YACZK: Yet Another Compiler for Zero-Knowledge *
(Poster Abstract)

Endre Bangerter[1], Stephan Krenn[2], Ahmad-Reza Sadeghi[3], Thomas Schneider[3]

[1] Bern University of Applied Sciences, Biel-Bienne, Switzerland
endre.bangerter@jdiv.org
[2] Bern University of Applied Sciences, Biel-Bienne, Switzerland, and
University of Fribourg, Switzerland
stephan.krenn@bfh.ch
[3] Horst Görtz Institute for IT-Security, Ruhr-University Bochum, Germany
{ahmad.sadeghi,thomas.schneider}@trust.rub.de

Abstract. Automatic generation of cryptographic protocols is an emerging field of research which aims to bring complex protocols into practice.
In this work we discuss the desired properties of a compiler for automatic generation of zero-knowledge proof of knowledge (ZKPoK) protocols. We evaluate and compare existing approaches with respect to these properties:
In particular, it seems to us that the authors of the paper accepted for USENIX Security 2010 (ZKPDL: A Language-Based System for Efficient Zero-Knowledge Proofs and Electronic Cash) were not aware of our previous work done within the European project "Computer Aided Cryptography Engineering" (CACE).
We hope that this poster stimulates scientific debates and exchange in this field of research.

1 Introduction

Three of the main challenges in applied cryptography are the design of protocols for certain purposes, their efficient implementation and the verification of these implementations. In particular, this is true for applications which require complex, non-standard crypto-primitives to be used as basic building blocks. To overcome these challenges and the associated security risks, a direction of research in applied cryptography has started to develop tools to automate these processes, e.g., in the area of secure multi-party computations [MOR03,MNPS04,BDNP08,BLW08,DGKN09].

Privacy-preserving applications, such as *idemix* [CH02,CL01] or *Direct Anonymous Attestation* [BCC04], are often relying on zero-knowledge proofs of knowledge (ZKPoK), i.e., subprotocols which allow a prover to convince a verifier that he knows some secret piece of information, without the verifier being able to learn anything about it. For instance, a user might be interested in proving that it is allowed to access some online service, without fully revealing his identity. Typical applications that use ZKPoK as fundamental building block are identification schemes [Sch91], interactive verifiable computation [CM99], and group signatures [Cam98] – just to name a few.

Yet, finding highly efficient protocols that realize complex proof-goals turns out to be a non-trivial task which requires intricate knowledge of many techniques and tricks such as [GQ90,Sch91,CDS94], [FO97,DF02,Ban05,BCK+08,Sma09]. To overcome this challenge, compilers for automatic generation of ZKPoK protocols have been developed:

– **CACE compiler.** The goal of the European project CACE (Computer Aided Cryptography Engineering) [4] is "To enable verifiable secure cryptographic software engineering to non-experts by developing a toolbox which automatically produces high-performance solutions from natural

* This work was performed within the FP7 EU project CACE (Computer Aided Cryptography Engineering).
[4] http://www.cace-project.eu

specifications". This toolbox includes languages and compilers for different abstraction levels including cryptographic primitives, secure communication, secure multi-party computation protocols, ZKPoK protocols as well as formal verification aspects [BBB+10]. Work package three of the CACE project is devoted to the development of a ZKPoK compiler. First results include a prototype compiler [Bri04,BBK+09,BBH+09] written in Java and a completely re-designed CACE compiler [BKS+09,BBG+09,ABB+10] written in Python.

– **ZKPDL compiler.** An alternative ZKPoK compiler was accepted for USENIX Security 2010 [MEK+10]. Unfortunately, the authors of [MEK+10] were not completely aware of the previously published works and code of the CACE compiler. Therefore, we would like to give a comparison of both compilers in this paper.

2 Comparison of ZKPoK Compilers

In the following we give desired properties of a compiler for automatic generation of ZKPoK protocols and compare the two compiler approaches w.r.t. these properties.

1. **Interfaces:** ZKPoK are hardly ever used as standalone applications, but rather embedded into larger protocols and systems. Thus, the output of a ZK-PoK compiler has to have clearly defined interfaces to other applications.
   – The ZKPDL compiler achieves this by providing a C++ API for the libraries generated by their compiler.
   – The CACE compiler generates entire C applications which can be called as subroutines by higher-level applications or executed directly.
2. **Modular Design:** For the compiler itself to be as extendible and flexible as possible, it is important to have a modular design. That is, it should allow for an easy exchange of single components such as the code generation backend, the underlying mathematical libraries, the network interface being used in the generated protocols, etc.
   – Modularity is an intricate part of the design decisions of the CACE compiler [BBK+09, Fig. 3]. The modularity of the compiler allowed for instance to easily add two backends: a code generation backend for C and a LaTeX backend for automatic generation of documentations.
   – According to [MEK+10, Fig. 2], the ZKPDL compiler consists of a single `compile()` routine.
3. **Verifiability:** In the context of security-sensitive applications, the need to ensure that "a program does what it is supposed to do" is even stronger than for every-day applications. For this reason, various standards such as Common Criteria require cryptographic applications to come along with a certificate of their correctness. Letting the user of a compiler prove the correctness of the compilation process *by hand* after every run of the compiler is nearly impossible or at least makes the compiler substantially less useful.
   – The CACE compiler includes a profound static code analysis of the protocols generated by the compiler. Additionally, a formal verification toolbox is built into the compiler as proposed in [BBK+09] and implemented in [ABB+10]. Currently, this verification toolbox automatically gives a formal verification of the proof of knowledge property (i.e., ensures that a malicious prover, not knowing the secret, cannot convince the verifier) for a large class of inputs.
   – The ZKPDL leaves verification as future work.

In addition to these general concepts of cryptographic compilers, we give a more detailed comparison of the two compiler s in Table 1:

The CACE compiler allows to describe a larger class of proof goals than the ZKPDL compiler. For instance, the latter is not capable of proving logical "OR" relations, and can therefore not be used to generate protocols for group signature schemes [Cam98]. Also, in contrast to the CACE compiler,

	CACE Compiler	ZKPDL compiler
Basic proof goals	knowledge of preimages under arbitrary group homomorphisms, including RSA or Paillier type homomorphisms [Pai99,RSA78]	exponentiation homomorphisms only
Basic protocols	SigmaPhi [Sch91], Damgård/Fujisaki [DF02], SigmaExp [BCK+08]	SigmaPhi [Sch91], Damgård/Fujisaki [DF02]
Composition techniques	Boolean AND, OR, and proving knowledge of k out of n secret values [CDS94]	Boolean AND only
Supported macros	none	(i) equality of secret values, (ii) multiplicative relations among secret values, and (iii) secret value belongs to public interval
Input language	inspired by the standard notation for ZK-PoK [CS97]	inspired by the English language
Output language	C for implementation, LaTeX for documentation	specifically design language with interpreter and API to C++
Type of output	complete source code, which can directly be compiled to machine code	meta-code or libraries, which partly have to be instantiated by the calling procedure
Optimizations	reduction of redundant terms in the proof goal	(i) reduction of redundant terms in the proof goal (ii) possibility of caching precomputations to reduce runtime
Additional features	(i) existence of a formal verification toolbox, which proves the correctness of the compilation process (ii) modular design with interfaces to other cryptographic compilers developed within CACE project	possibility to specify the generation of the protocol inputs

Table 1: Detailed Comparison of ZKPoK Compilers

the ZKPDL compiler is not suited to prove statements about homomorphic encryption schemes commonly used in secure multi-party computations such as proving knowledge of the plaintext encrypted within an RSA [RSA78] or Paillier [Pai99] ciphertext, as ZKPDL compiler only supports preimage proofs of homomorphisms of the form $\phi(w_1, \ldots, w_n) = \prod_{i=1}^{n} g_i^{w_i}$.

On the other hand, the ZKPDL compiler can yield lower runtimes for certain applications as it allows to pre-compute and cache frequently used values.

3 Conclusion

In summary, the ZKPDL Compiler seems to be suited well for generating highly efficient protocols for a specific class of protocols such as electronic cash, whereas the CACE Compiler is able to generate a larger variety of ZKPoK protocols.

We hope that the comparison of the different compilers available for ZKPoK protocols presented in this paper, helps users to pick the right tool for their needs.

References

ABB+10. J. Almeida, E. Bangerter, M. Barbosa, S. Krenn, A.-R. Sadeghi, and T. Schneider. A certifying compiler for zero-knowledge proofs of knowledge based on Σ-protocols. ESORICS 2010 (to appear), 2010.

AS01. A. Adelsbach and A.-R. Sadeghi. Zero-knowledge watermark detection and proof of ownership. In *Information Hiding*, volume 2137 of *LNCS*, pages 273–288. Springer, 2001.

Ban05. E. Bangerter. *Efficient Zero-Knowledge Proofs of Knowledge for Homomorphisms*. PhD thesis, Ruhr-University Bochum, 2005.

BBB+10. E. Bangerter, M. Barbosa, D.J. Bernstein, I. Damgard, D. Page, J.I. Pagter, A.-R. Sadeghi, and S. Sovio. Using compilers to enhance cryptographic product development. In *Information Security Solutions Europe (ISSE'10)*, pages 291–301. Vieweg+Teubner, January 2010.

BBG+09. E. Bangerter, S. Barzan, A. Grünert, W. Henecka, S. Krenn, A.-R. Sadeghi, and T. Schneider. CACE - WP3: Zero-knowledge proof of knowledge compiler. `http://zkc.cace-project.eu/`, 2009.

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

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

BCC04. E. Brickell, J. Camenisch, and L. Chen. Direct anonymous attestation. In V. Atluri, M. Backes, D. A. Basin, and M. Waidner, editors, *ACM CCS 04*, pages 132–145. ACM Press, 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. `http://eprint.iacr.org/`. Poster Session of EUROCRYPT 09.

BDNP08. A. Ben-David, N. Nisan, and B. Pinkas. FairplayMP: a system for secure multi-party computation. In *ACM Conference on Computer and Communications Security (CCS'08)*, pages 257–266, 2008. `http://fairplayproject.net/fairplayMP.html`.

BKS+09. E. Bangerter, S. Krenn, A.-R. Sadeghi, T. Schneider, and J.-K. Tsay. On the design and implementation of efficient zero-knowledge proofs of knowledge. In *Software Performance Enhancements for Encryption and Decryption and Cryptographic Compilers – SPEED-CC 09*, October 12-13, 2009.

BLW08. D. Bogdanov, S. Laur, and J. Willemson. Sharemind: A framework for fast privacy-preserving computations. In *European Symposium on Research in Computer Security (ESORICS'08)*, volume 5283, pages 192–206, 2008.

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

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 Y. Desmedt, editor, *CRYPTO 94*, volume 839 of *LNCS*, pages 174–187. Springer, 1994.

CH02. J. Camenisch and E. V. Herreweghen. Design and implementation of the idemix anonymous credential system. In V. Atluri, editor, *ACM CCS 02*, pages 21–30. ACM Press, 2002.

CL01. J. Camenisch and A. Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In B. Pfitzmann, editor, *EUROCRYPT 01*, volume 2045 of *LNCS*, pages 93–118. Springer, 2001.

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

CS97. J. Camenisch and M. Stadler. Efficient group signature schemes for large groups (extended abstract). In B. Kaliski, editor, *CRYPTO 97*, volume 1294 of *LNCS*, pages 410–424. Springer, 1997.

DF02. I. Damgård and E. Fujisaki. A statistically-hiding integer commitment scheme based on groups with hidden order. In Y. Zheng, editor, *ASIACRYPT 02*, volume 2501 of *LNCS*, pages 77–85. Springer, 2002.

DGKN09. I. Damgård, M. Geisler, M. Krøigård, and J. B. Nielsen. Asynchronous multiparty computation: Theory and implementation. In *Public Key Cryptography (PKC'09)*, volume 5443, pages 160–179, 2009. `http://viff.dk`.

FO97. E. Fujisaki and T. Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In B. Kaliski, editor, *CRYPTO 97*, volume 1294 of *LNCS*, pages 16–30. Springer, 1997.

GQ90. L. Guillou and J.-J. Quisquater. A "paradoxical" identity-based signature scheme resulting from zero-knowledge. In S. Goldwasser, editor, *CRYPTO 88*, volume 403 of *LNCS*, pages 216–231. Springer, 1990.

MEK+10. S. Meiklejohn, C. Erway, A. Küpçü, T. Hinkle, and A. Lysyanskaya. ZKPDL: A language-based system for efficient zero-knowledge proofs and electronic cash. USENIX 2010 (to appear), 2010.

MNPS04. D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay — a secure two-party computation system. In *USENIX Security 04*, 2004. `http://www.cs.huji.ac.il/project/Fairplay/fairplay.html`.

MOR03. P. MacKenzie, A. Oprea, and M. K. Reiter. Automatic generation of two-party computations. In *ACM Conference on Computer and Communications Security (CCS'03)*, pages 210–219, 2003.

Pai99. P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In J. Stern, editor, *EUROCRYPT 99*, volume 1592 of *LNCS*, pages 223–238. Springer, 1999.

RSA78. R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

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

Sma09. N. P. Smart, editor. *Final Report on Unified Theoretical Framework of Efficient Zero-Knowledge Proofs of Knowledge*. `http://www.cace-project.eu`, 2009. CACE Project Deliverable.