On the Security of the EMV Secure Messaging API (Extended Abstract)

Ben Adida, Mike Bond, Jolyon Clulow, Amerson Lin,
Ross Anderson, and Ronald L. Rivest

We present new attacks against the EMV financial transaction security system (known in Europe as "Chip and PIN"), specifically on the back-end API support for sending secure messages to EMV smartcards.

EMV is the new electronic payment system designed by VISA and Mastercard and widely deployed throughout Europe in the last 12 months. It aims to eventually supersede magnetic-stripe systems. Customers carry smartcards which, upon payment, engage in cryptographic protocols to authenticate themselves to their issuing bank. At the issuing bank (the "back end" for short), the Hardware Security Modules (HSMs), which are tasked with PIN storage and verification for ATM networks, have been extended to provide new EMV security functionality. The HSMs now authenticate and manage the massive card base, ensuring security in an environment particularly wary of insider attack.

The back-end HSMs expose a security Application Programming Interface (security API), which the untrusted banking application layer uses to perform cryptographic operations, and which enforces a security policy on the usage of the secret data it handles. In the last five years, the security of HSM APIs has come under close scrutiny from the academic community, and, recently, a number of HSM manufacturers have made their EMV functionality available for study.

The new EMV functionality includes three basic classes of commands: firstly, those to verify authorisation requests and produce responses or denials; secondly, those to manage the personalisation of smartcards during the issue process; and thirdly those to produce secure command messages, which are decrypted, verified and executed by smartcards for the purpose of updating security parameters in the field. This paper concentrates on this last class. Such secure messaging commands are used for many purposes: to change the PIN on a card, adjust the offline spending limits, replace cryptographic keys, or toggle international roaming.

We present two attacks, which, together, completely undermine the security of EMV secure messaging, assuming a corrupt insider with access to the HSM API for a brief period. Our first attack allows the injection of chosen plaintext into the encrypted field of a secure message destined for an EMV smartcard (this could be used to update the card's PIN or session key). The second attack discloses any card-exportable secret data, for instance a unique card authentication key. Only one of the two devices, the IBM 4758 CCA, was found vulnerable to this second class of attack, but it is particularly significant, because it is passive with respect to the card. Therefore, if such an attack were used to defraud a bank, it would be much harder to trace. Both attacks exploit the malleability of the CBC mode of operation, combined with an overly generic and extensible security API specification. CBC specifically allows for ciphertext truncation, which, combined with IBM's template-encryption functionality, yields an encryption oracle. This encryption oracle immediately enables adversarial data injections. In addition, using this encryption oracle and the ability in this same HSM to specify the secure-data injection offset, the attacker can split the secret data across a CBC block boundary and perform an offset-shifting attack to discover the contained plaintext one byte at a time.

The other HSM we examined, the Thales RG7000, did not permit injection of any chosen plaintext, so the previous encryption oracle attack does not succeed. However, we expose a different attack based on the lack of key separation between CBC encryption and MAC, permitting the MAC oracle functionality to be warped into an encryption oracle.

These attacks are significant in their impact on the security of the EMV electronic payment system, and submission of this work was significantly delayed in order to give vendors enough time to analyze new versions of their APIs and to fully explore the consequences.

The attacks exploit familiar pitfalls in the usage of the CBC mode of operation, though the HSM environment presents intriguing new variations. Though CBC attacks are not novel, the ease with which our attacks succeed indicates a particularly weak setting for this mode of operation. We find that security APIs are particularly vulnerable because an API adversary is far more powerful than a protocol adversary.

Security modeling of cryptographic primitives often provides the adversary with oracle access to certain functions and asks whether the adversary can use these queries to break a security feature of the primitive. In a number of such models, the queries are adaptively chosen: the result from earlier queries can be used to determined the input to later queries. The burden of such modeling is fairly high, as the adversary is given significant power. In practice, this model has often been ignored, as it seems too far removed from practical considerations.

However, in the case of an HSM or, more generally, a security API, the oracle adversarial model is particularly relevant. The HSM operator may well be the adversary, and the behaviour of a security API maps fairly closely to that of an oracle: the operator can issue large numbers of queries, adapt them according to prior output, and use the gleaned information to break a security property of the system.

In fact, the adversarial model of HSMs strongly resembles the theoretical construct where adversaries are given access to an Oracle. Thus, it is not surprising that cryptographic primitives found to be insecure under such Oracle-access assumptions render HSM APIs so easily vulnerable to attack.

Thus, in the context of defining security APIs, it may be best practice in the future to use cryptographic primitives that are proven secure against such significantly powerful adversaries.

The world of security APIs, where the API provider is often unable to detect fraudulent patterns, is quite vulnerable to attacks on cryptographic primitives, especially when compared with the world of conventional security protocols which mediate between multiple active parties.

While a large number of potential solutions may patch these particular problems, and some of the failures are a direct result of design compromises, the interesting research question is what the longer-term approach should be to ensure API security. One should consider how to correctly trade off between flexibility and atomicity, and whether more systematic, formal, approaches to building secure cryptographic APIs can help.