

The Synergy Between Programming Languages and Cryptography

Edited by

Gilles Barthe¹, Michael Hicks², Florian Kerschbaum³, and Dominique Unruh⁴

1 IMDEA Software Institute – Madrid, ES, gjbarthe@gmail.com

2 University of Maryland, US, mwh@cs.umd.edu

3 SAP Research – Karlsruhe, DE, florian.kerschbaum@sap.com

4 University of Tartu, EE, unruh@ut.ee

Abstract

Increasingly, modern cryptography (crypto) has moved beyond the problem of secure communication to a broader consideration of securing computation. The past thirty years have seen a steady progression of both theoretical and practical advances in designing cryptographic protocols for problems such as secure multiparty computation, searching and computing on encrypted data, verifiable storage and computation, statistical data privacy, and more.

More recently, the programming-languages (PL) community has begun to tackle the same set of problems, but from a different perspective, focusing on issues such as language design (e.g., new features or type systems), formal methods (e.g., model checking, deductive verification, static and dynamic analysis), compiler optimizations, and analyses of side-channel attacks and information leakage.

This seminar helped to cross-fertilize ideas between the PL and crypto communities, exploiting the synergies for advancing the development of secure computing, broadly speaking, and fostering new research directions in and across both communities.

Seminar November 30 to December 5, 2014 – <http://www.dagstuhl.de/14492>

1998 ACM Subject Classification D.4.6 Security and Protection, K.6.5 Security and Protection, F.3.1 Specifying and Verifying and Reasoning about Programs, D.2.4 Software/Program Verification

Keywords and phrases Security, Theory, Languages

Digital Object Identifier 10.4230/DagRep.4.12.29

Edited in cooperation with Matthew Hammer

1 Executive Summary

Michael Hicks

License  Creative Commons BY 3.0 Unported license
© Michael Hicks

The seminar schedule consisted of three components: short, two minute introduction talks (one for each participant), longer technical talks (Section 3) and open discussions on four different subjects. The first two days consisted of the introduction talks, followed by most of the technical talks. The seminar attendees had a mix of backgrounds, with one half (roughly) leaning heavily toward the PL (programming languages) side, and the other half leaning more towards the crypto side. The diversity of talks reflected this diversity of backgrounds,



Except where otherwise noted, content of this report is licensed under a Creative Commons BY 3.0 Unported license

The Synergy Between Programming Languages and Cryptography, *Dagstuhl Reports*, Vol. 4, Issue 12, pp. 29–47
Editors: Gilles Barthe, Michael Hicks, Florian Kerschbaum, and Dominique Unruh



Dagstuhl Reports

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

but there was much opportunity to meet in the middle and discuss open problems. The latter days mixed some remaining technical talks with open discussion sessions focusing on various problems and topics.¹ In particular, participants voted to select four breakout sessions: Secure Computation Compilers, Crypto verification, Obfuscation, and Verified implementations.

This section summarizes some interesting discussions from the seminar, in three parts. First, we consider the activities involved in developing programming languages the interface with cryptography, and surveying the research of the seminar participants. Second, we explore how reasoning in PL and Crypto compare and contrast, and how ideas from one area might be relevant to the other. Finally, we survey open problems identified during the discussions.

Programming languages for cryptography

One connection emerged repeatedly in the talks and discussions: the use of programming languages to do cryptography, e.g., to implement it, optimize it, and prove it correct.

Programming languages can be compiled to cryptographic mechanisms

Programming languages can make cryptographic mechanisms easier to use. For example, the systems Sharemind, ShareMonad, CBMC-GC, and WYSTERIA are all designed to make it easier for programmers to write secure multiparty computations (SMCs).

In an SMC, we have two (or more) parties X and Y whose goal is to compute a function F of their inputs x and y , whereby each party only learns the output $F(x, y)$, but does not “see” the inputs. Cryptographers have developed ways to compute such functions, such as garbled circuits² and computing on secret shares³, without need of a trusted third party. These systems shield the programmer from the workings of these mechanisms, compiling normal-looking programs to use the cryptography automatically. The languages can also provide additional benefits, such compiler-driven optimization.

This line of work is motivated by privacy- and/or integrity-preserving outsourcing of computation, e.g., as promised by The Cloud. Programming languages have been designed to compile to other kinds of crypto aside from SMC, like zero-knowledge proofs and authenticated data structures. Examples include Geppetto⁴, SNARKs for C⁵ and LambdaAuth⁶.

Combinations also exist, such as compiling to support Authenticated SNARKs.

Programming languages for implementing cryptography

The above languages aim to make computations secure through the use of cryptography, introduced by the language’s compiler. We are also interested in implementing the cryptographic algorithms themselves (e.g., for symmetric or public key encryption). The implementation

¹ As a break from the technical program, we went on a group outing to Trier on Wednesday afternoon, where we enjoyed a guided historical tour and enjoyed the city’s Christmas market.

² https://www.usenix.org/legacy/event/sec11/tech/full_papers/Huang.pdf

³ <http://www.math.ias.edu/~avi/PUBLICATIONS/MYPAPERS/GMW87/GMW87.pdf>

⁴ <https://eprint.iacr.org/2014/976.pdf>

⁵ <http://eprint.iacr.org/2013/507>

⁶ <http://amiller.github.io/lambda-auth/>

task could be made easier, more efficient, or more secure by employing a special-purpose language. Two representatives in this space are CAO⁷ and Cryptol⁸. Both are domain-specific, and both make it easier to connect implementations to tools for automated reasoning. The Seminar also featured work on synthesizing cryptography (block ciphers) from constraint-based specifications.⁹

Programming languages methods to prove security of cryptographic protocols and/or their implementations

When a cryptographer defines a cryptographic protocol, she must prove it is secure. Programming languages methods can be used mechanically confirm that a proof of security is correct. Systems like ProVerif¹⁰, CryptoVerif¹¹, EasyCrypt¹² and CertiCrypt¹³ support cryptographic protocol verification, with varying kinds of assurance. These systems build on ideas developed in general verification systems like Coq or Isabelle.

Likewise, when a programmer implements some cryptography (in a language like C), she would like to formally verify that the implementation is correct (no more Heartbleed!). For example, we'd like to know that an implementation does not have side channels, it uses randomness sufficiently, it has no buffer overflows, etc. Once again, verification can be achieved using tools that are underpinned by PL methods developed in formal verification research. Frama-C¹⁴ and Fstar¹⁵ have been used to verify implementations.

Formal reasoning for PL and Crypto

Beyond using PLs as a tool for easier/safer use of Crypto, there is an opportunity for certain kinds of thinking, or *reasoning*, to cross over fruitfully between the PL and Crypto communities. In particular, both communities are interested in formalizing systems and proving properties about them but they often use different methods, either due to cultural differences, or because the properties and systems of interest are simply different. During the seminar we identified both analogous, similar styles of reasoning in two communities and connection points between the different styles of reasoning.

Analogies between PL and Crypto reasoning

The Ideal/Real paradigm was first proposed by Goldreich, Micali, and Widgerson in their work on Secure Multiparty Computation (SMC) [3, 4], and further developed by Canetti in his universal composability (UC) framework¹⁶. The basic idea is to treat a cryptographic computation among parties as if it were being carried out by a trusted third party (the “ideal”), and then prove that the actual implementation (the “real”) emulates this ideal, in

⁷ <http://haslab.uminho.pt/mbb/software/cao-domain-specific-language-cryptography>

⁸ <https://galois.com/project/cryptol/>

⁹ <https://eprint.iacr.org/2014/774>

¹⁰ <http://prosecco.gforge.inria.fr/personal/bblanche/proverif/>

¹¹ <http://prosecco.gforge.inria.fr/personal/bblanche/cryptoverif/>

¹² <https://www.easycrypt.info/trac/>

¹³ <http://certicrypt.gforge.inria.fr/>

¹⁴ <http://frama-c.com/>

¹⁵ <http://research.microsoft.com/en-us/projects/fstar/>

¹⁶ <https://eprint.iacr.org/2000/067.pdf>

that the parties can learn nothing more than they would in a protocol involving a trusted party. (The paradigm also handles correctness, robustness, and other emergent properties.)

This is a classic kind of abstraction also present in formal verification: If a program P uses a module M that implements specification S , then relinking P to use M' , which also implements S , should preserve the correct execution of P . One talk, by Alley Stoughton, made the interesting observation that the Real/Ideal notion might be a suitable organizing principle around which to verify software is secure, essentially by using the Ideal as a richer kind of security property than is typical in PL (which often looks at properties like information flow control), and using abstraction in key ways to show it is enforced.

In the Crypto setting, the Real-to-Ideal connection is established probabilistically, considering a diminishing likelihood that a computationally bounded adversary would be able to tell the difference between the Real and Ideal. In the PL setting, the specification-implementation connection is established using methods of formal reasoning and logic, and usually without considering an adversary.

However, a notion of adversary does arise in PL-style reasoning. In particular, an adversary can be expressed as a context $C[\cdot]$ into which we place a computation e of interest that is subject to that adversary; the composition of the two is written $C[e]$. One PL property in this setup with a Crypto connection is contextual equivalence, which states that e and e' are equivalent iff for all contexts C the outcome of running $C[e]$ is the same as running $C[e']$ – e.g., both diverge or evaluate to the same result. In a PL setting this property is often of interest when proving that two different implementations of the same abstract data type have the same semantics (in all contexts). In a security setting we can view the contexts as adversaries, and e and e' as the Real and Ideal.

Another useful property is *full abstraction*.¹⁷ This property was originally introduced to connect an operational semantics to a denotational semantics – the former defines a kind of abstract machine that explains how programs compute, while the latter denotes the meaning of a program directly, in terms of another mathematical formalism (like complete partial orders). Both styles of semantics have different strengths, and full abstraction connects them: it requires that e and e' are observationally equivalent (according to the operational semantics) if and only if they have the same denotation (according to the denotational semantics).

In a Crypto setting, we might view the operational semantics as the Ideal and the denotational semantics as the Real, and full abstraction then states that despite the added observational power of the Real setting, an adversary cannot distinguish any more programs (i.e., learn any additional information) than he could in the Ideal setting. As a recent example of its use, Abadi and Plotkin used full abstraction to reason about the effectiveness of address space randomization. Another recent result is a fully abstract compiler from a type-safe high-level language to Javascript¹⁸; the compiler effectively defines the denotational semantics, and the fact that it is fully abstract means that the added adversarial power that Javascript provides cannot violate the source language's semantics.

Connections between PL and Crypto

The seminar also brought out ways that PL-style reasoning can be connected to Crypto-style reasoning for stronger end-to-end assurance of security. One connection point was at the Real/Ideal boundary. In particular, for privacy-preserving computation (or computation

¹⁷ <http://users.soe.ucsc.edu/~abadi/Papers/paper-csf-long.pdf>

¹⁸ <http://research.microsoft.com/en-us/um/people/nswamy/supp/full-abstraction.html>

preserving some other security property), Crypto-style reasoning can first be used to establish that the Real emulates the Ideal, and then PL-style reasoning can consider the security of the Ideal itself.

For example, consider the setting of SMC. Here, we have two (or more) parties X and Y that wish to compute a function F of their inputs x and y , whereby each party only learns the output $F(x, y)$, but does not “see” the inputs. That is, the security goal is to establish that the Real computation of $F(x, y)$ is indistinguishable from the Ideal model of executing F at a trusted third party. While Crypto can establish that a technique like garbled circuits effectively emulates a trusted third party, it does not establish that the output of F , even when computed by the Ideal, does not reveal too much information. For example, if $F(x, y) = y$ then X learns Y ’s value y directly. More subtly, if $F(x, y) = x > y$, then if $x = 1$, an output of TRUE tells X that Y ’s value $y = 0$. PL-style reasoning can be applied to functions F to establish whether they are sufficiently private, e.g., by using ideas like knowledge-based reasoning¹⁹ or type systems for differential privacy.²⁰ PL-style reasoning about knowledge can also be used to optimize SMCs by identifying places where a transformation would not affect security (e.g., no more is learned by an adversary observing the transformed program), but could improve performance.²¹

Another way to connect PL to Crypto is to factor security-sensitive computations into general-purpose and cryptographic parts. Then PL-style methods can be used to specify the overall computation with the Crypto parts carefully abstracted out. The proof of security then follows a PL approach, assuming guarantees provided by the Crypto parts, which are separately proved using Crypto techniques. In a sense we can think of the PL techniques as employing syntactic/symbolic reasoning, and the Crypto techniques employing computational/probabilistic reasoning.

This is the approach taken in LambdaAuth, a language extension for programming *authenticated data structures* (in the style of Merkle trees), in which the key idea involving the use of cryptographic hashes was abstracted into a language feature, and the proof of security combined a standard PL soundness proof along with a proof of the assumption that hash collisions are computationally difficult to produce. Recent work by Chong and Tromer on proof-carrying data similarly considers a language-level problem and proves useful guarantees by appealing to abstracted cryptographic mechanisms.²² Likewise, work on Memory Trace Obliviousness reasons about Oblivious RAM abstractly/symbolically in a PL setting to prove that the address trace of a particular program leaks no information.²³

Open problems

Beyond work that is being done, one goal of the seminar was to identify possible collaborations on future work. PL researchers and cryptographers work on common problems from different points of view, so one obvious next step is to collaborate on these problems.

One relevant problem is *side channels*. Cryptographers are concerned with side channels in their implementations, e.g., to make sure the time, space, or power consumption during

¹⁹ <http://www.cs.umd.edu/~mwh/papers/mardziel12smc.html>

²⁰ <http://www.cis.upenn.edu/~ahae/papers/dfuzz-popl2013.pdf>

²¹ <http://www.cs.umd.edu/~mwh/papers/rastogi13knowledge.html>

²² <https://eprint.iacr.org/2013/513>

²³ <http://www.cs.umd.edu/~mwh/papers/liu13oblivious.html>

an encryption/decryption operation does not reveal anything about the key. Likewise, PL folk care about side channels expressed at the language level, e.g. work by Andrew Myers' group on timing channels²⁴. Both groups bring a useful perspective.

Another common problem is *code obfuscation*. It was cryptographers that proved that virtual black box (VBB) obfuscation is impossible²⁵, and proposed an alternative indistinguishability-based definition. PL researchers, on the other hand, have looked at language-oriented views of obfuscation effectiveness, e.g., based on abstract interpretation²⁶. Just as the halting problem is undecidable, but practical tools exist that prove termination.²⁷ I believe that there is an opportunity here to find something useful, if not perfect.

Finally, the question of *composability* comes up in both Crypto and PL: Can we take two modules that provide certain guarantees and compose them to create a larger system while still ensuring properties proved about each module individually? Each community has notions for composability that are slightly different, though analogous, as discussed above. Can we make precise connections so as to bring over results from one community to the other? Crypto currencies, exemplified by BitCoin, are an area of exploding interest. An interesting feature about these currencies is that they provide a foundation for fair, secure multiparty computation, as demonstrated by Andrychowicz, Dziembowski, Malinowski, and Mazurek in their best paper at IEEE Security and Privacy 2014 [1, 2]. Could PL-style reasoning be applied to strengthen the guarantees provided by such computations? Cryptographic properties are often proved by making probabilistic statements about a system subject to a computationally bounded adversary. Could program analyses be designed to give probabilistic guarantees, drawing on the connection between adversary and context mentioned above, to thus speak more quantitatively about the chances that a property is true, or not, given the judgment of an analysis? How might random testing, which has proved highly useful in security settings, be reasoned about in a similar way?

Acknowledgements. We would like to thank Jonathan Katz for his initial involvement in organizing the seminar and Matthew Hammer for his help in preparing this report.

References

- 1 Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Łukasz Mazurek. Secure Multiparty Computations on Bitcoin. 2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18–21, 2014, pp. 443–458, IEEE CS. <http://dx.doi.org/10.1109/SP.2014.35>
- 2 Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Łukasz Mazurek. Secure Multiparty Computations on Bitcoin. Cryptology ePrint Archive, Report 2013/784, 2013. <https://eprint.iacr.org/2013/784>
- 3 Oded Goldreich and Ronen Vainish. How to solve any protocol problem – an efficiency improvement (extended abstract). In Carl Pomerance, editor, *Advances in Cryptology – CRYPTO'87*, volume 293 of *Lecture Notes in Computer Science*, pages 73–86. Springer Berlin Heidelberg, 1988.
- 4 Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *J. ACM*, 38(3):690–728, July 1991.

²⁴ <http://www.cs.cornell.edu/andru/papers/pltiming.html>

²⁵ <https://www.iacr.org/archive/crypto2001/21390001.pdf>

²⁶ http://dx.doi.org/10.1007/978-3-642-33125-1_11

²⁷ <http://research.microsoft.com/en-us/projects/t2/>

2 Table of Contents

Executive Summary

<i>Michael Hicks</i>	29
--------------------------------	----

Talk Abstracts

SNARKs on Authenticated Data <i>Manuel Barbosa</i>	37
Introduction to computer-aided cryptographic proofs <i>Gilles Barthe</i>	37
From CryptoVerif Specifications to Computationally Secure Implementations of Protocols <i>Bruno Blanchet</i>	37
A two-level approach for programming secure multi-party computing <i>Dan Bogdanov</i>	38
CBMC-GC: Secure Two-Party Computations in ANSI C <i>Niklas Buescher</i>	39
Enforcing Language Semantics Using Proof-Carrying Data <i>Stephen Chong</i>	39
Secure composition of protocols <i>Veronique Cortier</i>	40
WYSTERIA: A Programming Language for Generic, Mixed-Mode Multiparty Computations <i>Matthew A. Hammer</i>	40
Compiling SQL for encrypted data <i>Florian Kerschbaum</i>	41
Rational Protection Against Timing Attacks <i>Boris Köpf</i>	41
Proving the TLS Handshake Secure (as it is) – and will be <i>Markulf Kohlweiss</i>	42
Automated Analysis and Synthesis of Modes of Operation and Authenticated Encryption Schemes <i>Alex Malozemoff</i>	42
Crash Course on Cryptographic Program Obfuscation <i>Alex Malozemoff</i>	43
A Practical Testing Framework for Isolating Hardware Timing Channels <i>Sarah Meiklejohn</i>	43
Dejà Q: Using Dual Systems to Revisit q-Type Assumptions <i>Sarah Meiklejohn</i>	44
A Computational Model including Timing Attacks <i>Esfandiar Mohammadi</i>	44

Proving the Security of the Mini-APP Private Information Retrieval Protocol in EasyCrypt <i>Alley Stoughton</i>	45
Using the Real/Ideal Paradigm to Define Program Security <i>Alley Stoughton</i>	45
Enforcing Language Semantics Using Proof-Carrying Data <i>Eran Tromer</i>	45
Information leakage via side channels: a brief survey <i>Eran Tromer</i>	46
Verification of Quantum Crypto <i>Dominique Unruh</i>	46
Participants	47

3 Talk Abstracts

3.1 SNARKs on Authenticated Data

Manuel Barbosa (University of Minho – Braga, PT)

License © Creative Commons BY 3.0 Unported license
© Manuel Barbosa

Joint work of Barbosa, Manuel;Backes, Michael;Fiore, Dario; Reischuk, Raphael;

Main reference M. Backes, M. Barbosa, D. Fiore, R. Reischuk, “ADSNARK: Nearly Practical and Privacy-Preserving Proofs on Authenticated Data,” Cryptology ePrint Archive, Report 2014/617, 2014.

URL <http://eprint.iacr.org/2014/617>

Presentation of joint work with M. Backes, D. Fiore and R. Reischuk, available on ePrint. We discuss the problem of privacy-preserving proofs on authenticated data, where a party receives data from a trusted source and is requested to prove computations over the data to third parties in a correct and private way, i.e., the third party learns no information on the data but is still assured that the claimed proof is valid. We formalize the above three-party model, discuss concrete application scenarios, and then we design, build, and evaluate ADSNARK, a nearly practical system for proving arbitrary computations over authenticated data in a privacy-preserving manner. ADSNARK improves significantly over state-of-the-art solutions for this model. For instance, compared to corresponding solutions based on Pinocchio (Oakland’13), ADSNARK achieves up to 25x improvement in proof-computation time and a 20x reduction in prover storage space.

3.2 Introduction to computer-aided cryptographic proofs

Gilles Barthe (IMDEA Software – Madrid, ES)

License © Creative Commons BY 3.0 Unported license
© Gilles Barthe

In this tutorial I will present some recent developments in computer-aided cryptography.

3.3 From CryptoVerif Specifications to Computationally Secure Implementations of Protocols

Bruno Blanchet (INRIA – Paris, FR)

License © Creative Commons BY 3.0 Unported license
© Bruno Blanchet

Joint work of Cadé, David; Blanchet, Bruno

Main reference D. Cadé, B. Blanchet, “Proved Generation of Implementations from Computationally-Secure Protocol Specifications,” in Proc. of the 2nd Int’l Conf. on Principles of Security and Trust (POST’13), LNCS, Vol. 7796, pp. 63–82, Springer, 2013.

URL http://dx.doi.org/10.1007/978-3-642-36830-1_4

This talk presents a novel technique for obtaining implementations of security protocols, proved secure in the computational model. We formally specify the protocol to prove, we prove this specification secure using the computationally-sound protocol verifier CryptoVerif, and we automatically translate it into an implementation in OCaml using a new compiler that we have implemented. We proved that our compiler preserves security. We applied this approach to the SSH Transport Layer protocol: we proved the authentication of the

server and the secrecy of the session keys in this protocol and verified that the generated implementation successfully interacts with OpenSSH. The secrecy of messages sent over the SSH tunnel cannot be proved due to known weaknesses in SSH with CBC-mode encryption.

References

- 1 David Cadé and Bruno Blanchet. From computationally-proved protocol specifications to implementations and application to SSH. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA)*, 4(1):4–31, Mar. 2013.
- 2 David Cadé and Bruno Blanchet. Proved generation of implementations from computationally secure protocol specifications. *Journal of Computer Security*. To appear.

3.4 A two-level approach for programming secure multi-party computing

Dan Bogdanov (*Cybernetica AS – Tartu, EE*)

License © Creative Commons BY 3.0 Unported license
© Dan Bogdanov

Joint work of Bogdanov, Dan; Kerik, Liisi; Laud, Peeter; Pankova, Alisa; Pettai, Martin; Randmets, Jaak; Ristioja, Jaak; Siim, Sander; Tarbe, Karl

Main reference D. Bogdanov, P. Laud, J. Randmets, “Domain-Polymorphic Programming of Privacy-Preserving Applications,” in Proc. of the 1st ACM Workshop on Language Support for Privacy-enhancing Technologies (PETShop’13), pp. 23–26, ACM, 2013.

URL <http://dx.doi.org/10.1145/2517872.2517875>

The implementation of secure multi-party computation applications needs specialized programming tools to hide the complexity of cryptography from the developer. Furthermore, secure multi-party computation seems to fit naturally into shared data analysis. We need tools that keep development simple, while preserving optimization opportunities and allowing formal security analyses.

Our solution is to separate the development into two layers. First, a high-level imperative language is used by the IT system developer to implement the algorithms and business logic. This language is independent of the underlying cryptographic protocols and the number of parties used in the execution. It emits bytecode that is interpreted by a specific virtual machine.

Second, a lower level language is used to implement the atomic secure operations in this virtual machine. This language is used by experts in secure computation to implement the protocols. Thus, it knows about parties, network channels and other necessary primitives. The language can be functional in order to simplify optimization and security analysis.

We have implemented this model in the Sharemind secure multi-party computation system with good results. The high-level language SecreC is used by non-cryptographers to implement real-world applications and it has a standard library of over 25 000 lines of code. For the lower layer, we have two options. Our own protocol DSL is a functional language for implementing protocols based on secret sharing. But we also support protocols generated by the CBMC-GC compiler as Boolean circuits.

References

- 1 Dan Bogdanov, Peeter Laud, Jaak Randmets. *Domain-Polymorphic Programming of Privacy-Preserving Applications*. In Proceedings of the First ACM Workshop on Language Support for Privacy-enhancing Technologies, PETShop 2013, ACM Digital Library. 2013.

- 2 Dan Bogdanov, Peeter Laud, Jaak Randmets. *Specifying Sharemind’s Arithmetic Black Box*. In Proceedings of the First ACM Workshop on Language Support for Privacy-enhancing Technologies, PETShop 2013, ACM Digital Library. 2013.

3.5 CBMC-GC: Secure Two-Party Computations in ANSI C

Niklas Buescher (TU Darmstadt, DE)

License © Creative Commons BY 3.0 Unported license

© Niklas Buescher

Joint work of Franz, Martin; Holzer, Andreas; Katzenbeisser, Stefan; Schallhart, Christian; Veith, Helmut

Main reference M. Franz, A. Holzer, S. Katzenbeisser, C. Schallhart, H. Veith, “CBMC-GC: An ANSI C Compiler for Secure Two-Party Computations,” in Proc. of the 23rd International Conference on Compiler Construction (CC’14), LNCS, Vol. 8409, pp. 244–249, Springer, 2014.

URL http://dx.doi.org/10.1007/978-3-642-54807-9_15

Secure two-party computation (STC) is a computer security paradigm where two parties can jointly evaluate a program with sensitive input data, provided in parts from both parties. By the security guarantees of STC, neither party can learn any information on the other party’s input while performing the STC task. For a long time thought to be impractical, until recently, STC has only been implemented with domain-specific languages or hand-crafted Boolean circuits for specific computations. Our open-source compiler CBMC-GC is the first ANSI-C compiler for STC. It turns C programs into Boolean circuits that fit the requirements of garbled circuits, a generic STC approach based on circuits. Here, the size of the resulting circuits plays a crucial role since each STC step involves encryption and network transfer and is therefore extremely slow when compared to computations performed on modern hardware architectures. We report on newly implemented circuit optimization techniques that substantially reduce the circuit sizes compared to the original release of CBMC-GC.

References

- 1 A. Holzer, M. Franz, S. Katzenbeisser, and H. Veith. Secure Two-Party Computations in ANSI C. In *CCS’12*, 2012.
- 2 M. Franz, A. Holzer, S. Katzenbeisser, C. Schallhart, and H. Veith. CBMC-GC: An ANSI C Compiler for Secure Two-Party Computations, In *CC’14*, 2014

3.6 Enforcing Language Semantics Using Proof-Carrying Data

Stephen Chong (Harvard University – Cambridge, US)

License © Creative Commons BY 3.0 Unported license

© Stephen Chong

Joint work of Chong, Stephen; Tromer, Eran; Vaughan, Jeffrey A.

Main reference S. Chong, E. Tromer, J. A. Vaughan, “Enforcing Language Semantics Using Proof-Carrying Data,” Cryptology ePrint Archive, Report 2013/513, 2013.

URL <http://eprint.iacr.org/2013/513>

Sound reasoning about the behavior of programs relies on program execution adhering to the language semantics. However, in a distributed computation, when a value is sent from one party to another, the receiver faces the question of whether the value is well-traced: could it have been produced by a computation that respects the language semantics? If not, then accepting the non-well-traced value may invalidate the receiver’s reasoning, leading to bugs or vulnerabilities.

Proof-Carrying Data (PCD) is a recently-introduced cryptographic mechanism that allows messages in a distributed computation to be accompanied by proof that the message, and the history leading to it, complies with a specified predicate. Using PCD, a verifier can be convinced that the predicate held throughout the distributed computation, even in the presence of malicious parties, and at a verification cost that is independent of the size of the computation producing the value. Unfortunately, previous approaches to using PCD required tailoring a specialized predicate for each application, using an inconvenient formalism and with little methodological support.

We connect these two threads by introducing a novel, PCD-based approach to enforcing language semantics in distributed computations. We show how to construct an object-oriented language runtime that ensures that objects received from potentially untrusted parties are well-traced with respect to a set of class definitions. Programmers can then soundly reason about program behavior, despite values received from untrusted parties, without needing to be aware of the underlying cryptographic techniques.

3.7 Secure composition of protocols

Veronique Cortier (LORIA – Nancy, FR)

License © Creative Commons BY 3.0 Unported license
© Veronique Cortier

Joint work of Ciobaca, Stefan; Cortier, Veronique; Delaune, Stéphanie; Le Morvan, Éric

Consider your favorite key-exchange protocol. Assume it is secure. Is it possible to use it to implement a secure channel?

In all generality, the answer is no. In this talk, we review techniques for securely composing protocols, for various notions of composition.

3.8 Wysteria: A Programming Language for Generic, Mixed-Mode Multiparty Computations

Matthew A. Hammer (University of Maryland, US)

License © Creative Commons BY 3.0 Unported license
© Matthew A. Hammer

Joint work of Rastogi, Aseem; Hammer, Matthew A.; Hicks, Michael

Main reference A. Rastogi, M. A. Hammer, M. Hicks, “Wysteria: A Programming Language for Generic, Mixed-Mode Multiparty Computations,” in Proc. of the 2014 IEEE Symposium on Security and Privacy (SP’14), pp. 655–670, IEEE, 2014.

URL <http://dx.doi.org/10.1109/SP.2014.48>

URL <https://bitbucket.org/aseemr/wysteria/>

In a Secure Multiparty Computation (SMC), mutually distrusting parties use cryptographic techniques to cooperatively compute over their private data; in the process each party learns only explicitly revealed outputs. In this paper, we present WYSTERIA, a high-level programming language for writing SMCs. As with past languages, like Fairplay, WYSTERIA compiles secure computations to circuits that are executed by an underlying engine. Unlike past work, WYSTERIA provides support for mixed-mode programs, which combine local, private computations with synchronous SMCs. WYSTERIA complements a standard feature set with built-in support for secret shares and with wire bundles, a new abstraction that supports generic n-party computations. We have formalized WYSTERIA, its refinement

type system, and its operational semantics. We show that WYSTERIA programs have an easy-to-understand single-threaded interpretation and prove that this view corresponds to the actual multi-threaded semantics. We also prove type soundness, a property we show has security ramifications, namely that information about one party's data can only be revealed to another via (agreed upon) secure computations. We have implemented WYSTERIA, and used it to program a variety of interesting SMC protocols from the literature, as well as several new ones. We find that WYSTERIA's performance is competitive with prior approaches while making programming far easier, and more trustworthy.

3.9 Compiling SQL for encrypted data

Florian Kerschbaum (SAP AG – Karlsruhe, DE)

License © Creative Commons BY 3.0 Unported license
 © Florian Kerschbaum
Joint work of Kerschbaum, Florian; Härterich, Martin; Kohler, Mathias; Hang, Isabelle; Schaad, Andreas; Schröpfer, Axel; Tighzert, Walter
Main reference F. Kerschbaum, M. Härterich, M. Kohler, I. Hang, A. Schaad, A. Schröpfer, W. Tighzert, “An Encrypted In-Memory Column-Store: The Onion Selection Problem,” in Proc. of the 9th Int'l Conf. on Information Systems Security (ICISS'13), LNCS, Vol. 8303, pp. 14–26, Springer, 2013.
URL http://dx.doi.org/10.1007/978-3-642-45204-8_2

We present a problem in processing SQL over encrypted data. Encrypted database enable outsourcing to the cloud, but require adapting the encryption scheme to the SQL operation. A problem arises when one operator requires a different encryption scheme than its predecessor. Most notably sorting of (or range queries on) homomorphically encrypted data is not possible. A query like “SELECT TOP 3 zipcode GROUP BY zipcode ORDER BY SUM(revenue)” cannot be performed on encrypted data. The solution to this problem is deeper query analysis and compilation. We build the operator tree (relational algebra) of the SQL query, split it at the bottom most conflict and execute one part on the database and one part on the client. This implies a performance penalty for transferring more data to the client for some queries and always for query analysis, but enables full SQL functionality and policy configuration of the encryption (and hence potentially increasing security).

References

- 1 Florian Kerschbaum, Martin Härterich, Mathias Kohler, Isabelle Hang, Andreas Schaad, Axel Schröpfer, and Walter Tighzert. An Encrypted In-Memory Column-Store: The Onion Selection Problem. In *Proceedings of the 9th International Conference on Information Systems Security (ICISS)*, 2013.

3.10 Rational Protection Against Timing Attacks

Boris Köpf (IMDEA Software – Madrid, ES)

License © Creative Commons BY 3.0 Unported license
 © Boris Köpf

We present a novel approach for reasoning about the trade-off between security and performance in timing attacks, based on techniques from game theory and quantitative information-flow analysis. Our motivating example is the combination of input blinding and discretization

of execution times, for which the trade-off between security and performance can be cast formally.

We put our techniques to work in a case study in which we identify optimal countermeasure configurations for the OpenSSL RSA implementation. We determine situations in which the optimal choice is to use a defensive, constant-time implementation and a small key, and situations in which the optimal choice is a more aggressively tuned (but leaky) implementation with a longer key.

References

- 1 Goran Doychev and Boris Köpf. Rational protection against timing attacks, 2015.
- 2 Boris Köpf and Markus Dürmuth. A provably secure and efficient countermeasure against timing attacks. In *CSF*, pages 324–335. IEEE, 2009.

3.11 Proving the TLS Handshake Secure (as it is) – and will be

Markulf Kohlweiss (Microsoft Research UK – Cambridge, GB)

License © Creative Commons BY 3.0 Unported license
© Markulf Kohlweiss

Joint work of M. Kohlweiss, K. Bhargavan, A. Delignat-Lavaud, C. Fournet, A. Pironti, P-Y. Strub, S. Zanella-Béguélin

Main reference K. Bhargavan, C. Fournet, M. Kohlweiss, A. Pironti, P-Y. Strub, S. Zanella Béguélin, “Proving the TLS Handshake Secure (As It Is),” in Proc. of the 4th Annual Cryptology Conference – Advances in Cryptology – Part II (CRYPTO’14), LNCS, Vol. 8617, pp. 235–255, Springer, 2014.

URL http://dx.doi.org/10.1007/978-3-662-44381-1_14
URL <http://www.mitls.org>

The TLS Internet Standard features a mixed bag of cryptographic algorithms and constructions, letting clients and servers negotiate their use for each run of the handshake.

I present an analysis of the provable security of the TLS handshake, as it is implemented and deployed. To capture the details of the standard and its main extensions, it relies on miTLS, a verified reference implementation of the protocol. This motivates the use of new agile security definitions and assumptions for the signatures, key encapsulation mechanisms (KEM), and key derivation algorithms used by the TLS handshake. To validate the model of key encapsulation, the analysis shows that the KEM definition is satisfied by RSA ciphersuites under the plausible assumption that PKCS#1v1.5 ciphertexts are hard to re-randomize.

I also touch on the need to adapt our KEM model to support the recent session hash and extended master secret draft TLS extension that binds TLS master secrets to the context in which they were generated.

3.12 Automated Analysis and Synthesis of Modes of Operation and Authenticated Encryption Schemes

Alex Malozemoff (University of Maryland, US)

License © Creative Commons BY 3.0 Unported license
© Alex Malozemoff

Joint work of Malozemoff, Alex; Green, Matthew; Katz, Jonathan

Main reference A. J. Malozemoff, J. Katz, M.D. Green, “Automated Analysis and Synthesis of Block-Cipher Modes of Operation,” Cryptology ePrint Archive: Report 2014/774, 2014.

URL <https://eprint.iacr.org/2014/774>

In this talk, we present two approaches to synthesizing encryption schemes. We first discuss a work published at CSF 2014, where we synthesize block-cipher modes of operations, which

are mechanisms for probabilistic encryption of arbitrary length messages using any underlying block cipher. We propose an automated approach for the security analysis of block-cipher modes of operation based on a "local" analysis of the steps carried out by the mode when handling a single message block. We model these steps as a directed, acyclic graph, with nodes corresponding to instructions and edges corresponding to intermediate values. We then introduce a set of labels and constraints on the edges, and prove a meta-theorem showing that any mode for which there exists a labeling of the edges satisfying these constraints is secure (against chosen-plaintext attacks). This allows us to reduce security of a given mode to a constraint-satisfaction problem, which in turn can be handled using an SMT solver. We couple our security-analysis tool with a routine that automatically generates viable modes; together, these allow us to synthesize hundreds of secure modes.

In the second part of the talk, we discuss recent work extending this approach to authenticated encryption schemes, which both encrypts and authenticates arbitrary-length messages using a block-cipher as a building block.

3.13 Crash Course on Cryptographic Program Obfuscation

Alex Malozemoff (University of Maryland, US)

License © Creative Commons BY 3.0 Unported license
© Alex Malozemoff

Joint work of Apon, Daniel; Huang, Yan; Katz, Jonathan; Malozemoff, Alex

Main reference D. Apon, Y. Huang, J. Katz, A. J. Malozemoff, "Implementing Cryptographic Program Obfuscation," Cryptology ePrint Archive, Report 2014/779, 2014.

URL <https://eprint.iacr.org/2014/779>

In this talk, we give a brief overview of cryptographic program obfuscation, discussing the definitions, a high-level description of the main construction, and some performance results.

3.14 A Practical Testing Framework for Isolating Hardware Timing Channels

Sarah Meiklejohn (University College London, GB)

License © Creative Commons BY 3.0 Unported license
© Sarah Meiklejohn


This work identifies a new formal basis for hardware information flow security by providing a method to separate timing flows from other flows of information. By developing a framework for identifying these different classes of information flow at the gate level, one can either confirm or rule out the existence of such flows in a provable manner.

References

- 1 Jason Oberg, Sarah Meiklejohn, Timothy Sherwood, and Ryan Kastner. A practical testing framework for isolating hardware timing channels. In *Proceedings of the Conference on Design, Automation and Test in Europe, DATE'13*, pages 1281–1284, 2013.

3.15 Dejà Q: Using Dual Systems to Revisit q-Type Assumptions

Sarah Meiklejohn (University College London, GB)

License  Creative Commons BY 3.0 Unported license
© Sarah Meiklejohn


After more than a decade of usage, bilinear groups have established their place in the cryptographic canon by enabling the construction of many advanced cryptographic primitives. Unfortunately, this explosion in functionality has been accompanied by an analogous growth in the complexity of the assumptions used to prove security. Many of these assumptions have been gathered under the umbrella of the “uber-assumption,” yet certain classes of these assumptions—namely, q-type assumptions—are stronger and require larger parameter sizes than their static counterparts. In this paper, we show that in certain groups, many classes of q-type assumptions are in fact implied by subgroup hiding (a well-established, static assumption). Our main tool in this endeavor is the dual-system technique, as introduced by Waters in 2009. We show that q-type assumptions are implied—when instantiated in appropriate groups—solely by subgroup hiding.

References

- 1 Melissa Chase and Sarah Meiklejohn. Dejà Q: Using dual systems to revisit q-type assumptions. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 622–639, 2014.

3.16 A Computational Model including Timing Attacks

Esfandiar Mohammadi (Universität des Saarlandes, DE)

License  Creative Commons BY 3.0 Unported license
© Esfandiar Mohammadi

Joint work of Backes, Michael; Manoharan, Praveen; Mohammadi, Esfandiar
Main reference M. Backes, P. Manoharan, E. Mohammadi, “TUC: Time-sensitive and Modular Analysis of Anonymous Communication,” in Proc. of the of the 27th IEEE Computer Security Foundations Symp. (CSF’14), pp. 383–397, IEEE, 2014.
URL <http://dx.doi.org/10.1109/CSF.2014.34>

Cryptographic proofs about security protocols typically abstract from timing attacks. For some security protocols, however, timing attacks constitute the most effective class of attacks, such as the anonymous communication protocol Tor. We present TUC (for Time-sensitive Universal Composability): the first universal composability framework that includes a comprehensive notion of time. TUC, in particular, includes network-based timing attacks against multi-party protocols (e.g., Tor). We, furthermore, discuss how system-level can be modelled in TUC.

3.17 Proving the Security of the Mini-APP Private Information Retrieval Protocol in EasyCrypt

Alley Stoughton (MIT Lincoln Laboratory – Lexington, US)

License © Creative Commons BY 3.0 Unported license
© Alley Stoughton

Joint work of Stoughton, Alley; Herzog, Jonathan; Varia, Mayank

Mini-APP is a simple private information retrieval (PIR) protocol involving a very simple kind of database. It's my simplification of a PIR protocol developed by cryptographers at the University of California, Irvine, as part of IARPA's APP (Advanced Privacy Protection) program. I will describe the Mini-APP protocol, define its security using the real/ideal paradigm, and give a high level explanation of how I proved its security using the EasyCrypt proof assistant.

3.18 Using the Real/Ideal Paradigm to Define Program Security

Alley Stoughton (MIT Lincoln Laboratory – Lexington, US)

License © Creative Commons BY 3.0 Unported license
© Alley Stoughton

Joint work of Stoughton, Alley; Johnson, Andrew; Beller, Samuel; Chadha, Karishma; Chen, Dennis; Foner, Kenneth; Zhivich, Michael

Main reference A. Stoughton, A. Johnson, S. Beller, K. Chadha, D. Chen, K. Foner, M. Zhivich, "You Sank My Battleship!: A Case Study in Secure Programming," in Proc. of the 9th Workshop on Programming Languages and Analysis for Security (PLAS'14), pp. 2:2–2:14, ACM, 2014.

URL <http://dx.doi.org/10.1145/2637113.2637115>

I present an example of how the real/ideal paradigm of theoretical cryptography can be used as a framework for defining the security of programs that don't necessarily involve any cryptographic operations, and in which security is enforced using programming language abstractions. Our example is the two player board game Battleship, and we'll consider an implementation of Battleship in the concurrent functional programming language Concurrent ML, giving an informal argument as to why its players are secure against possibly malicious opponents.

3.19 Enforcing Language Semantics Using Proof-Carrying Data

Eran Tromer (Tel Aviv University, IL)

License © Creative Commons BY 3.0 Unported license
© Eran Tromer

Joint work of Chong; Stephen; Tromer, Eran; Vaughan, Jeffrey A.

Main reference S. Chong, E. Tromer, J. A. Vaughan, "Enforcing Language Semantics Using Proof-Carrying Data," Cryptology ePrint Archive, Report 2013/513, 2013.

URL <https://eprint.iacr.org/2013/513>


Sound reasoning about the behavior of programs relies on program execution adhering to the language semantics. However, in a distributed computation, when a value is sent from one party to another, the receiver faces the question of whether the value is well-traced: could it have been produced by a computation that respects the language semantics? If not, then accepting the non-well-traced value may invalidate the receiver's reasoning, leading to bugs or vulnerabilities.

Proof-Carrying Data (PCD) is a recently-introduced cryptographic mechanism that allows messages in a distributed computation to be accompanied by proof that the message, and the history leading to it, complies with a specified predicate. Using PCD, a verifier can be convinced that the predicate held throughout the distributed computation, even in the presence of malicious parties, and at a verification cost that is independent of the size of the computation producing the value. Unfortunately, previous approaches to using PCD required tailoring a specialized predicate for each application, using an inconvenient formalism and with little methodological support.

We connect these two threads by introducing a novel, PCD-based approach to enforcing language semantics in distributed computations. We show how to construct an object-oriented language runtime that ensures that objects received from potentially untrusted parties are well-traced with respect to a set of class definitions. Programmers can then soundly reason about program behavior, despite values received from untrusted parties, without needing to be aware of the underlying cryptographic techniques.

3.20 Information leakage via side channels: a brief survey


Eran Tromer (Tel Aviv University, IL)

License  Creative Commons BY 3.0 Unported license
© Eran Tromer

Security of modern computer systems relies on the ability to enforce separation between mutually-untrusting processes or virtual machines. The communication between these processes/VMs is supposedly controlled by the platform (OS, VMM and hardware) according to a policy. Alas, information flow is a fickle thing: subtle and unexpected interaction between processes through the underlying hardware can convey information, and thereby violate enforcement of separation. Such “side channels” have long been the bane of secure system partitioning. In recent years, they have been recognized as especially dangerous in the age of multitenancy in cloud computing. Analogous challenges arise for corruption of computation and data by induced faults. This talk briefly surveys the challenge, and approaches to mitigating such attacks at the levels of engineering, algorithms, software and program analysis.

3.21 Verification of Quantum Crypto

Dominique Unruh (University of Tartu, EE)

License  Creative Commons BY 3.0 Unported license
© Dominique Unruh

We discussed the challenge of verifying post-quantum secure cryptography, and argue that support for such verification might be achievable at little extra cost in tools like EasyCrypt, both for the implementer of the tool, as well as for the user who writes the proof.

Follow-up discussions have already led to active and fruitful collaboration with the developers of EasyCrypt (specifically Gilles Barthe, François Dupressoir, Pierre-Yves Strub) on this topic.

Participants

- Joseph Ayo Akinyele
Johns Hopkins University –
Baltimore, US
- David Archer
Galois – Portland, US
- Manuel Barbosa
University of Minho – Braga, PT
- Gilles Barthe
IMDEA Software – Madrid, ES
- Karthikeyan Bhargavan
INRIA – Paris, FR
- Bruno Blanchet
INRIA – Paris, FR
- Dan Bogdanov
Cybernetica AS – Tartu, EE
- Niklas Büscher
TU Darmstadt, DE
- Stephen Chong
Harvard University, US
- Véronique Cortier
LORIA – Nancy, FR
- Francois Dupressoir
IMDEA Software – Madrid, ES
- Cédric Fournet
Microsoft Research UK –
Cambridge, GB
- Matthew Hammer
University of Maryland, US
- Michael Hicks
University of Maryland, US
- Catalin Hritcu
INRIA – Paris, FR
- Stefan Katzenbeisser
TU Darmstadt, DE
- Florian Kerschbaum
SAP AG – Karlsruhe, DE
- Boris Köpf
IMDEA Software – Madrid, ES
- Markulf Kohlweiss
Microsoft Research UK –
Cambridge, GB
- Sven Laur
University of Tartu, EE
- Alex Malozemoff
University of Maryland, US
- Sarah Meiklejohn
University College London, GB
- Esfandiar Mohammadi
Universität des Saarlandes, DE
- Axel Schröpfer
SAP AG – Walldorf, DE
- Alley Stoughton
MIT Lincoln Laboratory –
Lexington, US
- Eran Tromer
Tel Aviv University, IL
- Dominique Unruh
University of Tartu, EE
- Santiago Zanella-Béguelin
INRIA – Paris, FR

