

Existing and Future Security APIs for Trusted Platform Modules

Ronald Toegl
ronald.toegl@iaik.tugraz.at
IAIK, Graz University of Technology

May 1st, 2008

The Trusted Computing Group (TCG) has specified the Trusted Platform Module (TPM) to extend legacy computer architectures with a hardware-anchor for security. While its functionality resembles that of a smart card, it is in addition physically bound to the device. A tamper resistant casing contains low-level blocks for asymmetric key cryptography, key generation, cryptographic hashing and random number generation. With these components it is designed to keep secret keys protected from any remote attacker. Additional high-level functionality consists of protected non-volatile storage, integrity collection and reporting (attestation), binding of data to a device or a state (sealing), time stamping and identity management.

The TCG architecture not only contains standards for hardware but also for accompanying software components. The TCG Software Stack (TSS) is a highly complex design that provides a security API for the C language. It not only covers the functionality of the TPM but also offers additional services such as external key storage. Split in different layers it exposes several interfaces to both, local and remote applications.

In this talk we will present the currently available TSS specification and in addition our mapping of it into the Java language. From this implementation and the experience gained with it, we conclude that a novel, more compact API is desirable. Instead of full TPM feature coverage, it should rather contain a smaller and practically usable set of high-level functions that represent the core Trusted Computing concepts. We believe that such Trusted Computing APIs, old and new, are a fruitful field for further analysis and research.

We are now leading the standardization of the "Trusted Computing API for Java" in the Java Community Process. The design of this future standard is still in an early phase, and we desire to integrate the results of the security API analysis community in it. This may include approaches that limit the search space for analysis tools or abstaining from the use of advanced language features in Java so that existing formalisms may be employed to model it. Not only will the new Java API and initial implementations be free and openly available, but the open design process itself is offering the chance to make the design amenable to analysis.

Verifying APIs and Protocols Together

Graham Steel

Laboratoire Spécification et Vérification

It has long been the practice when designing cryptographic security protocols to make some assumptions about endpoint security - for example, that ‘Alice’s long term keys remain secret’ or ‘honest players do not reveal their private keys’. However, the increasingly hostile threat scenario of the Internet has called into question the validity of such assumptions about endpoints. Designers are starting to turn to tamper-resistant hardware to try to guarantee some properties of endpoint security. We know that designing the security APIs for these devices is very tricky, and hence the field of API analysis has emerged. One major scientific challenge is to connect the abstract security properties assumed by protocols to properties required of security APIs, in order to prove that an API does (or does not) provide the security guarantees required for the assumptions of the protocols it is designed to implement. A suitable language for expressing these properties is required. New applications will require more fine-grained properties than previous key establishment protocols. For example it may not suffice to say a certain key k must be kept secret: we may also require that it may only be used to encrypt data signed with a certain other key k' - for example, to ensure GPS location data transmitted by a vehicle is correct. The language for properties must be related to the semantics used for the analysis of protocols, so that we can carry out verification of the protocol given some properties of the endpoint.

This talk will describe this problem in more detail and give some tentative details of how it might be solved.

Reducing Protocol Analysis with XOR to the XOR-free Case in the Horn Theory Based Approach

Ralf Küsters and Tomasz Truderung

University of Trier, Germany
{kuesters,truderung}@uni-trier.de

Abstract

In the Horn theory based approach for cryptographic protocol analysis, cryptographic protocols and (Dolev-Yao) intruders are modeled by Horn theories and security analysis boils down to solving the derivation problem for Horn theories. This approach and the tools based on this approach, including ProVerif, have been very successful in the automatic analysis of cryptographic protocols w.r.t. an unbounded number of sessions. However, dealing with the algebraic properties of operators such as the exclusive OR (XOR) has been problematic. In particular, ProVerif cannot deal with XOR.

In this work, we show how to reduce the derivation problem for Horn theories with XOR to the XOR-free case. Our reduction works for an expressive class of Horn theories. A large class of intruder capabilities and protocols that employ the XOR operator can be modeled by these theories. Our reduction allows us to carry out protocol analysis by tools, such as ProVerif, that cannot deal with XOR, but are very efficient in the XOR-free case. We implemented our reduction and, in combination with ProVerif, applied it in the automatic analysis of several protocols that use the XOR operator, including the IBM CCA, for which we found a new attack.

Minimizing Threats from Flawed Security APIs

Mohammad Mannan
Carleton University, Canada
mmannan@scs.carleton.ca

Despite best efforts from security API designers, flaws are often found in widely deployed security APIs. Even APIs with a ‘formal’ proof of security may not guarantee absolute security when used in a real-world device or application. In parallel to spending research efforts to improve security of these APIs, we argue that it may be worthwhile to explore design criteria that would reduce the impact of an API exploit, assuming flaws cannot completely be removed from security APIs. We use such a design philosophy in dealing with PIN cracking attacks on financial PIN processing APIs; several of these attacks have been reported in the last few years, e.g., Berkman and Ostrovsky (FC 2007), Bond (CHES 2001).

One primary reason that PIN cracking attacks are possible is that actual user PINs, although encrypted, travel from ATMs to a verification facility through several (untrustworthy) intermediate switches. If, for example, hashed PINs were sent in an encrypted form, attackers may not be able to reveal real user PINs even in the presence of API flaws. However, as PINs are generally short (4 digits), an offline dictionary attack may still easily allow recovery of actual PINs. Following this direction, we outline a simple solution called *salted-PIN* as follows. A randomly generated salt value of adequate length (e.g. 128-bit) is stored on a bank card in plaintext, and in an encrypted form at a verification facility under a bank-chosen *salt key*. Instead of sending the regular user PIN, salted-PIN requires an ATM to generate a *Transport Final PIN* from a user PIN, account number, and the salt value (stored on the bank card) through, e.g., a pseudo-random function. We explore different attacks on this solution, and propose variants of salted-PIN that can protect against known attacks. Depending on the solution variation, attacks at a malicious intermediate switch now may only reveal the Transport Final PIN; both the user PIN and salt value remain beyond the reach of an attacker’s switch. This work has partly been discussed in a short paper presented in FC 2008.

Analysing protocols of the TPM

Liqun Chen
HP Labs, UK

Mark Ryan
HP Labs, UK, and
University of Birmingham

May 1, 2008

Abstract

The Trusted Platform Module (TPM) is a hardware chip specified by the Trusted Computing Group (TCG) industry consortium. There are 100 million TPMs currently in existence, mostly in high-end laptops. The TPM stores cryptographic keys and other sensitive data in its shielded non-volatile memory. Application software such as Microsoft's BitLocker and HP's HP ProtectTools use the TPM in order to guarantee security properties.

Proof of possession of values known as authData is required by user processes in order to use TPM keys. Because authData may be derived from potentially guessable passwords, the TCG has stipulated in the current version (version 1.2) of the specification that TPM manufacturers should implement resistance to dictionary attacks on authData.

We show that in certain circumstances dictionary attacks can be performed off-line, and therefore the resistance offered by the TPM is compromised. In that way, an attacker can circumvent some crucial operations of the TPM, and impersonate the TPM owner to the TPM, or the TPM to its owner. For example, he can unbind data or migrate keys without possessing the required authorisation data, or fake the creation of TPM keys. This means that any application that relies on the TPM may be vulnerable to attack.

We propose some new solutions and modifications to the TPM specification to prevent the off-line attacks, and we also provide the way to integrate these modifications into the TPM command architecture with minimal change.