Bringing Zero-Knowledge Proofs of Knowledge to Practice

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

[1] Security Engineering Lab, Bern University of Applied Sciences, Switzerland
endre.bangerter@bfh.ch

[2] Security Engineering Lab, Bern University of Applied Sciences, Switzerland, and University of Fribourg, Switzerland
{stefania.barzan,stephan.krenn}@bfh.ch

[3] Horst Görtz Institute for IT-Security, Ruhr-University Bochum, Germany
{ahmad.sadeghi,thomas.schneider}@trust.rub.de

[4] LSV, École Normale Supérieure de Cachan, France, and CNRS, France, and INRIA, France
tsay@lsv.ens-cachan.fr

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 Trusted Platform Module (TPM) chip.

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 making them accessible to crypto and security engineers. To this end we are developing compilers and related tools that support and partially automate the design, implementation, verification and secure 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 a secret value that satisfies a given relation (*proof of knowledge property*), without the verifier being able to learn anything about the secret (*zero-knowledge*

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

© Springer-Verlag Berlin Heidelberg 2013

property). For a formal definition we refer to [1]. There are fundamental results showing that all relations in NP have ZK-PoK [2–5]. The corresponding protocols are of theoretical relevance, but are much too inefficient to be used in practical applications.

In contrast to these generic, but practically useless protocols, there are various protocols which are efficient enough for real world use. 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 [6], interactive verifiable computation [7], group signatures [8], secure watermark detection [9], and efficient secure multiparty computation [10] – just to name a few.

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) [11], 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 [12], which was released by IBM into the Eclipse Higgins project, an open source effort dedicated to developing software for user-centric identity management. Identity mixer is probably one of the most advanced protocol suites supporting the “transient relationship paradigm”.

Up to now, design, implementation and verification of the formal cryptographic security properties (i.e., zero-knowledge and proof of knowledge property) as well as code security properties (e.g., security against buffer overflows, race conditions, side channel vulnerabilities) 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 are considerably more complex than other crypto primitives such as signature- and encryption schemes or hash functions.

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 partially automate the design, implementation, verification, and secure implementation of ZK-PoK protocols. For instance the compiler which is part of our toolbox, will take as input a high-level specification of the goals of a ZK-PoK, automatically find a corresponding protocol, and output its implementation in, e.g., Java or C code. We have already developed and implemented a language and compiler that automates the latter step from protocol specification to code generation. Finding a protocol from a high-level specification is subject of ongoing research. Also, we are working on tool-based support for the verification of the security properties of ZK-PoK.

In the following we describe the challenges pertaining to using ZK-PoK in practice in Sec. 2 and give an overview of a solution blueprint and first results on solving these challenges in Sec. 3.

1.1 Related Work

ZK-PoK were introduced in [15], and the first efficient protocols for preimage proofs in groups of known order were given in [6, 16]. Unified frameworks for preimage proofs in known order groups were given in the following by [17–20]. A profound analysis of the Σ^{Φ}-protocol was performed by Cramer [21].

The first efficient solution for proofs in unknown order groups was given in [22] and has been corrected by Damgård and Fujisaki [23]. Subsequently, other variants overcoming some of their restrictions have been proposed. In very recent work, a long overdue unified framework for exponentiation homomorphisms in arbitrary groups was given by Camenisch et al. [24].

An efficient way to combine arbitrary Σ-protocols was described by Cramer et al. [17].

To bridge the gap between theory and practice, a first prototype of a zero-knowledge compiler was started in [25, 26], and was later extended in [27]. Yet, its authors state explicitly that it was designed as a proof of concept prototype only. This prototype handles proofs in known order groups only, and includes neither a verification tool nor extensions to achieve concurrent ZK or non-interactivity. Unfortunately, multiple proofs are combined in a very inefficient way only. Furthermore, the input language of this compiler is less intuitive than ours.

Our input language was inspired by the commonly used notation of Camenisch and Stadler [28]. Yet, this is an inprecise and ambiguous notation. Therefore we augment it by the missing parts such as group descriptions, etc.

As basic building blocks we apply the techniques from [6, 16, 21] in known order groups, proofs in unknown order groups are done by applying those from [23, 27]. Predicates are combined using the method described in [17] instantiated with Shamir's secret sharing scheme [29]. To obtain non-interactive ZK and concurrent ZK we use the Fiat-Shamir heuristic [30].

Similar work to ours was performed in the field of secure function evaluation [31, 32]. Their compilers allow to specify the function to be evaluated in a high-level language, and output executable code. In principle, zero-knowledge proofs could be realized by secure function evaluations. Yet, the resulting protocols are significantly less efficient than those generated by our compiler.

Compiler support for an efficient and secure low-level implementation of cryptographic primitives resistant against software side-channels [33] and applications to elliptic curve cryptography [34] is provided by Cryptography Aware language and cOmpiler (CAO) [35].

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 [6] realizes the first semantic goal mentioned above, and verifiable encryption protocols [36] 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). Finally, by a *(protocol) implementation* we refer to actual code (e.g., in C or Java) realizing a specification.

Now, let us turn to the challenges mentioned above.

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 of 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, less complex 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 a 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. The 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 and other primitives can be composed in a modular way and are easily accessible to designers.

Efficiency of implementation process. The step going from the protocol specification of a ZK-PoK to its 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 [12, 37] protocols have shown that a manual implementation can be tedious and error prone and easily takes person weeks. Moreover, protocol specifications are often written by cryptographers while 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. For instance, parts of the prover's algorithm in the DAA protocol [11] are run inside a relatively simple and low cost TPM chip while the verifier's algorithm may run 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 the "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 should specially focus on the optimization of crypto operations.

Optimization in general, requires substantial experience and an intricate understanding of the runtime environment.

Correctness and security of implementations. The correctness and security of the protocol implementations is primordial. One can distinguish two classes of correctness and security properties. One are the cryptographic security properties, which are formalized mathematically and are present already on a protocol specification level. These properties are: correctness (the protocol works when prover and verifier are honest), zero-knowledge and proof of knowledge. At the current state of the art, the crypto community will not accept a ZK-PoK protocol, unless these properties are formally proven on a specification level. These proofs are often non-trivial and certainly tedious and time consuming, and as a result there exist various published protocols that contain flaws in their security analysis. For instance the security proof in [22] was incomplete as outlined and corrected in [23].

Of course one also needs to assert that those security properties are indeed assured by the implementation of the protocol (i.e., that an implementation correctly reflects the specification).

Of equal importance are security issues that occur at an implementation level. These include security against generic implementation errors like buffer overflows, race conditions etc., but also crypto specific code problems, such as side-channel vulnerabilities. Getting these code security issues right requires substantial know-how, which is often not part of the skill set of developers.

3 Solution Blueprint and Results

In the following we give a brief description of our compiler suite (see Fig. 1), and sketch how it can be used for resolving the challenges explained above. Also first results achieved for each of these challenges will be described.

From a usage perspective, our compiler suite takes a description of the semantic goal in a *high-level language*, and outputs a protocol implementation together with a formal proof of its correctness. From a technical point of view, the compiler is divided into three parts, which we want to discuss briefly now:

- The compiler will take a description of the semantic goal in a *high-level language* as input, and translate it into a *protocol specification* in a first compiler step (the *high-level compiler*) by choosing the most appropriate techniques to meet the user's requirements. This protocol specification describes a unique protocol, without containing the exact algorithms or single messages to be exchanged, etc.
- In a second step, the *protocol compiler* will expand this protocol specification into C or Java code, as well as LaTeX-code for documentation purposes.
- Both compiler steps will add annotations, including information about decisions made, to their output. The semantic goal, the protocol specification and its implementation will be given to the *protocol verification toolbox*, which using those annotations will formally verify that the implementation indeed realizes the semantic goal in a secure way.

Let us now turn to how we plan to tackle the problems stated in Sec. 2.

The "efficiency of implementation process" and "efficiency of code" challenges are equally important but less difficult to achieve than the two others; we therefore only discuss them briefly.

Efficiency of implementation process. This goal is achieved inherently by our compiler based approach, as the implementation is automatized. Our first prototype runs within less than one second, and we expect the final version to run within a couple of seconds. To ease usage of our compiler suite, a tool-chain with a consistent user interface could be given to our compiler. This could be based on an IDE such as Eclipse and support the developer by syntax highlighting, performance testing, etc.

The input language is inspired by the notation introduced in [28]. Yet, to remove unambiguities, some more information has to be added to this language.

Let us consider the following example:

$$ZKPoK\left[(\chi) : y = g^{\chi}\right]$$

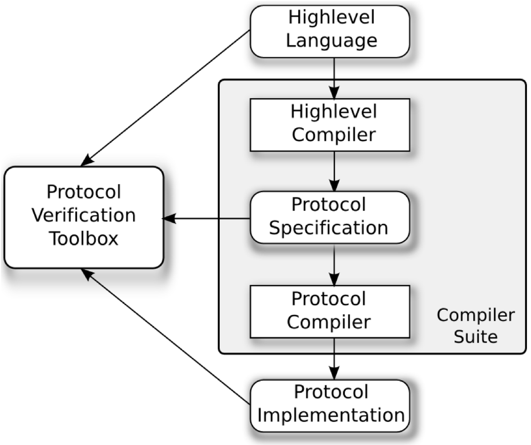

Fig. 1. Architecture of our framework for automatic generation and verification of ZK-PoK protocols

specifies a proof of knowledge of the discrete logarithm $x \in G$ of $y \in H$ in base $g \in H$. This notation does not contain any information about the groups G, H, the order of g, y, or the knowledge error that has to be achieved.

Still, from having the above protocol description and knowledge about the groups, etc., it's straightforward to obtain the input of our compiler, such as:

```
Declaration{
  Prime(1024) p;
  Prime(160) q;
  G=Zmod+(q) x;
  H=Zmod*(p) g, y;
}

ProtocolInputs{
  ProverPrivate  := x;
  ProverPublic   := p,q,g,y;
  VerifierPublic := p,q,g,y;
}

ProtocolProperties{
  KnowledgeError      := 80;
  ProtocolComposition := P_1;
}

SigmaPhi P_1 {
  Homomorphism (phi : G -> H : (a) |-> (g^a));
  ChallengeLength := 80;
  Relation ((y) = phi(x));
}
```

Lines 01-06 declare the group elements being used in the subsequently described protocol. In lines 08-12 the inputs of the parties are described. If for example the verifier did not know a public parameter, it would be sent by the prover in a synchronization step. Lines 14-17 declare the knowledge error that has to be reached, and how the predicates should be composed. Finally, lines 18-23 describe the only predicate in this example. This description is a direct analogon to the Camenisch-Stadler notation [28].

Efficiency of code. As mentioned above, the "efficiency of code" challenge has to be dealt with on two levels. On a high level, the compiler has to find the most efficient protocol specification meeting a given semantic proof goal. The choice of the proof technique to use will depend on the priorities the user gives to communicational- respectively computational complexity, as there is often a tradeoff between those. A deeper discussion is given in the next paragraph. On a low level, we'll provide a compiler backend that outputs code in the CAO ("Cryptography Aware language and cOmpiler") language [35]. 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 within the CACE project.

Let us discuss the remaining challenges in more detail.

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 [28] which is used to formulate the intended semantic goal. We enrich this with non-functional properties which allow to specify optimization goals (e.g., optimize computational or communicational 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.

In our architecture (cf. Fig. 1) the high-level compiler is responsible for finding a protocol specification that realizes the semantic proof goal and simultaneously takes into consideration the user's non-functional specifications. To enable the compiler to make "good" decisions, the compiler backend reports the costs on a specific target platform upwards to the high-level compiler. For example efficient interval proofs can be realized either with the techniques of [38] or [39] with different costs.

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 [17, 19–21]. Our extended theoretical framework is capable to cope with arbitrary combinations of protocols in hidden order groups (e.g., RSA groups) as well [14, 23, 24]. To this end, we have conceived the new Σ^{exp} protocol [27], which yields efficient ZK-PoK in a more modular manner than the existing protocols [14, 23, 24].

A first prototype of our compiler and semantic language [27, 40] implements a subset of the envisaged compiler framework. It already supports the generation of various crypto-systems such as Pedersen commitments/verifiable secret sharing [41], Schnorr authentication/signatures [6], electronic cash [42–44], group signatures [45], or ring signatures [17].

Correctness and security of implementations. One of our main goals concerning the security of the code output by the compiler, is to formally verify the zero-knowledge and proof of knowledge properties. To this end we are developing a protocol verification toolbox as part of our compiler framework (see Fig. 1). Its task is to accomplish a semi- or (ideally) fully automatic formal verification of these properties.

We currently focus on the proof of knowledge property. The toolbox takes as input the user's description of the semantic goal and the protocol implementation (output by the compiler). It then interprets this information in order to assemble a proof goal for the Isabelle/HOL theorem prover [46]. The theorem prover then formally verifies whether the protocol is indeed a proof of knowledge for the given goal (by constructing a knowledge extractor).

One step towards automating this verification process is to consider the most relevant proof strategies used in existing published proofs and to develop corresponding proof tactics for the theorem prover. Also, to facilitate this automated verification, the different parts of our compiler (i.e., the high-level compiler respectively the protocol compiler) annotate helper data to the code they output.

We have already formally verified the proof of knowledge property for basic protocols such as those in [6, 23] and generic AND– and OR– compositions among those. Currently, we are tackling more complex protocols.

Last but not least, also to assert code security properties (e.g., against buffer overflows and side channel attacks) we rely on the verified compiler backend to output CAO-code [35] (see above). The CAO language is designed to automatically generate secure implementations resistant against software side-channels.

References

1. Bellare, M., Goldreich, O.: On Defining Proofs of Knowledge. In: Brickell, E.F. (ed.) CRYPTO 1992. LNCS, vol. 740, pp. 390–420. Springer, Heidelberg (1993)
2. Dwork, C., Feige, U., Kilian, J., Naor, M., Safra, M.: Low Communication 2-Prover Zero-Knowledge Proofs for NP. In: Brickell, E.F. (ed.) CRYPTO 1992. LNCS, vol. 740, pp. 215–227. Springer, Heidelberg (1993)
3. Goldreich, O., Micali, S., 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), 691–729 (1991); Preliminary version in 27th FOCS (1986)
4. Ishai, Y., Kushilevitz, E., Ostrovsky, R., Sahai, A.: Zero-knowledge from secure multiparty computation. In: ACM Symposium on Theory of Computing – STOC 2007, pp. 21–30. ACM Press, New York (2007)
5. Prabhakaran, M., Rosen, A., Sahai, A.: Concurrent zero knowledge with logarithmic round-complexity. In: IEEE Symposium on Foundations of Computer Science – FOCS 2002, pp. 366–375. IEEE Computer Society, Washington, DC (2002)

6. Schnorr, C.: Efficient signature generation by smart cards. Journal of Cryptology 4(3), 161–174 (1991)
7. Camenisch, J., Michels, M.: Proving in Zero-Knowledge that a Number Is the Product of Two Safe Primes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 107–122. Springer, Heidelberg (1999)
8. Camenisch, J.: Group Signature Schemes and Payment Systems Based on the Discrete Logarithm Problem. PhD thesis, ETH Zurich, Konstanz (1998)
9. Adelsbach, A., Rohe, M., Sadeghi, A.R.: Complementing zero-knowledge watermark detection: Proving properties of embedded information without revealing it. Multimedia Systems 11(2), 143–158 (2005)
10. Lindell, Y., Pinkas, B., Smart, N.P.: Implementing Two-Party Computation Efficiently with Security Against Malicious Adversaries. In: Ostrovsky, R., De Prisco, R., Visconti, I. (eds.) SCN 2008. LNCS, vol. 5229, pp. 2–20. Springer, Heidelberg (2008)
11. Brickell, E., Camenisch, J., Chen, L.: Direct anonymous attestation. In: Atluri, V., Backes, M., Basin, D.A., Waidner, M. (eds.) ACM Conference on Computer and Communications Security – CCS 2004, pp. 132–145. ACM Press (2004)
12. Camenisch, J., Herreweghen, E.V.: Design and implementation of the idemix anonymous credential system. In: Atluri, V. (ed.) ACM Conference on Computer and Communications Security – CCS 2002, pp. 21–30. ACM Press (2002), `http://www.zurich.ibm.com/security/idemix/`
13. Bangerter, E.: Efficient Zero-Knowledge Proofs of Knowledge for Homomorphisms. PhD thesis, Ruhr-University Bochum (2005)
14. Bangerter, E., Camenisch, J., Maurer, U.: Efficient Proofs of Knowledge of Discrete Logarithms and Representations in Groups with Hidden Order. In: Vaudenay, S. (ed.) PKC 2005. LNCS, vol. 3386, pp. 154–171. Springer, Heidelberg (2005)
15. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof-systems. In: ACM Symposium on Theory of Computing – STOC 1985, pp. 291–304. ACM Press, New York (1985)
16. Guillou, L.C., Quisquater, J.-J.: A "Paradoxical" Identity-Based Signature Scheme Resulting from Zero-Knowledge. In: Goldwasser, S. (ed.) CRYPTO 1988. LNCS, vol. 403, pp. 216–231. Springer, Heidelberg (1990)
17. Cramer, R., Damgård, I., Schoenmakers, B.: Proof of Partial Knowledge and Simplified Design of Witness Hiding Protocols. In: Desmedt, Y.G. (ed.) CRYPTO 1994. LNCS, vol. 839, pp. 174–187. Springer, Heidelberg (1994)
18. Camenisch, J., Stadler, M.: Proof systems for general statements about discrete logarithms. Technical Report 260, Institute for Theoretical Computer Science, ETH Zürich (1997)
19. Brands, S.: Rapid Demonstration of Linear Relations Connected by Boolean Operators. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 318–333. Springer, Heidelberg (1997)
20. Bresson, E., Stern, J.: Proofs of Knowledge for Non-monotone Discrete-Log Formulae and Applications. In: Chan, A.H., Gligor, V.D. (eds.) ISC 2002. LNCS, vol. 2433, pp. 272–288. Springer, Heidelberg (2002)
21. Cramer, R.: Modular Design of Secure yet Practical Cryptographic Protocols. PhD thesis, CWI and University of Amsterdam (1997)
22. Fujisaki, E., Okamoto, T.: Statistical Zero Knowledge Protocols to Prove Modular Polynomial Relations. In: Kaliski Jr., B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 16–30. Springer, Heidelberg (1997)

23. Damgård, I., Fujisaki, E.: A Statistically-Hiding Integer Commitment Scheme Based on Groups with Hidden Order. In: Zheng, Y. (ed.) ASIACRYPT 2002. LNCS, vol. 2501, pp. 125–142. Springer, Heidelberg (2002)
24. Camenisch, J., Kiayias, A., Yung, M.: On the Portability of Generalized Schnorr Proofs. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 425–442. Springer, Heidelberg (2009)
25. Briner, T.: Compiler for zero-knowledge proof-of-knowledge protocols. Master's thesis, ETH Zurich (2004)
26. Camenisch, J., Rohe, M., Sadeghi, A.R.: Sokrates - a compiler framework for zero-knowledge protocols. In: Western European Workshop on Research in Cryptology – WEWoRC 2005 (2005)
27. Bangerter, E., Camenisch, J., Krenn, S., Sadeghi, A.R., Schneider, T.: Automatic generation of sound zero-knowledge protocols. Cryptology ePrint Archive, Report 2008/471 (2008), `http://eprint.iacr.org/`, Poster Session of EUROCRYPT 2009
28. Camenisch, J.L., Stadler, M.A.: Efficient Group Signature Schemes for Large Groups (Extended Abstract). In: Kaliski Jr., B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 410–424. Springer, Heidelberg (1997)
29. Shamir, A.: How to share a secret. Communications of the ACM 22(11), 612–613 (1979)
30. Fiat, A., Shamir, A.: How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (1987)
31. MacKenzie, P., Oprea, A., Reiter, M.K.: Automatic generation of two-party computations. In: Jajodia, S., Atluri, V., Jaeger, T. (eds.) ACM Conference on Computer and Communications Security – CCS 2003, pp. 210–219. ACM Press (2003)
32. Malkhi, D., Nisan, N., Pinkas, B., Sella, Y.: Fairplay – a secure two-party computation system. In: Proceedings of the 13th Conference on USENIX Security Symposium – SSYM 2004 (2004), `http://www.cs.huji.ac.il/project/Fairplay/fairplay.html`
33. Barbosa, M., Page, D.: On the automatic construction of indistinguishable operations. Cryptology ePrint Archive, Report 2005/174 (2005), `http://eprint.iacr.org/`
34. Barbosa, M., Moss, A., Page, D.: Compiler Assisted Elliptic Curve Cryptography. In: Meersman, R. (ed.) OTM 2007, Part II. LNCS, vol. 4804, pp. 1785–1802. Springer, Heidelberg (2007)
35. Barbosa, M., Noad, R., Page, D., Smart, N.P.: First steps toward a cryptography-aware language and compiler. Cryptology ePrint Archive, Report 2005/160 (2005), `http://eprint.iacr.org/`
36. Ateniese, G.: Verifiable encryption of digital signatures and applications. ACM Transactions on Information and System Security 7(1), 1–20 (2004)
37. Camenisch, J.L., Lysyanskaya, A.: An Efficient System for Non-transferable Anonymous Credentials with Optional Anonymity Revocation. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 93–118. Springer, Heidelberg (2001)
38. Boudot, F.: Efficient Proofs that a Committed Number Lies in an Interval. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 431–444. Springer, Heidelberg (2000)
39. Lipmaa, H.: On Diophantine Complexity and Statistical Zero-Knowledge Arguments. In: Laih, C.-S. (ed.) ASIACRYPT 2003. LNCS, vol. 2894, pp. 398–415. Springer, Heidelberg (2003)

40. Bangerter, E., Briner, T., Henecka, W., Krenn, S., Sadeghi, A.-R., Schneider, T.: Automatic Generation of Sigma-Protocols. In: Martinelli, F., Preneel, B. (eds.) EuroPKI 2009. LNCS, vol. 6391, pp. 67–82. Springer, Heidelberg (2010)
41. Pedersen, T.P.: Non-interactive and Information-Theoretic Secure Verifiable Secret Sharing. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 129–140. Springer, Heidelberg (1992)
42. Brands, S.: Untraceable Off-Line Cash in Wallets with Observers. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 302–318. Springer, Heidelberg (1994)
43. Chan, A., Frankel, Y., Tsiounis, Y.: Easy come - easy go divisible cash. Technical Report TR-0371-05-98-582, GTE (1998), Updated version with corrections
44. Okamoto, T.: An Efficient Divisible Electronic Cash Scheme. In: Coppersmith, D. (ed.) CRYPTO 1995. LNCS, vol. 963, pp. 438–451. Springer, Heidelberg (1995)
45. Camenisch, J., Lysyanskaya, A.: Signature Schemes and Anonymous Credentials from Bilinear Maps. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 56–72. Springer, Heidelberg (2004)
46. Paulson, L.C.: Isabelle. LNCS, vol. 828. Springer, Heidelberg (1994)