Modular Cryptographic Verification by Typing

Cédric Fournet
Microsoft Research

Markulf Kohlweiss
Microsoft Research

Modularity in programming through the use of well defined interfaces serves the same goals as secure composition in cryptography. The intuition is that modules that export the same programming interface, as well as modules that implement the same cryptographic functionality, should be interchangeable with respect to the properties of the rest of the system, both in a programming and in a cryptographic sense.

We develop a cryptographic verification method based on modular programming and typechecking. We use a composition theorem in the simulation-based security flavor to reason about mutually-replaceable modules in a cryptographic sense. We use refinement types to reason about the preservation of properties as we replace modules behind typed interfaces.

Protocol modularity To reason about large software systems that rely on cryptographic algorithms, we need modularity both for programming and for cryptography.

We first reason cryptographically about an ideal functionality that is emulated by a particular cryptographic module, much as with universal composability or reactive simulatability [6, 1]. We then replace the real module of the protocol by an ideal module. We repeat this until all modules are ideal. The ideal modules can be shown to export interfaces that allow us to reason about the security properties of the whole program. In a second step, we check the security of the whole program purely using typing, or other static program analyses, adapted from symbolic techniques for protocol verification [3, 5].

We illustrate our approach on variants of the code of a sample protocol for authenticating remote procedure calls (RPC) adapted from Bhargavan et al. [5], using the refinement type checker F7 [4] and a library that emulates an ideal MAC functionality. In the protocol, a client and a server exchange requests s and responses t coded as strings. To authenticate and correlate these strings, the protocol uses MACs computed from a shared key k. We show how to separate the cryptographic treatment of MACs using unforgeability under chosen-message attacks and the protocol verification using refinement types for authentication.

Cryptographic modularity Computational soundness results for symbolic cryptography can be seen as a special case of modular verification. In that case, a global ideal functionality used for expressing soundness, e.g. [2], consists of a complete Dolev-Yao library [7] where all cryptographic computations are replaced by algebraic terms, all given the same abstract type for bytes. In other cases, it is preferable to consider separate, simpler ideal functionalities that remain closer to real cryptographic algorithms, such as functionalities that still operate on concrete bytes but exclude cryptographic failures.

Modularity is usually subject to conditions, depending on the underlying cryptography. Type interfaces with refinement types are a good match for defining these restrictions, in a way that can be automatically enforced for protocol verification. Refinement types go both ways; they define post-conditions on the results of cryptographic operations, e.g. that a MAC cannot be forged, but also allow us to describe the pre-conditions on these operations that are required to use them securely. Conditions could express important restrictions on the adversary, for instance that a CCA secure encryption scheme provides security only if its interface excludes any environment that may create an encryption cycle [8]. Conditions could also be intended to 'honest' users in the environment, for instance that they only use a key with its intended primitive, and never reveal it to the adversary. Another condition may be that honest users do not decrypt any ciphertext that is not known to be a correct encryption using the same key. Irrespective of our interpretation of these conditions, as actual restrictions on a malicious adversary or as contracts towards programmers, we can systematically express and typecheck them.

We illustrate our approach by extending the RPC protocol outlined above with sample key exchange mechanism that establish sessions-specific MAC keys using long-term secrets. We show how the key exchange can be separately verified then composed with the RPC protocol.

References

[1] M. Backes, B. Pfitzmann, and M. Waidner. The reactive simulatability (rsim) framework for asynchronous systems. *Inf. Comput.*, 205(12):1685–1720, 2007.

[2] M. Backes, M. Maffei, and D. Unruh. Computational sound verification of source code. In *ACM Conference on Computer and Communications Security*, Oct. 2010.

[3] J. Bengtson, K. Bhargavan, C. Fournet, A. D. Gordon, and S. Maffeis. Refinement types for secure implementations. In *21st IEEE Computer Security Foundations Symposium (CSF'08)*, pages 17–32, 2008.

[4] K. Bhargavan, C. Fournet, and A. D. Gordon. F7: refinement types for F#, Sept. 2008. Available from http://research.microsoft.com/F7/.

[5] K. Bhargavan, C. Fournet, and A. D. Gordon. Modular verification of security protocol code by typing. In *ACM Symposium on Principles of Programming Languages (POPL'10)*, Jan. 2010.

[6] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145, 2001.

[7] D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, IT–29(2):198–208, 1983.

[8] R. Küsters and M. Tuengerthal. Universally composable symmetric encryption. In *CSF*, pages 293–307. IEEE Computer Society, 2009. ISBN 978-0-7695-3712-2.