Computationally Sound Proofs of Security for a Key Management API

EXTENDED ABSTRACT

Graham Steel

LSV, INRIA & CNRS & ENS-Cachan

1 Security APIs for Tamper Resistant Devices

Security solutions for information systems are increasingly making use of tamper-resistant cryptographic devices, whether they are smartcards carried by commuters on a mass transit system, or high-throughput Hardware Security Modules in a bank ATM transaction processing processing facility. Over the last few years we have been analysing the key management APIs of such tamper-resistant devices. The design of the API is critical for the security of the system, since it is the API which controls information flow between the data on the tamper resistant device and the hostile outside world. We have shown a number of standard APIs to be susceptible to attacks whereby sensitive keys are revealed in clear [1, 3, 4]. More recently, we have proposed our own generic API and proved security properties for it [2]. We have also proven security properties for a variant of the popular PKCS#11 standard API, albeit with weaker properties than our own design [4]. However, all these proofs are in the symbolic model. Naturally we would like to extend our results to a suitable 'computational' model. We discuss this ongoing work in this extended abstract.

2 Computational Soundness for APIs

While there is only one Dolev-Yao adversary[1], there are many flavours of 'computational' adversary models. Which is the most suitable for API analysis? It is instructive to look at attacks that have been discovered on existing APIs. For example, in our work on the Eracom variant of PKCS#11, we discovered an attack starting from a scenario in which the intruder learns, by some out-of-model means, a session key (i.e. a key for encrypting and decrypting data). By means of some legitimate API calls with carefully chosen parameters, the attacker is able to turn this session key into a key-encrypting key, and then use it to discover the value of all the keys stored in the device. This kind of attack would seems to suggest a computational model capturing adaptive corruption of keys, i.e. corruption during protocol execution. There is existing work in translating results in a Dolev-Yao model to adaptive corruption models [5]. Panjwani has shown that for a Dolev-Yao style trace containing just symmetric keys encrypted under other symmetric keys, with no key cycles, security against adaptive corruptions can be inferred from the Dolev-Yao result. More precisely, the intruder is given an 'oops' command that models the revelation of a key. He can use this any number of times during his interaction with

[1] more or less..

the API. At the end of his interaction, he will have a certain set of keys in his knowledge set, by the usual Dolev-Yao style rules for symbolic encryption and decryption. Some other keys may remain secret. Panjwani's result shows that these keys remain secret (i.e. random) in an equivalent computational adaptive corruption game, with the usual assumptions (semantic security of the encryption scheme, no key cycles). Note that there are some extra restrictions, however. The intruder may not ask for decryptions under keys he doesn't already know, something which most APIs allow in one form or another. He can't mount other 'active' attacks either, where he e.g. creates fake messages to send to the API.

How close are our symbolic proofs to the domain of this result? We consider first the Eracom-based PKCS#11 variant and its symbolic security proof [4, Fig. 6] (this is the patched version, where the corruption-based attack mentioned above is no longer possible). Most of the output consists of keys encrypted with other keys, but we have also encryption and decryption of arbitrary plaintexts by session keys, and we have hashing used in HMACs to add integrity of key attributes to the key wrapping/unwrapping process. This is important as it prevents an attacker from e.g. turning a session key into a wrapping key, which will lead to as attacks. The Panjwani results cover only symmetric key encryption. Additionally, it is trivial to introduce key cycles in the API as it is currently specified.

We have also proven security properties for a Generic API for implementing key management protocols such as those from the Clark-Jacob corpus. This API effectively describes an encryption scheme that gives us the properties that we want, in that it enforces tagging of elements with their type (session key, nonce, public data) and their intended recipient (which enables us to prove novel properties about secrecy of keys shared between honest users even when the protocol is run between corrupted host machines). This API does not allow long-term keys to be updated, but it does prevent key cycles.

This talk will discuss possible ways to being these results closer together.

References

[1] V. Cortier, G. Keighren, and G. Steel. Automatic analysis of the security of XOR-based key management schemes. In O. Grumberg and M. Huth, editors, *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, number 4424 in LNCS, pages 538–552, 2007.

[2] V. Cortier and G. Steel. Synthesising secure APIs. Technical Report RR-6882, INRIA, 2009. 27 pages.

[3] S. Delaune, S. Kremer, and G. Steel. Formal analysis of PKCS#11. In *Proceedings of the 21st IEEE Computer Security Foundations Symposium (CSF'08)*, pages 331–344, Pittsburgh, PA, USA, June 2008. IEEE Computer Society Press.

[4] S. Fröschle and G. Steel. Analysing PKCS#11 key management APIs with unbounded fresh data. In *To appear in Proceedings of ARSPA-WITS'09*, 2009.

[5] Saurabh Panjwani. Tackling adaptive corruptions in multicast encryption protocols. In *Theory of Cryptography Conference*, 2007.