Lightweight Crypto in Reverse

Abstract

The aim of this thesis is to look at cryptography in a reversible computing environment. Reversibility works on the principle that no information can be destroyed during the computation, so any computation can be traced to the beginning. Also, in a clean setting no information is unintentionally copied. These features have the implication that any residual information is either known (e.g. null-initialized memory stays zero at the end of the computation), or the encryption implementation can be potentially attacked by using the residual information to lower the searched entropy for brute-force decryption.

In an emerging classification of encryption algorithms, we consider *lightweight encryption* algorithms for several reasons: reversible computation has a direct relation to low-power applications, and also the current breed of programming languages is still rather experimental; standard practices have not evolved yet, so preferably simpler algorithms shall be considered first.

Project Description

With the ongoing boom of small devices, often related to terms like "Internet of Things" and "smart home", we see a new requirement in the area of security, particularly cryptography. These devices have too little processing power (e.g. due to small physical sizes or limitations in terms of electrical power consumption) that they cannot run a complex encryption scheme that is normally required by today's standards when, for example, browsing the web; yet, they require at least some level of encryption due to the fact that a large portion of them will be transmitting data wirelessly. These require what is now known as LIGHTWEIGHT CRYPTOGRAPHY - a set of hashing algorithms, ciphers and other that are optimized for as little CPU processing as possible.

An emerging area in computer science is the concept of REVERSIBLE COMPUTING, where programs can be executed in both directions of the program flow, forwards and backwards. Two languages that have existing interpreters are Janus (imperative, C-like) and RFun (functional, Haskell-like). Both languages are still experimental and have not yet established a programming culture or community and their features are still being extended. To give an insight, the most complex algorithms implemented in Janus are either sorting or simple compression algorithms; for RFun, it is likely its self-interpreter. These facts alone make this topic interesting to explore.

However, the combination of cryptography and reversibility gives us the opportunity to look for some side-channel vulnerabilities. There are several of these, which in theory could be prevented in a reversible setting. Reversibility implies that no piece of information is destroyed. Since reversible programs can be executed in both directions, this also means that no information can be created, or added; only data transformation from input to output occurs. This shows a lack of direct information leakage - a one kind of side-channel vulnerability. Other theoretically possible channels are power consumption (requiring the actual reversible hardware), execution time (due to the "data-transformation" style of the implementation, the execution should be more-or-less free of branching) and possibly even electromagnetic radiation patterns (heat and radio) could be more predictable for any input. However, due to the lack of reversible hardware, we can focus only on those side-channels, that are at least somewhat independent of it. Therefore, in this thesis we focus only on the direct information leakage from correct algorithm implementation and execution timing[1]. Work in this area can also help in the future, when reversible hardware is accessible and the hardware-dependent side channels mentioned above can be tested. Furthermore, as a nice bonus of the reversible setting, we get a decryption function just by reversing the program flow on the encryption.

[1]Note that even the same code can take different amounts of time to execute based on the input; such is the experience from complex processors like the x86 architecture. Thus, in this sense we can only analyse the actual code from the perspective of branching, when, e.g., dealing with special cases.

Learning Objectives

At the end of the project I shall be able to:

1. write functionally correct reversible code,
2. analyse difficulties in writing code in reversible languages from programmer's perspective,
3. analyse potential issues when writing security-sensitive code in reversible languages,
4. implement several cryptography-related functions; block ciphers and hash functions,
5. assess the implementation from security perspective, including an analysis of side-channels,
6. and suggest improvements in development of the reversible languages.

References

[1] Alex Biryukov and Léo Paul Perrin. Lightweight cryptography lounge, 2015.

[2] Alex Biryukov and Léo Paul Perrin. State of the art in lightweight symmetric cryptography, 2017.

[3] Michael Kirkedal Thomsen and Holger Bock Axelsen. Interpretation and programming of the reversible functional language rfun. In *Proceedings of the 27th Symposium on the Implementation and Application of Functional Programming Languages*, IFL '15, pages 8:1–8:13, New York, NY, USA, 2015. ACM.

[4] Tetsuo Yokoyama and Robert Glück. A reversible programming language and its invertible self-interpreter. In *Proceedings of the 2007 ACM SIGPLAN Symposium on Partial Evaluation and Semantics-based Program Manipulation*, PEPM '07, pages 144–153, New York, NY, USA, 2007. ACM.

[5] Holger Bock Axelsen and Tetsuo Yokoyama. Programming techniques for reversible comparison sorts. In Xinyu Feng and Sungwoo Park, editors, *Programming Languages and Systems*, pages 407–426, Cham, 2015. Springer International Publishing.