# Cryptographic Security Verification for C by Symbolic Execution

Mihhail Aizatulin[1,2] (avatar@hot.ee) Andrew Gordon[2]
Jan Jürjens[3]

[1]The Open Univerisity [2]Microsoft Research Cambridge [3]University of Dortmund

We present a method for verification of cryptographic protocol implementations. The method starts from the C source code of a protocol and aims to prove secrecy and authentication properties fully automatically, without relying on a formalised protocol specification. We have implemented our approach and successfully tested it on a simple implementation (about 600 lines of code). Our goal is to scale to implementations like OpenSSL or Kerberos.

Our work is motivated by the desire to minimise the gap between verified and executable code. Very little has been done in this area for low-level languages like C. There are numerous tools to find low-level bugs in code (such as buffer overflows and zero division) and there are verifiers for cryptographic protocols that work on fairly abstract descriptions, but so far very few attempts have been done to verify cryptographic security directly from low-level code, with notable exceptions of [2] and [3].

We verify the protocol code in two steps. First we use symbolic execution to extract the formats of messages that the implementation generates together with cryptographic checks that it performs. A typical message format expression inferred by our algorithm would look like

```
write(payload_len|7c|payload|
      HMAC(sha1, i7|7c|52657175657374|payload, key)),
```

where | denotes string concatenation. As a second step we perform some further abstractions that get rid of bitwise concatenation and parsing constructs, so that the above summary reduces to

$$\texttt{write}(f_1(\texttt{payload}, \texttt{hash}(f_1(c_{52657175657374}, \texttt{payload}), \texttt{key})).$$

This is a level of abstraction that can be handled by existing high-level cryptographic verification tools. We use ProVerif [1] to finish verification of the protocol.

The biggest drawback of our current symbolic execution implementation is that it only follows a single path in the protocol execution. This is enough to produce an accurate model when there is only one main path, whereas libraries like OpenSSL contain multiple nontrivial paths. Thus, our main direction for future work is to add support for nontrivial control flow.

# References


[1] Bruno Blanchet. Automatic verification of correspondences for security protocols. *Journal of Computer Security*, 17(4):363–434, 2009.

[2] Sagar Chaki and Anupam Datta. Aspier: An automated framework for verifying security protocol implementations. Technical Report 08-012, Carnegie Mellon University, October 2008.

[3] J. Goubault-Larrecq and F. Parrennes. Cryptographic protocol analysis on real C code. In *Proceedings of the 6th International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI'05)*, volume 3385 of *Lecture Notes in Computer Science*, pages 363–379. Springer, 2005.