Applying Runtime Verification to Group Key Establishment*

Christian Colombo[1], Maria Isabel Gonzalez Vasco[2], Mark Vella[1], and Pavol Zajac[3]

[1] University of Malta
[2] Universidad Rey Juan Carlos
[3] Slovak University of Technology in Bratislava

1 Introduction

Securing communication networks relies fundamentally on the availability of secure cryptographic primitives, which enable authentication, confidentiality, and integrity. A typical starting point to establish secure communication channels is the execution of a protocol that authenticates all involved users/devices and establishes a common secret key among them. This task is referred to as *authenticated group key establishment* (AGKE).

It is standard cryptographic practice to establish provable security guarantees in a suitable theoretical model, abstracting from implementation details. Modeling parties as Turing machines and adversarial capabilities through abstract oracles is a common technique that facilitates a theoretical analysis, but the implementation of a "theoretically secure" protocol may well be susceptible to runtime attacks. A number of authors have explored the use of runtime verification as a means of hardening a security-critical protocol implementation [2, 10, 7, 8]. The aim of this presentation is to give a brief overview of these works.

2 Runtime Verification for Security Protocols

Runtime verification [4, 3] involves the observation of a software system — usually through some form of instrumentation — to assert whether the specification is being adhered to. There are several levels at which this can be done:

Low level At a low level, runtime verification can be used to check software elements which are not specific only to protocol implementations. Rather, such checks would be useful in the context of any application where security is paramount. For example, Signoles et al. [9] provide a platform for C programs, Frama-C, which can automatically check for a wide range of undefined behaviours such as arithmetic overflows, undefined downcasts, and invalid pointer references. More specific to cryptographic algorithms, Secure Flow [1] has been built on top of Frama-C to protect against control-flow based timing attacks by monitoring information flow labels for all values of interest. The platform can also be used as a platform for checking higher level properties mentioned next.

High level At the highest level of abstraction, a couple of approaches [2, 10, 7, 8] check properties directly derived from the protocol design (which would have been checked through the security model). This approach ensures that even though the protocol would have been theoretically verified, the implementation does not diverge from the intended behaviour due to bugs or attacks.

* This work is supported by NATO project *Secure Communication in the Quantum Era* SPS G5448. This project aims to propose and implement a Group Key Establishment Protocol in a post-quantum setting, i. e., a purely classical implementation that is designed to withstand an attacker with access to a large-scale quantum computer.

An example of a temporal property in this category taken from SSL protocol verification [2] is *either no data is sent by the client or the server hash is verified to match the client's version before any data is sent*. This can be expressed in several formalisms. The one chosen in this case is LTL [5], which is a commonly used specification language in the runtime verification community.

In-between An alternative which seems to be lacking is to operate at the medium level of abstraction where the monitoring is aimed specifically to protect the security protocol from targeted attacks. While the consequence of such attacks might lead to the violation of high level properties of the protocol, if well planned such attacks — even if successful — might go unnoticed. Regrettably, the body of work at this level is limited

3 Challenges Ahead

We conclude this short abstract by highlighting two main challenges we foresee in the application of runtime verification for AGKE protocols:

Unintended consequences The incorrect implementation of runtime monitor can facilitate a new type of side-channel not present in the system before deploying the monitor (e.g., a monitor response can lead to a new type of reaction attack). The deployment of a runtime monitor thus need to be considered in the whole security context.

Hybrid security analysis Russo and Sabelfeld [6] showed that dynamic analysis, i.e., runtime verification, cannot be as effective as static analysis techniques, that is, unless both techniques are combined. Therefore combining static analysis with runtime verification can be useful not only in limiting runtime overheads, but also as a means of extending the latter's effectiveness.

References

1. G. Barany and J. Signoles. Hybrid information flow analysis for real-world C code. In *Tests and Proofs - 11th International Conference, TAP 2017, Held as Part of STAF 2017, Marburg, Germany, July 19-20, 2017, Proceedings*, pages 23–40, 2017.
2. A. Bauer and J. Jürjens. Runtime verification of cryptographic protocols. *Computers & Security*, 29(3):315–330, 2010.
3. S. Colin and L. Mariani. Run-time verification. In *Model-Based Testing of Reactive Systems*, volume 3472 of *Lecture Notes in Computer Science*, pages 525–555, 2004.
4. M. Leucker and C. Schallhart. A brief account of runtime verification. *The Journal of Logic and Algebraic Programming*, 78(5):293 – 303, 2009.
5. A. Pnueli. The temporal logic of programs. In *Foundations of Computer Science (FOCS)*, pages 46–57. IEEE, 1977.
6. A. Russo and A. Sabelfeld. Dynamic vs. static flow-sensitive security analysis. In *Proceedings of the 23rd IEEE Computer Security Foundations Symposium, CSF 2010, Edinburgh, United Kingdom, July 17-19, 2010*, pages 186–199, 2010.
7. K. Selyunin, S. Jaksic, T. Nguyen, C. Reidl, U. Hafner, E. Bartocci, D. Nickovic, and R. Grosu. Runtime monitoring with recovery of the SENT communication protocol. In *Computer Aided Verification - 29th International Conference, CAV*, pages 336–355, 2017.
8. J. Shi, S. Lahiri, R. Chandra, and G. Challen. Verifi: Model-driven runtime verification framework for wireless protocol implementations. *CoRR*, abs/1808.03406, 2018.
9. J. Signoles, N. Kosmatov, and K. Vorobyov. E-acsl, a runtime verification tool for safety and security of C programs (tool paper). In *RV-CuBES 2017. An International Workshop on Competitions, Usability, Benchmarks, Evaluation, and Standardisation for Runtime Verification Tools, September 15, 2017, Seattle, WA, USA*, pages 164–173, 2017.
10. X. Zhang, W. Feng, J. Wang, and Z. Wang. Defensing the malicious attacks of vehicular network in runtime verification perspective. In *2016 IEEE International Conference on Electronic Information and Communication Technology (ICEICT)*, pages 126–133, Aug 2016.