the design and implementation of ZK-PoK protocols. For instance the *High Level Compiler* which is part of our toolbox, will take as input a high-level specification of the goals of a ZK-PoK and automatically finds the specification of a corresponding protocol. The *Protocol Compiler* transforms this protocol specification into an executable implementation, e.g., Java or C code or documentation of the protocol in LaTeX. We have already developed and implemented a language and compiler that automates the latter step from protocol specification to code generation for proofs in known-order groups [BCK+08]. Extending this compiler to hidden-order groups and automatically finding a protocol from a high-level specification is subject of ongoing research.

In this extended poster abstract we describe the challenges pertaining to automatically generate sound ZK-PoK for practical use in Sec. 2 and give an overview of a solution blueprint and first results on solving these challenges in Sec. 3. Our approach for a consistent but efficient theoretical framework for ZK-PoK in known- and hidden-order groups is sketched in Sec. 4.

2 Challenges

In the following paragraphs we will describe the main challenges that ZK-PoK pose to crypto engineers and protocol designers, which we aim to tackle with our compiler suite.

Let us introduce some notation first. By the *semantic goal* of a ZK-PoK we refer to *what a prover wants to demonstrate in zero-knowledge*. For instance, the semantic goal can be to prove knowledge of a discrete logarithm of a group element with respect to another group element. A more complex goal is to prove that a given cipher-text encrypts a valid (with respect to some given public key) signature on a specific message. By a ZK-PoK *protocol (specification)* we refer to the actual description of a protocol (i.e., the operations of prover and verifier and the messages being exchanged). For instance, the well known Schnorr protocol [Sch91] realizes the first semantic goal mentioned above, and verifiable encryption protocols [Ate04] realize the latter. It is important to note that given a semantic goal, there can be many different protocols realizing that goal; also sometimes

one does not know how to construct an efficient protocol realizing a goal (which does not mean that there is no better protocol than using a generic protocols for NP statements [GMW91,DFK+93,PRS02,IKOS07]). Finally, by a *(protocol) implementation* we refer to actual code (e.g., in C or Java) realizing a specification.

Designing ZK-PoK. On a conceptual level ZK-PoK are easy to grasp and intuitive: formulating the semantic goal of a ZK-PoK is an easy task for a protocol designer. It essentially boils down to formulating the requirements on a ZK-PoK. Yet, finding a protocol specification realizing a semantic goal is in many cases difficult or impossible for people who don't have extensive expertise in the field. As a result, we believe that unlike other, more simple crypto primitives such as encryption, signatures, etc., ZK-PoK are not part of the toolbox of many crypto engineers. This in turn lets us conjecture that the potential of novel applications that can be built using ZK-PoK is only poorly exploited.

Why is it actually often hard to find ZK-PoK protocol meeting a semantic specification? The main problem is the lack of a unified, modular and easy to understand theoretical framework underlying the various ZK-PoK protocols and proof techniques. As a result there is no methodological formal way to guide cryptographic protocol designers. In fact, there is a large number of tricks and techniques "to prove this and that", yet combining various tricks and preserving the security properties (i.e., the ZK and PoK properties) is not straightforward and is non-modular. Composition of techniques often needs intricate knowledge of the technique at hand, and may also require modification of the technique. For instance some techniques only work under certain algebraic assumptions and preconditions. These can be conditions on the order of the algebraic group and group elements being used, conditions on whether the prover knows the factorization of a composite integer, distributions of protocol inputs etc.. The algebraic conditions in turn require tuning protocol parameters. As a result, finding and designing ZK-PoK protocols is a heuristic process based on experience and a detailed understanding of the techniques being used. In contrast, encryption and signature schemes are much more easily accessible to designers.

Efficiency of implementation process. The step going from a protocol specification to a protocol implementation is often considered to be trivial from a conceptual point of view. Yet in practice it is not. In fact, experiences made while implementing, e.g., a prototype of the identity mixer [CL01,CH02] protocols have shown that a manual implementation is tedious and error prone and easily takes person weeks. Moreover, often protocol specifications are written by cryptographers and the implementation is done by SW engineers. This "skill gap" may lead to implementation errors. The former often don't care sufficiently or don't have the skills to cope with implementation issues and their specifications may be slightly incomplete; the latter may have a hard time to assess implementation decisions, which depend on cryptographic subtleties. Additionally, minor changes in the semantic goal often result in fundamental changes of the resulting protocol.

Efficiency of code. Getting efficient code, in terms of computation time, memory usage, size of messages sent over the network, number of message exchanged, etc., can be of great concern when using ZK-PoK. The choice of the resource to optimize may greatly differ depending on the actual device on which the code is run. In the DAA protocol [BCC04] for example, parts of the prover's algorithm is executed by a relatively simple and low cost TPM chip while the verifier's algorithm runs on a powerful computer.

There are at least two places where one can optimize ZK-PoK. On a high-level, there is potential for optimization by finding the most efficient protocol specification realizing a given semantic goal (this type of optimization is closely related to "designing ZK-PoK" issue described above). On a lower level one can optimize the code implementing a given protocol, much like the optimization performed by compilers for conventional programming languages like C, Java etc, whereas one can specially concentrate on crypto operations. Optimization in general, requires substantial experience and an intricate understanding of the runtime environment.

3 Solution Blueprint and Results

In the following we sketch how we envision to resolve the challenges described above and describe first results. Fig. 1 shows the main components of our toolbox on a very high-level. One is the *High Level Compiler* that takes the semantic goal of ZK-PoK to be proven and finds the corresponding *Protocol Specification*. From this, the *Protocol Compiler* generates the *Protocol Implementation* e.g., as Java or C code realizing the semantic goal.

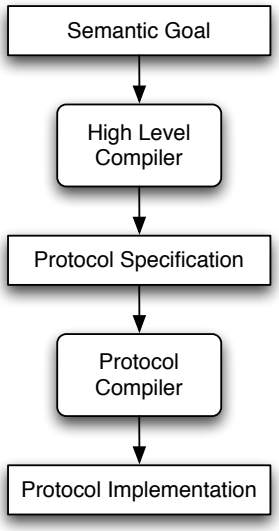

Fig. 1. Architecture of our framework for automatic generation of sound ZK-PoK protocols.

Designing ZK-PoK. At the moment we are designing a high-level language in which the semantic goal of a ZK-PoK together with its non-functional properties can be formulated in a user-friendly way. The language is inspired by the well-known Camenisch-Stadler notation [CS97] which is used to formulate the intended semantic goal. We enrich this with non-functional properties which allow to specify optimization constraints (e.g., upper bounds for computation- or communication complexity) and the security level (e.g., knowledge error, tightness of the statistical zero-knowledge property, etc.) of the protocol being generated. In this high-level language, we abstract away as many technical details as possible to ease design and usage of ZK-PoK for non-experts.

To be able to actually build a compiler for the semantics of that high-level language we are currently working on a unified theoretical framework for the various ZK-PoK

techniques. For this, we extend the existing theory for zero-knowledge proofs which by now mainly deals with known order groups [CDS94,Cra96,Bra97,BS02]. Our extended theoretical framework is capable to cope with arbitrary combinations of protocols in hidden order groups (e.g., RSA groups) as well [DF02,BCM05,CKY09]. To this end, we have conceived the new Σ^{exp} protocol [BCK+08], which yields efficient ZK-PoK in a more modular but still efficient manner than the existing protocols [DF02,BCM05,CKY09]. More details on this framework will be given in Sec. 4.

A first prototype [BCK+08] of our compiler and semantic language implements a subset of the envisaged compiler framework. It already supports the automatic generation of sound ZK-PoK protocols for known order groups which includes various cryptosystems such as Pedersen commitments/verifiable secret sharing [Ped92], Schnorr authentication/signatures [Sch91], electronic cash [Bra94,Oka95,CFT98], group signatures [CL04], and ring signatures [CDS94].

Efficiency of implementation process. In fact, this challenge is solved inherently by our compiler based approach (i.e., by automating the implementation process). Our first prototype of the compiler performs its work in less than a second, and we expect the full fledged version of the compiler to run in a couple of seconds. There are other features to improve the efficiency of the implementation process. For instance, one could give our toolchain a consistent user interface which is based on an integrated development environment such as Eclipse to assist application developers in all steps involved: specification of proof goals (e.g., syntax highlighting), documentation (e.g., LaTeX diagrams of the generated protocols), or performance tests of the generated code (e.g., code profiling).

Efficiency of code. As mentioned above, this challenge has to be dealt with on a high level (i.e., finding the most efficient protocol for a given semantic goal) and on a low level (i.e., by optimizing the code implementing a protocol). For the high-level optimization, the high level compiler will choose the most appropriate proof techniques according to the user's requirements on communicational and computational complexity. For example interval proofs can be done either with techniques described in [Bou00] or [Lip03] which have different communication and computation complexity. To support low level optimization we'll (additionally to C) also provide a compiler backend that outputs code in the CAO ("Cryptography Aware language and cOmpiler") language [BNPS05]. This is a language and a compiler geared towards the generation of an efficient and secure low-level implementation of cryptographic primitives; CAO is also being developed in the CACE project.

Results obtained so far. We have implemented a first prototype of the protocol compiler which is capable to generate Σ-protocols (which can easily be converted into ZK-PoK protocols) for groups with known order [Bri04,CRS05,BCK+08]. More precisely, we have designed a language which is inspired by the widely used Camenisch-Stadler notation [CS97]. ZK-PoK protocol specifications in this language are then translated by the compiler either into Java code or documentation in LaTeX. The Java code can be integrated into higher level systems that make use of the corresponding ZK-PoK. The LaTeX documentation can be used for documenting the protocols and also for verification purposes. To the best of our knowledge, this is the first compiler suite to support the automatic generation of sound ZK-PoK protocols. Currently, we are working on a compiler which supports also protocols in hidden-order groups [DF02,BCM05,BCK+08,CKY09] and generation of C or CAO code.

4 The Σ^{exp}-protocol - An unconditionally portable Σ-protocol in the auxiliary string model

The newly introduced framework of Camenisch, Kiayias and Yung [CKY09] spotlights several known problems when proving preimages of exponentiation homomorphisms (i.e. homomorphisms of the form $\phi(x_1, \ldots, x_n) = g_1^{x_1} \cdots g_n^{x_n}$), especially when such proofs are performed within groups of unknown order, where the *generalized Schnorr protocol* has to be applied. To overcome these problems, they introduce the notion of *unconditionally portable protocols* and give a generic protocol transformation from a given generalized Schnorr protocol (which is a Σ-protocol) into an unconditionally portable protocol. Both, the existing transformation of [BCM05] in the random oracle model and its adoption for the standard model [CKY09] need six moves.

However, most protocol transformations that need to be applied in order to use such honest-verifier ZK protocols in practice are defined for Σ-protocols, which by definition have three moves only. Hence they can not be applied immediately to those six-move protocols. Examples are the highly efficient and widely used Fiat-Shamir transformation for non-interactive ZK (NIZK) in ROM [FS87], or constructions for concurrent zero-knowledge (CZK) [Dam00,MP03,Vis06].

The Σ^{exp}-protocol [BCK+08], which was introduced independently from, and earlier than [CKY09], fits excellently into this framework and in fact is a three-move unconditionally portable protocol in the auxiliary string model without random oracles. Indeed, the Σ^{exp}-protocol can easily be derived from the six-move protocols of [BCM05] and [CKY09], by replacing the first, fifth and sixth move of these protocols with an auxiliary string holding the safeguard group and bases. This auxiliary string can either be generated and certified by a trusted third party or generated by the verifier who proves in ZK its correctness during a setup phase which is run once and for all and amortizes if the protocol is run often.

As the Σ^{exp}-protocol is a Σ-protocol it can easily be combined with other Σ-protocols by boolean operators AND and OR with standard composition techniques. Arbitrary monotone access structures can be realized efficiently using secret sharing schemes such as [Sha79] as described in [CDS94]. Yet, these techniques can also be adopted for usage with the protocols given in [BCM05,CKY09]. The major advantage of the Σ^{exp} protocol is that it can easily be transformed into NIZK and CZK with the above standard techniques and hence is much better suited for use in practical applications. For this, we will incorporate the Σ^{exp}-protocol into our compiler framework.

References

[ARS05] A. Adelsbach, M. Rohe, and A.-R. Sadeghi. Complementing zero-knowledge watermark detection: Proving properties of embedded information without revealing it. *Multimedia Systems*, 11(2):143–158, 2005.

[Ate04] G. Ateniese. Verifiable encryption of digital signatures and applications. *ACM Transactions on Information and System Security*, 7(1):1–20, February 2004.

[BCC04] E. Brickell, J. Camenisch, and L. Chen. Direct anonymous attestation. In *Proc. ACM CCS 2004*, pages 132–145. ACM, 2004.

[BCK+08] E. Bangerter, J. Camenisch, S. Krenn, A.-R. Sadeghi, and T. Schneider. Automatic generation of sound zero-knowledge protocols. Cryptology ePrint Archive, Report 2008/471, 2008.

[BCM05] E. Bangerter, J. Camenisch, and U. Maurer. Efficient proofs of knowledge of discrete logarithms and representations in groups with hidden order. In *International Workshop on Practice and Theory in Public-Key Cryptography – PKC 05*, volume 3386 of *LNCS*, pages 154–171. Springer, 2005.

[BNPS05] M. Barbosa, R. Noad, D. Page, and N.P. Smart. First steps toward a cryptography-aware language and compiler. Cryptology ePrint Archive, Report 2005/160, 2005.

[Bou00] F. Boudot. Efficient proofs that a committed number lies in an interval. In *Advances in Cryptology – EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 431–444. Springer, 2000.

[Bra94] S. Brands. Untraceable off-line cash in wallet with observers. In *Advances in Cryptology – CRYPTO 93*, volume 773 of *LNCS*, pages 302–318. Springer, 1994.

[Bra97] S. Brands. Rapid demonstration of linear relations connected by boolean operators. In *Advances in Cryptology – EUROCRYPT 97*, volume 1233 of *LNCS*, pages 318–333. Springer, 1997.

[Bri04] T. Briner. Compiler for zero-knowledge proof-of-knowledge protocols. Master's thesis, ETH Zurich, 2004.

[BS02] E. Bresson and J. Stern. Proofs of knowledge for non-monotone discrete-log formulae and applications. In *ISC '02: Proceedings of the 5th International Conference on Information Security*, pages 272–288. Springer, 2002.

[Cam98] J. Camenisch. *Group Signature Schemes and Payment Systems Based on the Discrete Logarithm Problem*. PhD thesis, ETH Zurich, Konstanz, 1998.

[CDS94] R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *Advances in Cryptology – CRYPTO 94*, volume 839 of *LNCS*, pages 174–187. Springer, 1994.

[CFT98] A. Chan, Y. Frankel, and Y. Tsiounis. Easy come - easy go divisible cash. Technical Report TR-0371-05-98-582, GTE, 1998. Updated version with corrections.

[CH02] J. Camenisch and E. V. Herreweghen. Design and implementation of the idemix anonymous credential system. In *Proc. ACM CCS 2002*, pages 21–30. ACM, 2002. `http://www.zurich.ibm.com/security/idemix/`.

[CKY09] J. Camenisch, A. Kiayias, and M. Yung. On the portability of generalized schnorr proofs. Cryptology ePrint Archive, Report 2009/050, 2009. To appear at *EUROCRYPT 2009*.

[CL01] J. Camenisch and A. Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *Advances in Cryptology – EUROCRYPT 2001*, volume 2045, pages 93–118. Springer, 2001.

[CL04] J. Camenisch and A. Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In *Advances in Cryptology – CRYPTO 2004*, volume 3152 of *LNCS*, pages 56–72. Springer, 2004.

[CM99] J. Camenisch and M. Michels. Proving in zero-knowledge that a number is the product of two safe primes. In *Advances in Cryptology – EUROCRYPT 99*, volume 1592 of *LNCS*, pages 107–122. Springer, 1999.

[Cra96] R. Cramer. *Modular Design of Secure yet Practical Cryptographic Protocols*. PhD thesis, CWI and University of Amsterdam, 1996.

[CRS05] J. Camenisch, M. Rohe, and A.-R. Sadeghi. Sokrates - a compiler framework for zero-knowledge protocols. In *Western European Workshop on Research in Cryptology – WEWoRC 05*, 2005.

[CS97] J. Camenisch and M. Stadler. Efficient group signature schemes for large groups (extended abstract). In *Advances in Cryptology – CRYPTO 97*, volume 1294, pages 410–424. Springer, 1997.

[Dam00] I. Damgård. Efficient concurrent zero-knowledge in the auxiliary string model. In *Advances in Cryptology – EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 418–430. Springer, 2000.

[DF02] I. Damgård and E. Fujisaki. A statistically-hiding integer commitment scheme based on groups with hidden order. In *Advances in Cryptology – ASIACRYPT 2000*, volume 2501 of *LNCS*, pages 77–85. Springer, 2002.

[DFK+93] C. Dwork, U. Feige, J. Kilian, M. Naor, and M. Safra. Low communication 2-prover zero-knowledge proofs for np. In *Advances in Cryptology – CRYPTO 92*, volume 740 of *LNCS*, pages 215–227, London, UK, 1993. Springer.

[FS87] A. Fiat and A. Shamir. How to prove yourself: practical solutions to identification and signature problems. In *Advances in Cryptology – CRYPTO 86*, volume 263 of *LNCS*, pages 186–194. Springer, 1987.

[GMW91] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the ACM*, 38(1):691–729, 1991. Preliminary version in 27th FOCS, 1986.

[IKOS07] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. Zero-knowledge from secure multiparty computation. In *Proc. 39th ACM Symp. on Theory of Computing – STOC 07*, pages 21–30, New York, NY, USA, 2007. ACM.

[Lip03] H. Lipmaa. On diophantine complexity and statistical zeroknowledge arguments. In *Advances in Cryptology – ASIACRYPT 2000*, volume 2894 of *LNCS*. Springer, 2003.

[LPS08] Y. Lindell, B. Pinkas, and N. Smart. Implementing two-party computation efficiently with security against malicious adversaries. In *Security in Communication Networks – SCN 2008*, volume 5229 of *LNCS*, pages 2–20. Springer, 2008.

[MP03] D. Micciancio and E. Petrank. Simulatable commitments and efficient concurrent zero-knowledge. In *Advances in Cryptology – EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 140–159. Springer, 2003.

[Oka95] T. Okamoto. An efficient divisible electronic cash scheme. In *Advances in Cryptology – CRYPTO 95*, volume 963 of *LNCS*, pages 438–451. Springer, 1995.

[Ped92] T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Advances in Cryptology – CRYPTO 91*, volume 576 of *LNCS*, pages 129–140. Springer, 1992.

[PRS02] M. Prabhakaran, A. Rosen, and A. Sahai. Concurrent zero knowledge with logarithmic round-complexity. In *Proc. 43rd IEEE Symp. on Foundations of Comp. Science – FOCS 02*, pages 366–375, Washington, DC, USA, 2002. IEEE Computer Society.

[Sch91] C. Schnorr. Efficient signature generation by smart cards. *Journal Of Cryptology*, 4(3):161–174, 1991.

[Sha79] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[Vis06] I. Visconti. Efficient zero knowledge on the internet. *ICALP '06 - Automata, Languages and Programming*, pages 22–33, 2006.