Research Workshop on Future TPM Functionality: Final Report

Chris J. Mitchell (editor)

9th September 2006

1 Introduction

The purpose of this short document is to summarise the discussions on future Trusted Platform Module (TPM) functionality held during a Technical Workshop at the First European Summer School on Trusted Infrastructure Technologies. It is hoped that this document will serve as a source of information on possible directions for future research relating to the TPM. It is also hoped that these topics can be discussed further at a future summer school.

The discussions at the Technical Workshop covered the following main topics, and a summary of the points raised on each of these topics is given below.

- Secure boot;
- Certification;
- Privacy;
- TPM command set; and
- TPM command structure.

Finally note that background references for these discussions include [2, 11, 16], as well as, of course, the TCG specifications themselves[1].

2 Secure boot

Secure boot is not currently enabled by the TCG specifications. However, much work on secure boot has been conducted independently of the TCG. Most notably, the concept of secure boot has been discussed by Tygar and Yee [19], Clark [7], Arbaugh, Farber and Smith [1] and Itoi and Arbaugh

[1] www.trustedcomputinggroup.org

[13]. Each of these papers describe a similar process, in which the integrity of system components is measured, and these measurements then compared against a set of expected measurements which must be securely stored and accessed by the platform during the boot process.

Tygar and Yee [19] were amongst the first to describe a secure boot mechanism [1]. They discuss the possibility of using a secure co-processor to facilitate a secure boot. The expected integrity measurements of system components are stored within the secure co-processor non-volatile memory, where their integrity and privacy can be assured. The secure co-processor is first to take control of the system, and it verifies the integrity of system components, for example the boot strap program, the OS kernel and system utilities, before handing over to the host CPU. Tygar and Yee also discuss issues surrounding the use of a secure boot floppy, containing system verification code, rather than using a secure co-processor, which requires significant architectural revisions to most computer systems [1].

Clark and Hoffman [7] present a system in which a PCMCIA card is used to facilitate a secure boot. In this case, the host's boot sector and a series of checksums for boot files and host executables are stored on a PCMCIA card. When the card is inserted into the host, the user is initially authenticated to the card by entering a password. The card is also authenticated to the host after knowledge of a secret shared between the card and the host has been demonstrated. If both authentications are successful, the card allows the host to read the boot sector and any required checksums from the card. When the boot sequence completes, control is given to the operating system, whose configuration has been either retrieved from the PCMCIA card or measured and verified against the expected measurement value stored on the PCMCIA card [7]. The physical security of both the host and the card are assumed.

Arbaugh, David and Smith [1] require the addition of a PROM board and the modification of the system BIOS. Their AEGIS model is based upon four fundamental assumptions. It is assumed that an attacker is unable or unwilling to replace the motherboard, CPU and a portion of the system ROM/BIOS, which contains a small section of trusted software. It is also assumed that an expansion card/PROM board, which contains cryptographic certificates and copies of essential boot process components for recovery, is present. The integrity of this expansion card, called the AEGIS ROM, must also be maintained. It is implied by Arbaugh, David and Smith that the cryptographic certificates contained within the PROM board enable the identities of entities, trusted to certify trustworthy configurations of software components on incoming component certificates, to be verified. These certificates may, for example, take the form of self-signed public key certificates of entities permitted to certify trustworthy configurations of software components. A specific method by which entities are authorised to certify trustworthy configurations of software components is not specified.

Finally, it is assumed that a trusted source exists to support the recovery of platform components, for example a network host or a trusted ROM card located within the host.

Before a secure boot process can be completed the computing platform must be initialised with a number of items (see [1] and [13]).

1. For every component on the platform which requires a secure boot, an authorised entity must generate a hash of that software component and then create a credential which contains the component hash, a component identifier and an expiration date. An authorised entity is one trusted by the system to certify trustworthy configurations of software components. Arbaugh, David and Smith imply that this trust relationship is established through the use of 'cryptographic certificates' installed in the AEGIS ROM. As stated above, details of the trust establishment mechanism are not defined.

2. The credential is digitally signed by the authorised entity.

3. This signed credential is then stored on the host, in the platform component to be securely booted or in a data block of a flash memory device on the host's motherboard.

4. The AEGIS ROM and BIOS boot block contain a small section of trusted software, signed credential(s), authorised entity public key certificates and recovery code, whose integrity is assumed.

The secure boot process is as follows (see [1] and [13]).

1. The first section of the BIOS executes, i.e. the part which contains small section of trusted software, and computes a checksum over its address space and the address space of the AEGIS ROM. This process protects against ROM failures.

2. The hash of the remainder of the BIOS, is then computed.

3. Execution control is then passed to this second section of the BIOS if:

   - Its associated credential has not expired;
   - The signature on the credential is valid;
   - The hash value stored in the credential matches the value computed in step 2.

4. This BIOS component then hashes each of the expansion ROMs and verifies them against their expected values.

5. This hashing and verification continues until the system has been booted into an expected state.

If at any stage during the boot process there is an integrity failure, the failed component is replaced using components either stored on an AEGIS expansion card/PROM board, or retrieved from a trusted network host. Itoi and Arbaugh [13] extend the AEGIS system to work with smartcards.

We now examine suggested methods for secure boot implementation using either a version 1.1 or 1.1b compliant TPM. Versions 1.1 and 1.1b of the TCG TPM specifications define a data integrity register (DIR) to be a storage register that holds a 20 byte digest value. These versions of the TCG specifications require that the TPM contains only one 20-byte DIR in a TPM-shielded location, although the TPM could incorporate more than one DIR. While the exact purpose of DIRs was not specified, their use in the implementation of a secure boot process is briefly examined in [2], and is now described.

If a TPM contains the same number of DIRs as PCRs, the expected value of every PCR can be written to its corresponding DIR. Every time a PCR is filled and its final value computed, it is compared to its corresponding DIR value. If the two values match, the boot process continues; otherwise an exception is called and the boot process halted.

Alternatively, if the TPM has access to non-volatile memory, all expected PCR values may be held in unprotected non-volatile memory, and a summary, i.e. a cumulative digest, is held in a single DIR. Every time a PCR is filled and its final value computed, it is checked that:

1. Each PCR value, when calculated, matches the expected value held in the non-volatile memory; and

2. The cumulative digest of the expected table of PCR values matches the value held in the DIR.

Read access to DIRs must be provided without the need for any authorisation data to be input as, typically, no authorisation information is available at the early stage in the boot process when the DIR value must be read. In the version 1.2 specifications, use of the DIR has been deprecated. The TPM must still, however, support DIR functionality in the general-purpose non-volatile storage area.

Before discussing the concept of secure boot further, a set of use cases, in which a secure boot mechanism may be useful/necessary/advantageous, would be desirable. This would allow us to assess whether the requirement for the addition of secure boot functionality to the TCG specifications is justified.

Secondly, we need to potentially consider a fourth root of trust — a root of trust for verification (RTV). To prevent boot continuing, the RTV has to be able to affect flow of the boot process. What additional properties do you need to provide the RTV function? If the TCG TPM specifications were to support a secure boot mechanism the TPM would be transformed from a

passive to an active component. That is, it would need to affect other PC components — this could cause major issues. Alternatively, a root of trust for verification may be implemented outside of the TPM, as is the case with the RTM. Should the RTV be defined by the TCG?

In order to enable a secure boot process, reference integrity metrics must also be discussed. How should they be certified? What is the infrastructure needed to support this? Does it have to be standardised?

Finally, it is interesting to consider whether the LaGrande Technology (LT) processor extensions [11] make the situation any different. Intel LT provides a kind of secure boot, which bypasses the existed authenticated boot. Does this mean that the current authenticated boot needs revisiting? Of course, this secure boot process is Intel processor-specific, so it would be 'good' if we could extend the idea to other architectures/vendors.

3 Certification

The concept of certification works relatively well when one considers closed systems. A comment was made that the immediate goals of Trusted Computing infrastructures would be centred on closed systems, effectively creating islands of trust. Whilst this may obviate certain shortcomings in developing a PKI infrastructure within a single organisation, things can become significantly more complicated when one considers cross certification. Suddenly issues such as re-parenting, subordination of one CA to another, revocation and re-issuance/replacement and the hierarchy of trust all become major hurdles that need to be addressed. Different CAs and paths have different validity periods and constraints; certificate paths can contain loops and certificate semantics can change on different iterations through the loop.

It is vital to realise that a PKI is not just another IT project; it requires a combined organisational, procedural, and legal approach in which the complexity is enormous. Initial PKI efforts vastly underestimated the amount of work involved [12].

Trusted Computing assumes the presence of Endorsement CAs (the majority of which will probably be manufacturers, but not necessarily all of them), Privacy CAs, including possibly DAA Issuers, and SKAE CAs. In this environment, an SKAE CA is 'reliant' on a Privacy CA, which in turn is 'reliant' on an Endorsement CA. Given some of the requirements of some of the TCG specifications, such as: a Privacy-CA MUST check that the TPM manufacturer, model and version numbers are acceptable [18], it would appear that a close relationship is needed between Endorsement CAs and Privacy CAs.

If the Endorsement CA is not a manufacturer, how are the trust relations affected when a platform wishes to communicate with another entity that

does not exist within their trust island? Additionally, whilst it may be acceptable to say for the time being that entities will evolve to fulfil the roles of the different CAs, it is important that we consider the incentives necessary for someone to do so. To be accepted, a PKI must provide perceived value, lest Trusted Computing heads the way of SET.

4 Privacy

A great deal of TPM functionality is present to keep the privacy advocates happy. Indeed, much of it is present as a result of specific requests relating to privacy. Given the high complexity of TPM, this raises the question 'Are all these mechanisms really necessary?' To put it another way, whilst privacy is clearly/undeniably important, is the current set of privacy-supporting features too heavy?

One widely discussed example of a technology present purely to address privacy concerns is the DAA protocol. It would appear that DAA may not be needed in all environments, e.g. that defined by the Mobile Phone Working Group (MPWG), but it may, nevertheless, be valuable elsewhere. One key question would appear to be 'Do we have convincing real-world use cases where DAA is necessary?' One interesting use case of DAA is where it is used to validate an AIK to a privacy CA; the privacy CA then does not have the ability to link this AIK to a platform or to any other AIKs (since it does not see the EK).

More generally, use cases relevant to all aspects of privacy functionality would be highly desirable.

One other aspect of privacy-related functionality relates to the presence of three switches causing reboot. Is this really necessary?

The point was also made that the potential lack of complete control over the operation of TPM may cause privacy concerns.

Finally, how much is privacy a user perception issue, and how much is it a genuine issue relating to potential use/abuse of Personally Identifying Information (PII)?

5 TPM command set

The TPM has a complex and rich set of commands. This API is, in some ways, rather similar to a Crypto API for a hardware security module. It is well know that such APIs are extremely hard to design from a security perspective, and security vulnerabilities have been discovered in apparently very well-designed such APIs. The interested reader is referred to [3, 4, 5, 6, 8, 9, 14, 15] as well as chapter 15 of [10].

This raises the obvious question: is the TPM API cryptographically sound? That is, are there sequences of command calls which can be used to:

- Learn information about the secrets (keys or data) stored within the TPM;
- Make unauthorised modifications to data stored within the PM, and/or to modify data stored externally to the TOM but protected by it;
- Produce unauthorised signatures or MACs on data strings; and/or
- In general, perform any function which breaks the security policy of the TPM.

It would appear that some work has already been done both to detect vulnerabilities and to try to verify selected parts of the command set (including the 'get AIK' protocols). However, this work is far from complete or conclusive.

TPM command set design decisions of some relevance include the following.

- It was designed to minimise the need to store state between commands. It is hoped that this will reduce the risk of unfortunate combinations of commands.
- The TPM is not permitted to sign externally provided data strings without first 'wrapping' them.

The issue of error codes merits further consideration. There is always a tension between security requirements (which suggest minimising the number of error codes), and usability requirements (which suggest maximising the informativeness of error codes). How many should there be? Is the current set the right balance between being helpful without giving too much away? This appears to be an area that has not really been explored.

Finally, there may be a requirement for an extra command relating to monotonic counters. It would be nice to have a means of verifying the correctness of a counter value exported by a TPM, by being able to obtain a TPM signature on the value (assuming the channel between the TPM and the recipient of the counter is not trusted). More specifically, it would be potentially useful to have a command to get the TPM to sign a statement saying this is the current counter (probably including an externally supplied nonce for freshness checking). Does this apply to any other information one might retrieve from a TPM, e.g. random data? Adding such signed reports from the TPM (signed using an AIK) would appear not to break any fundamental security policies. However, there has been no requirement for such extra commands. To convince the TCG to add such commands would require the construction of a use case.

6 TPM command structure

A number of questions were raised about the structure of individual TPM commands. It was stated that, when designing the commands, there seemed to be plenty of choice. Thus it is important to consider whether or not the right design decisions been made. That is, how might the commands be wrongly structured? This is a particular matter of concern because, when choices are made arbitrarily, sometimes a sub-optimal decision is made because the choice has not been analysed completely.

The TCG architecture is predicated on the assumption that the TPM is trusted, but that the TSS, even if malicious, cannot damage trust in the TPM. It is important to note that implementations of the TSS will be large and complex (the specification contains 800 pages). Commends and responses exchanged between a calling application and he TOM are protected (encrypted and MACed) using a secret shared by the application and the TOM; hence tampering of TPM commands by an untrustworthy TSS will be det6ected. Of course, if authorisation values are cached in the TSS, then protection against a malicious TSS is lost.

Are some TPM error conditions more serious than others? If the random number generator (RNG) fails, then it is clearly very serious. There is a requirement to do something analogous to FIPS 140 [17] internally to the TPM to check the operation of the RNG. However, the detailed design of the RNG functional blocks not mandated.]

Other particularly critical functionality includes the implementations of cryptography, notably hashing and RSA. Is there anything that can be done (analogously to redundancy) to improve hardware reliability.

Finally, if users download malicious software which corrupts the TSS, then it might misuse any authorisation values which it sees. Of course, if software is measured then the PCR values will detect this, and if a hypervisor is present then it would only damage one 'compartment'.

Acknowledgements

The contents of this document have been produced as a collaborative effort by all the participants in the technical workshop. Special thanks are due to Shane Balfe and Eimear Gallery, for writing big chunks of this document, and to Shane Balfe, Eimear Gallery, David Grawrock and Graeme Proudler, for leading the technical discussions.

References

[1] W. A. Arbaugh, D. J. Farber, and J. M. Smith. A secure and reliable bootstrap architecture. In *Proceedings of the 1997 IEEE Symposium*

on Security and Privacy, pages 65–71, Oakland, California, USA, May 1997. IEEE Computer Society Press.

[2] B. Balacheff, L. Chen, S. Pearson, D. Plaquin, and G. Proudler. *Trusted Computing Platforms: TCPA Technology in Context*. Prentice Hall, Upper Saddle River, New Jersey, USA, 2003.

[3] M. Bond. Attacks on cryptoprocessor transaction sets. In C. K. Koc, D. Naccache, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems — CHES 2001, Third International Workshop, Paris, France, May 14–16, 2001, Proceedings*, volume 2162 of *Lecture Notes in Computer Science*, pages 220–234. Springer-Verlag, Berlin, 2001.

[4] M. Bond and R. Anderson. API-level attacks on embedded systems. *IEEE Computer Magazine*, 34(10):67–75, October 2001.

[5] M. Bond and P. Zielinski. Decimalisation attacks for PIN cracking, 2002. Preprint, Computer Laboratory, University of Cambridge.

[6] K. Brincat and C. J. Mitchell. Key recovery atacks on MACs based on properties of cryptographic APIs. In B. Honary, editor, *Cryptography and Coding, 8th IMA International Conference, Cirencester, UK, December 17–19, 2001, Proceedings*, volume 2260 of *Lecture Notes in Computer Science*, pages 63–72. Springer-Verlag, Berlin, 2001.

[7] P. C. Clark and L. J. Hoffman. BITS: A smartcard protected operating system. *Communications of the ACM*, 37(11):66–94, November 1994.

[8] J. Clulow. The design and analysis of cryptographic APIs for security devices, 2003. M.Sc. Dissertation, University of Natal, Durban, South Africa.

[9] J. Clulow. On the security of PKCS #11. In C. D. Walter, C. K. Koc, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems — CHES 2003, 5th International Workshop*, volume 2779 of *Lecture Notes in Computer Science*, pages 411–425. Springer-Verlag, 2003.

[10] A. W. Dent and C. J. Mitchell. *User's Guide to Cryptography and Standards*. Artech House, Boston, MA, 2005.

[11] D. Grawrock. *The Intel safer computing initiative: Bulding blocks for trusted computing*. Intel Press, Hillsboro, OR, 2006.

[12] Peter Gutman. PKI: It's not dead, just resting. *Computer*, 35(8):41–49, 2002.

[13] N. Itoi, W. A. Arbaugh, S. J. Pollack, and D. M. Reeves. Personal secure booting. In *Proceedings of the 6th Australasian Conference on*

Information Security and Privacy, volume 2119 of *Lecture Notes In Computer Science*, pages 130–141. Springer-Verlag, Berlin, 2001.

[14] C. J. Mitchell. Key recovery attack on ANSI retail MAC. *Electronics Letters*, 39:361–362, 2003.

[15] C. J. Mitchell. Truncation attacks on MACs. *Electronics Letters*, 39:1439–1440, 2003.

[16] C. J. Mitchell, editor. *Trusted Computing*. IEE, London, 2005.

[17] National Institute of Standards and Technology (NIST). *Federal Information Processing Standards Publication 140-2 (FIPS PUB 140-2): Security Requirements for Cryptographic Modules*, June 2001.

[18] Trusted Computing Group. *TCG Credential Profiles*, January 2006. Version 1.0: Revision 0.981.

[19] J. Tygar and B. Yee. Dyad: A system for using physically secure coprocessors. Technical report CMU-CS-91-140R, Carnigie Mellon University, May 1991.