# Attestation and Trusted Computing


## Abstract

A look at the cryptographic techniques and protocols used in trusted computing with particular attention to remote attestation.



CSEP 590: Practical Aspects of Modern Cryptography
March 2006


J. Christopher Bare

## Introduction

Trusted computing, in some form, is almost certain to become part of the computing landscape over the next few years. This is because email viruses, trojans, spyware, phishing scams, key-stroke loggers, and security exploits are so much a part of the landscape already.

The current computing infrastructure was built with a premium on openness and interoperability which has paid huge dividends in terms of creativity and innovation. But, the same openness is somewhat problematic for security. Experience has shown that the access control model of present operating systems is inadequate against many types of attacks particularly in the hands of inexpert users.

Previous attempts to roll out cryptographic infrastructure to a mass-market have met with limited success.[10] The various competing visions of trusted computing seek to strike a workable balance between enhanced security and openness and backward-compatibility.

One component of trusted computing that has attracted particular attention is *remote attestation*. Attestation allows a program to authenticate itself and remote attestation is a means for one system to make reliable statements about the software it is running to another system. The remote party can then make authorization decisions based on that information.

This paper will look at the cryptographic techniques and protocols used in trusted computing with particular attention to remote attestation.

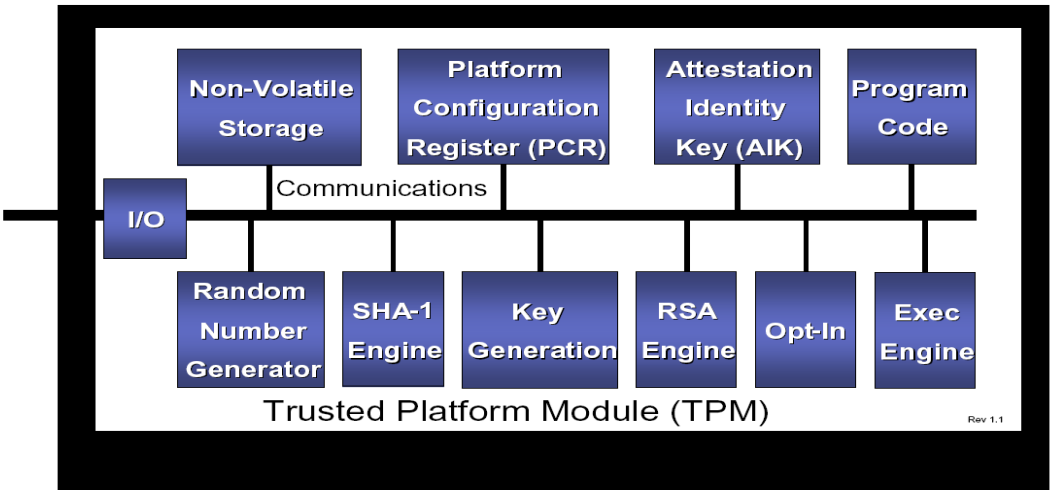
## Trusted Computing

The term trusted computing applies to a number of distinct proposals and initiatives with the general goal of engineering more security into commodity computing systems. The Trusted Computing Group (TCG)[1] is an industry coalition with the goal of creating standards and specifications. Microsoft is a member of the TCG and has its own initiative, Next Generation Secure Computing Base (NGSCB), formerly called Palladium.[2,3]

Some generally agreed upon features of trusted computing are:

- *secure boot* allows the system to boot into a defined and trusted configuration.
- *curtained memory* will provide strong memory isolation; memory that cannot be read by other processes including operating systems and debuggers.
- *sealed storage* allows software to keep cryptographically secure secrets.
- *secure I/O* thwarts attacks like key-stroke loggers and screen scrapers.
- *integrity measurement* is the ability to compute hashes of executable code, configuration data, and other system state information.
- *remote attestation* allows a trusted device to present reliable evidence to remote parties about the software it is running.

We will look most closely at integrity measurement and attestation, as well as the security coprocessor that serves as a local root of trust for these operations. The security coprocessor, or Trusted Platform Module (TPM), is a tamper resistant piece of cryptographic hardware built onto the system board that implements primitive cryptographic functions on which more complex features can be built.




*Trusted Platform Module Architecture* [1]

The TPM has the following capabilities:

- performing public key cryptographic operations
- computing hash functions
- key management and generation
- secure storage of keys and other secret data
- random number generation
- integrity measurement
- attestation

The TPM is manufactured with a public/private key pair built into the hardware, called the *endorsement key* (EK). The EK is unique to a particular TPM and is signed by a trusted Certification Authority (CA).

## Integrity Measurement

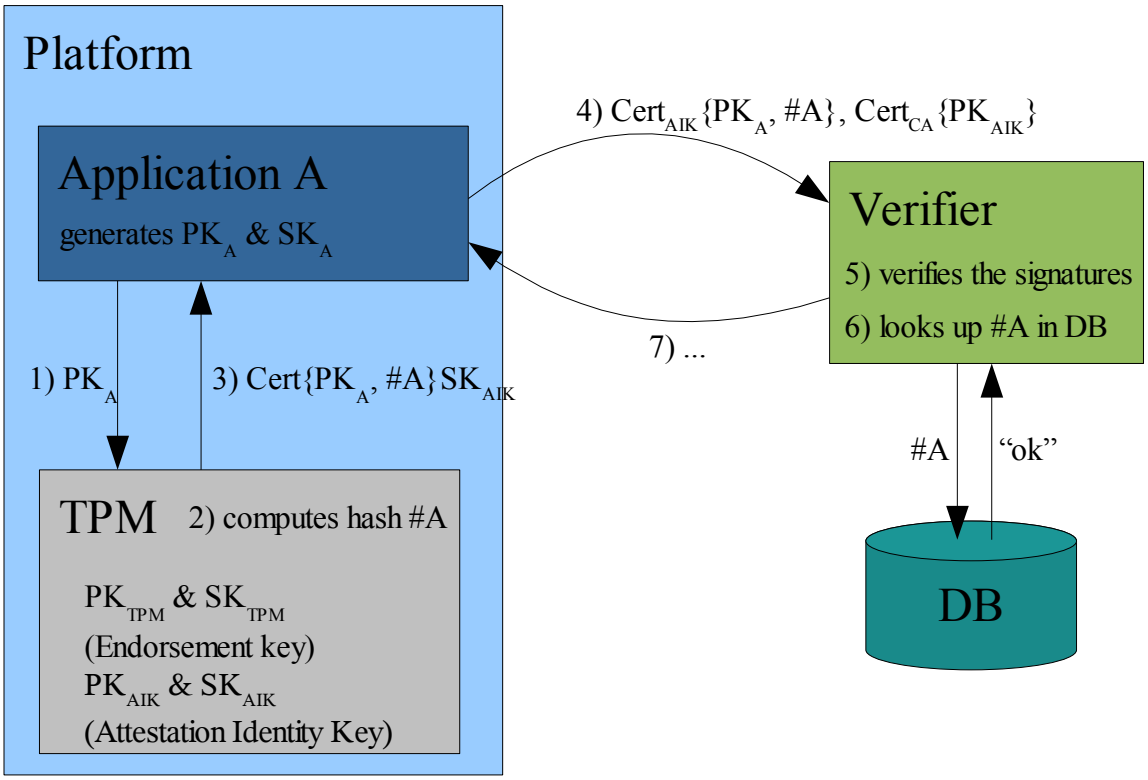
Measurement is the process by which information about the software, hardware, and configuration of a system is collected and digested. At load-time, the TPM uses a hash function to fingerprint an executable, an executable plus its input data, or a sequence of such files. These hash values are used in attestation to reliably establish code identity to remote or local verifiers.

The hash values can also be used in conjunction with the sealed storage feature. A secret can be sealed along with a list of hash values of programs that are allowed to unseal the secret. This allows the creation of data files that can only be opened by specific applications.

## Attestation

Attestation is a mechanism for software to prove its identity. The goal of attestation is to prove to a remote party that your operating system and application software are intact and trustworthy. The verifier trusts that attestation data is accurate because it is signed by a TPM whose key is certified by the CA.

A basic remote attestation protocol looks something like this[4]:




1. The application "A" generates a public/private key pair $PK_A$ & $SK_A$ and asks the TPM to certify it.
2. The TPM computes a hash value #A of the executable code of program "A".
3. The TPM creates a certification including $PK_A$ and #A and signs it with the attestation identity key $SK_{AIK}$.
4. When application "A" wishes to authenticate itself to a remote party, it sends the cert. of its public key and hash value #A along with a cert. issued to the TPM by a trusted certification authority (CA).
5. The remote party to verifies the cert. chain.
6. The remote party looks #A up in a database which maps hash values to trust levels.
7. If application "A" is deemed trustworthy, we continue the communication, probably by using $PK_A$ to establish a session key.

The attestation protocol can be run bidirectionally to allow mutual authentication. For example, a bank wishes to ensure the integrity of the client and the client would like to be sure that they are not connecting to a phishing site.

A more complex protocol might provide the verifier with evidence of the whole software stack including the firmware, OS, and applications. An example of an integrity measurement from a Linux based implementation of trusted computing is shown[7].

```
#000: BC...AB (bios and grub stages aggregate)
#001: A8...5B grub.conf (boot configuration)
#002: 1238A...A22D1 vmlinuz-2.6.5-bk2-lsmtcg
#003: 84ABD...BDA4F init (first process)
#004: 9ECF0...1BE3D ld-2.3.2.so (dynamic linker)
...
#439: 2300D...47882 persfw user (client policy agent)
#440: BB18C...BDD12 libpdauthzn.so (policy client libraries)
#441: D12D9...829EE libpdcore.so
...
#453: DF541A...BE160 local.conf (policy agent)
#454: 6AC585...DC781 authzn persfw.db (policy db)
...
```

### *Remote Policy Enforcement*

The decision of which applications and platforms to trust is a security policy decision. Attestation gives computing entities the ability to accept connections only from those that agree to enforce their policies. A corporate VPN, which is concerned about attacks from compromised clients, may wish to accept connections only from clients running a verified OS at a specified patch level and running a firewall program with a sufficiently strict set of configuration rules. Attestation provides the means to enforce these kinds of policies.

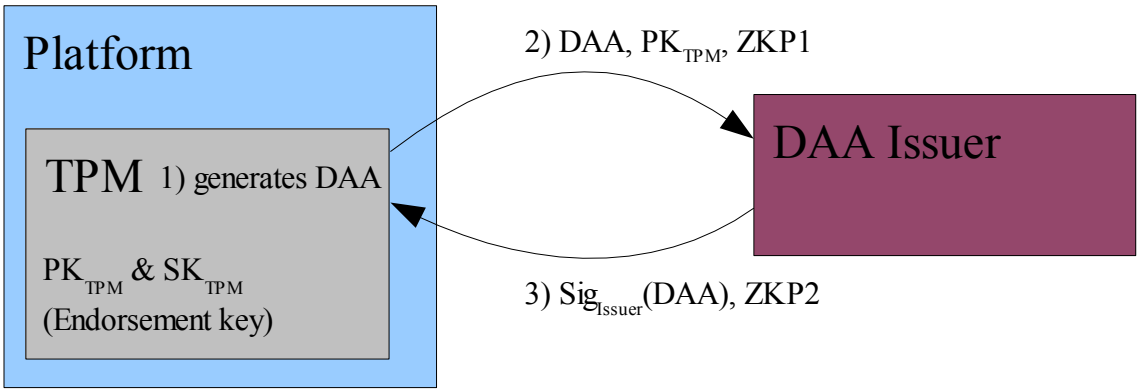
The policies enforced through attestation are entirely arbitrary. They are limited only by the ability to express them either in code or configuration files. We could even imagine policy being expressed in a domain specific language and an automated negotiation process taking place between two communicating entities. The negotiation could establish whether their respective policies are compatible and on what terms and with what restrictions they can interoperate. One example of work is being done in this area is the Web Services Policy Language specification.

### *Privacy Concerns*

The above protocol raises privacy concerns. The fear is that the attestation key and its certificate can be used to track activity and compromise privacy. Originally, a trusted third party was proposed that would provide anonymized identity services to users. The trusted third party would issue certified attestation keys that were not directly traceable to the device and could issue multiple attestation keys to a device in order to prevent correlation. Due to problems with this proposal, another protocol was developed that provides a anonymity without the need for a trusted third party[11].

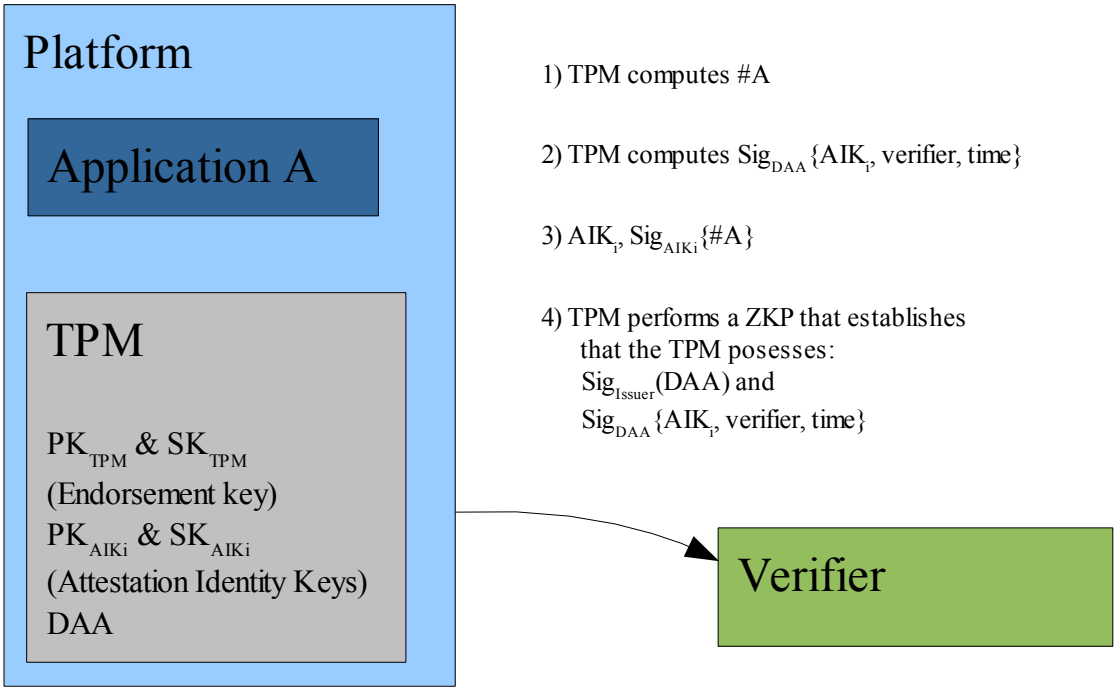
### *Direct Anonymous Attestation*

The direct anonymous attestation (DAA) scheme[6] requires an initial interaction with a DAA issuer to obtain a certificate for a DAA key. Unlike the trusted third party scheme, this need be performed only once.




Both the DAA key and the DAA issuer's signature have a special form. The TPM retains a secret unknown to the DAA issuer and must prove to the issuer that DAA was properly constructed (ZPK1). This prevents the DAA issuer from correlating the attestation keys $AIK_i$ with the DAA key. The DAA issuer also proves that his part of the computation was executed properly (ZKP2). Thus the DAA issuer need not be trusted.

The DAA key is used to sign attestation keys $AIK_i$ generated by the TPM which can then be used for attestation. Attestation proceeds normally, with the attestation data signed with $AIK_i$.



The final trick is to prove to the verifier the $AIK_i$ is valid. The verifier needs to be sure that the TPM does, in fact, possess a DAA key certified by the DAA issuer and the signature of $AIK_i$, the verifier's name, and a timestamp with the same DAA key. The DAA key is not revealed to the verifier in this process.

The construction of the DAA key uses a technique related to a group signature key and the zero knowledge proofs are based on the exponentiation in a finite field as are Diffie-Hellman and RSA. The Fiat-Shamir heuristic is used to merge the steps of the zero knowledge proofs.

## Limitations of Attestation

In either form, the process of attestation has a few limitations[5]. Attestation can reliably tell a verifier what applications are running on a client machine, but the verifier must still make the judgment about whether each given piece of software is trustworthy.

The decision to trust a given piece of software will likely be based on a white list – a list of software known to be trustworthy. A blacklist would be easily subverted by periodically changing a few bytes in a malicious program and might become unmanageably large. Auditing software for security and maintaining a database of trusted applications is a nontrivial task. Any imaginable vetting process is sure to be expensive, cumbersome, and potentially fraught with liability.

Attestation happens at the time executable code is loaded and is therefore unable to provide a view into program behavior at run-time. A program may become compromised during execution, for example, by a buffer overflow exploit. This will not be visible through the attestation mechanism. This risk is partially mitigated by the ability to detect and refuse to communicate with a program with known vulnerabilities.

Frequent release of patches or upgrades poses a problem because new approved hash values must be propagated to verifiers. If the possible configurations can be restricted to monotonically increasing patch level, the problem remains manageable. But, if patches may be applied arbitrarily, the number of possible binaries grows exponentially in the number of patches.

If TPM keys are compromised, then revocation becomes an issue. In the case of direct anonymous attestation, complete anonymity and the ability to detect rogue TPMs seem to be opposing goals. Some compromise of anonymity must be made to allow fraudulent TPM keys to be detected by verifiers.

## Controversy

Trusted computing and attestation are the subject of a raging controversy. Some of the arguments are outlined here.

Enabling trusted computing on a device will result in a certain loss of control. An owner may even be viewed as an adversary on her own machine. This aspect of trusted computing makes some uncomfortable. To address these concerns an "owner-override"

mode has been proposed that would give the owner of the hardware the ability to cause the TPM to give false attestation[9].

An adversarial view of the user is, at times, necessary. Trusted computing seeks to defend against malicious code running with the permissions of the user. In the utility computing use case, a service provider leases computing capacity on a shared data center to outside clients. For this type of business to be viable, customers must be assured that their data is safe from the owners and administrators of the hardware. The owner specifically wants to limit his own privileges. Digital rights management (DRM), controversial in its own right, requires this functionality as well.

Attestation is designed to allow remote policy enforcement. These policies may be benign or draconian and users will need to assess them carefully. There's little doubt that some will try to abuse them.

For example, an unscrupulous businesses may enact policies that result in vendor lock-in [8]. Products from one vendor may refuse to interoperate with those of a competitor. Data files can be cryptographically bound to an application and software vendors will have little incentive to make it easy to migrate to another brand of software. Reverse engineering for compatibility can be rendered impossible.

Competition may be curbed from another angle. In order to establish trust, the software on either side of a transaction must be approved by both parties. Given the expense and risk of assessing the security of even a moderately complex piece of software, this could have the effect of locking out smaller players. For a small but honest software company, or open source project, getting a product on the "trusted" list might prove to be an insurmountable barrier, thus reducing competition, choice in the software market, and garage innovation.

## Conclusion

Trusted computing and attestation provides the means for remote policy enforcement. Where the user's and verifier's interests align this is a good thing. It is clearly in everyone's interest to curtail malicious software. But in situations of opposing interests, caution is warranted.

Perhaps what's needed is a way of allowing the user to specify his own policy and an automated way of flagging or rejecting conflicting policy from outsiders. This might be a more sensible form of "owner override" than the proposed ability to attest falsely.

### *Accountability vs. Enforcement*

In the trusted computing literature, there are distinct emphases on enforcement mechanism and on privacy. It is possible that online rights activists may have over emphasized privacy at the expense of other freedoms. Certainly, privacy or anonymity is vital in some situations, like voting. But in others, it is not very beneficial. Anonymity prevents accountability, and accountability is effective at preventing all sorts of bad behavior.

For example, if all code had to have a certified signature, we could verify the signature when the loading the code. The signer could be held responsible for any back doors or malicious behavior of the code. And if the code was modified for nefarious purposes, the modification would cause the signature verification to fail. The system could refuse to run any programs whose signature did not validate.

In this signed-code scenario, there would be no need for maintaining white lists of hash values, at least for ordinary security situations. Obtaining a certified signing key presents a much smaller barrier to the small players than an infrastructure that requires explicit approval. By default, we trust the program to behave honestly. Yet, we have the means to redress any crimes that do occur.

Where there is accountability, people obey the rules because there are consequences of not doing so. The key benefit of accountability is that things don't have to be locked down as tightly. More openness can be preserved.

### *Acceptance of Trusted Computing*

Many, possibly most, users will gladly accept some loss of control in exchange for a more secure computing environment. Where the mechanisms of trusted computing are abused, users will seek legal remedies or the market remedy of taking their business elsewhere. Eventually, consensus or legislation will define accepted standards. Cryptography is a powerful tool. The question that remains is how wisely it will be used.

Computer users are conditioned by decades of experience to expect the computer to do as it is told. It remains to be seen how people will react the first time their computer says, "I'm sorry, Dave, but I'm afraid I can't do that."

## References


[1] Trusted Computing Group, https://www.trustedcomputinggroup.org/.

[2] P. England, B. Lampson, J. Manferdelli, M. Peinado, and B. Willman, "A Trusted Open Platform", *IEEE Computer*, vol. 36, no. 7, pp. 55-62, 2003.

[3] M. Peinado, Y. Chen, P. England, and J. Manferdelli, "NGSCB: A trusted open system.", *Proceedings of the 9th Australasian Conference on Information Security and Privacy (ACISP 2004)*, 2004.

[4] T. Garfinkel, M. Rosenblum, and D. Boneh, "Flexible OS Support and Applications for Trusted Computing", *Proceedings of the 9th Workshop on Hot Topics in Operating Systems*, May 2003.

[5] V. Haldar, D. Chandra, and M. Franz. "Semantic Remote Attestation – A Virtual Machine directed approach to Trusted Computing". *USENIX Virtual Machine Research and Technology Symposium*, May 2004.

[6] E. Brickell, J. Camenisch, and L. Chen. "Direct anonymous attestation", *ACM Conference on Computer and Communications Security*, pp. 132-145, 2004.

[7] Reiner Sailer, Trent Jaeger, Xiaolan Zhang, Leendert van Doorn, "Attestation-based policy enforcement for remote access", *Proceedings of the 11th ACM conference on Computer and communications security*, October 2004.

[8] Ross Anderson, "Cryptography and Competition Policy - Issues with Trusted Computing", *2nd Annual Workshop on Economics and Information Security,* May 2003.

[9] Seth Schoen, "Trusted Computing: Promise and Risk", *Electronic Frontier Foundation*, 2003.

[10] P. Gutmann. "PKI: It's not dead, just resting", *IEEE Computer*, 35(8):41-49, 2002.

[11] E. Brickell, "Direct Anonymous Attestation: Achieving Privacy in Remote Authentication", *ZISC Information Security Colloquium,* June, 2004