A Robust Integrity Reporting Protocol for Remote Attestation

Frederic Stumpf[⋆], Omid Tafreschi[⋆⋆], Patrick Röder[⋆⋆⋆], Claudia Eckert

Darmstadt University of Technology,
Department of Computer Science,
D-64289 Darmstadt, Germany,
{stumpf,tafreschi,roeder,eckert}@sec.informatik.tu-darmstadt.de

Abstract. Trusted Computing Platforms provide the functionality of remote attestation, i.e. attesting the configuration and status of a system to a remote entity. Remote attestation hereby proves integrity and authenticity of system environments. This is crucial for policy enforcement, which in turn is needed in many usage scenarios, e.g., DRM. However, applying remote attestation solely allows masquerading attacks. These attacks are possible since the concept of remote attestation does not provide any means for establishing secured communication channels. In this paper we describe this kind of attacks against protocols for remote attestation and present a protocol for preventing masquerading attacks.

1 Introduction

The increasing complexity of IT-systems is a major factor for the growing number of system vulnerabilities. Trusted Computing techniques [BCP+03] offer one possibility to overcome this shortcoming by providing a tamper-resistant hardware structure that provides a *root of trust*. The introduction of the Trusted Platform Module (TPM) which is specified by the Trusted Computing Group (TCG) provides vital functions for IT-security. Besides the potential of offering a protected storage for storing cryptographic keys, the Trusted Platform Module also offers the possibility to attest the configuration of the local platform to a remote platform which is called *remote attestation* (RA). This process is of particular interest in context of Digital Rights Management (DRM) [LSNS03] or Enterprise Security [SvDW04] to ensure that the client platform is trustworthy and is behaving in accordance with a defined policy. This becomes relevant when sensitive data is transferred which should only be used in an authorized way, i.e. the content can be used by an authorized user on a specified platform.

[⋆] The author is supported by the German Research Foundation (DFG) under grant EC 163/4-1, project *TrustCaps*.

[⋆⋆] The author is supported by the German Federal Ministry of Education and Research under grant 01AK706C, project *Premium*.

[⋆⋆⋆] The author is supported by the PhD program *Enabling Technologies for Electronic Commerce* of the German Research Foundation (DFG).

For this purpose a policy can be defined which is enforced by the local client software. Since the policy enforcement mechanism is executed on the client-side, an attacker can deactivate this mechanism by modifying the local client software.

To avoid these kinds of attacks one may attest the status of the client platform using the RA before sending sensitive data to the client. Thus, the sender has a confirmation about the trustworthiness of the platform in question. However, RA protocols are not per se resident against masquerading attacks[1]. An attacker who is in control of one malicious and one honest client may bypass the remote attestation by spoofing his malicious client to be the honest one. The attacker simply forwards all attestation queries and sends them to one client, which is in a trustworthy system state. The honest client sends the answer back, which must only be transferred back to the requester. This attack is a masquerading [RMAW99] attack, with the extension that the attacker is also in possession of the honest client. But, as we will see in the following sections, this is not an essential requirement. After this procedure the malicious client platform state has then been approved trustworthy and protected digital content is transferred in the next step to the pretended trusted platform, that does not enforces any policies to protect this documents.

We believe that the masquerading attack is critical in the context of DRM systems. In this case, the content provider must be sure, that the policy, which defines the usage permissions, is enforced correctly on the DRM-client. Therefore, the content provider first attests the state of the platform before he delivers protected digital content. This content is encrypted to bind it to the attested machine. However, the content provider is never sure that the key which is used for binding the content is stored on the attested client and is not on a second machine.

The remainder of this paper is organized as follows: We provide some background information on the TCG concepts in Section 2. Section 3 introduces the remote attestation and Section 4 describes the attack. In Section 5 the attack scheme is evaluated and the requirements for the attack are explained. Section 6 shows that this type of attack can easily be avoided by introducing a new integrity reporting protocol. Section 7 reviews related work, and finally Section 8 provides the conclusions.

2 Background on TCG-mechanisms

The core of the TCG mechanisms [Gro06,BCP$^+$03] is the Trusted Platform Module (TPM), which is basically a smartcard soldered on the mainboard of a PC. The TPM serves as the *root of trust*, because its hardware implementation makes it difficult to tamper with, and therefore it is assumed to be trustworthy. One must also assume that the hardware vendor is trustworthy and has designed the TPM chip according to the specification. Although the TPM chip is not specified to be tamper-resistant, it is tamper-evident, meaning that unauthorized manipulations can be detected.

[1] These attacks are in [GPS06] referred to as relay attacks.

The TPM can create and store cryptographic keys, both symmetric and asymmetric. These keys can either be marked migratable or non-migratable, which is specified when the key is generated. In contrast to non-migratable keys, migratable keys can be transferred to another TPM. Due to its limited storage capacity, the TPM can also store keys on the hard disk. In this case, these keys are encrypted with a non-migratable key, assuring the same level of security as if the keys were stored directly in the TPM. The TPM is able to perform calculations on its own, e.g., it can use the generated keys for encryption and decryption.

In the context of this paper, the Platform Configuration Registers (PCRs) are of particular interest. These registers are initialized on power up and are used to store the software integrity values. Software components are measured by the TPM and the corresponding hash-value is then written to this platform configuration register by extending the previous value of a specific PCR. The following cryptographic function is used to calculate the values for the specific registers:

$$Extend(PCR_N, value) = SHA1(PCR_N || value)$$

For every measured component an event is created and stored in the stored measurement log (SML). The PCR values can then be used together with the SML to attest the platform's state to a remote party. To make sure that these values are authentic, they are signed with a non-migratable TPM signing key, the Attestation Identity Key (AIK). The remote platform can compare these values with reference values to see whether the platform is in a trustworthy state or not. The TCG assumes that a trusted operating system[2] measures the hash value of every process started after the boot process.

The TPM additionally offers a number of different signing keys. One major key is the Endorsement Key (EK) which is generated by the module manufacturer and injected into the TPM. The EK uniquely identifies the TPM and is used to prove that the TPM is genuine. In addition, the EK is used to obtain an Attestation Identity Key (AIK). An AIK is created inside the TPM, signed with the private portion of the EK, and the public part is transferred to a third party (a Privacy-CA). The Privacy-CA verifies that the platform is a genuine TPM and creates a certificate which binds the identity key to the identity label and generic information about the platform. This certificate, also known as identity credential is sent to the TPM and later used to attest the authenticity of a platform configuration.

3 Remote Attestation

The remote attestation is used to attest the configuration of an entity to a remote entity. This procedure is widely used to get integrity information before a client

[2] A trusted OS is not part of the TCG specification. For a description of a trusted OS confer [GPC$^+$03,Bas06,SZJvD04].

proceeds with the communication in order to use a service or receive data, e.g., digital content. This mechanism is referred as integrity reporting and can be applied in many scenarios and different applications, such as controlling access to a network depending on the trustworthiness of the client [SJZvD04]. The integrity reporting mechanism is also one requirement mechanism in the context of DRM applications, since it is obviously required that the DRM-client software is in a trustworthy state and executes a certain policy to prohibit unauthorized use, copy or redistribution of intellectual property [RC05].

3.1 Integrity Reporting Protocols

The concept of remote attestation has been developed to enable integrity reporting protocols. In this section we discuss an integrity reporting protocol proposed by [SZJvD04], which is based on the challenge-response authentication [BM92] and is used to validate the integrity of an attesting system.

Figure 1 illustrates the remote attestation of B against A and provides the background information on integrity reporting protocols. In step 1 and 2, A creates a non-predictable nonce and sends it to the attestor B. In step 3a, the attestor loads the Attestation Identity Key from the protected storage of the TPM by using the storage root key (SRK). In the next step, the attestor performs a *TPM_Quote* command, which is used to sign the selected PCRs and the provided nonce with the private key AIK_{priv}. Additionally, the attestor retrieves the stored measurement log (SML). In step 4, the attestor sends the response consisting of the signed *Quote*, signed nonce and the SML to A. The attestor also delivers the AIK credential which consists of the AIK_{pub} that was signed by a Privacy-CA.

1.	A	: create a non-predictable 160bit *nonce*
2.	A → B	: ChallengeRequest(*nonce*)
3a.	B	: loadkey(AIK_{priv})
3b.	B	: retrieve $Quote = sig\{PCR, nonce\}_{AIK_{priv}}$
3c.	B	: get stored measurement log (SML)
4.	B → A	: ChallengeResponse(*Quote*, SML) and $cert(AIK_{pub})$
5a.	A	: validate $cert(AIK_{pub})$
5b.	A	: validate $sig\{PCR, nonce\}_{AIK_{priv}}$
5c.	A	: validate *nonce* and SML using PCR

Fig. 1. Integrity reporting protocol [SZJvD04]

In step 5a, A validates if the AIK credential was signed by a trusted Privacy-CA thus belonging to a genuine TPM. A also verifies whether AIK_{pub} is still valid by checking the certificate revocation list of the trusted issuing party. This step was also designed to discover masquerading by comparing the unique identification of B with the system identification given in AIK_{pub}. Nevertheless, this verification does not discover masquerading attacks as we will see in the next section.

In the next step, A verifies the signature of the *Quote* and checks the freshness of *Quote* in step 5c. Based on the received stored measurement log and the PCR values A processes the SML and re-computes the received PCR values. If the computed values match the signed aggregate, the SML is valid and untampered. A now only verifies if the delivered integrity reporting values match given reference values, thus A can decide if the remote party is in a trustworthy system state.

According to [SZJvD04] the protocol described above should be resistant against replay attacks, tampering and masquerading. However, an attacker can successfully perform masquerading attacks on that protocol. We describe these in the following section.

4 Masquerading Attack Scheme

The attacker considered here has two platforms under his control. One platform runs a trustworthy operating system with the client software that enforces a certain policy, e.g., the DRM-client. This platform (C) represents the client that is conform to the original client software and therefore untampered. This platform is also equipped with a genuine TPM that supports the policy enforcement of the DRM-client. The attacker is also in control of one malicious client platform (M) that wants to gain control of protected digital content. We refer to this client as malicious client, since his enforcement mechanism has been tampered with and it is not conform to the original client software. We require for our attack that C answers the request from M, i.e. C is not configured to answer only requests from A. This is a necessary requirement for the success of the attack and is discussed in detail in section 5.

In our attack scheme, the attacker bypasses the remote attestation of M, by using the platform configuration of the honest client to attest his malicious client running on M.

1.	A	: create a non-predictable 160bit *nonce*
2.	A $\rightarrow$ M	: ChallengeRequest(*nonce*)
2a.	M $\rightarrow$ C	: ChallengeRequest(*nonce*)
3a.	C	: loadkey(AIK_{priv})
3b.	C	: retrieve $Quote = sig\{PCR, nonce\}_{AIK_{priv}}$
3c.	C	: get stored measurement log (SML)
4.	C $\rightarrow$ M	: ChallengeResponse(*Quote*, SML) and $cert(AIK_{pub})$
4a.	M $\rightarrow$ A	: ChallengeResponse(*Quote*, SML) and $cert(AIK_{pub})$
5a.	A	: validate $cert(AIK_{pub})$
5b.	A	: validate $sig\{PCR, nonce\}_{AIK_{priv}}$
5c.	A	: validate *nonce* and SML using PCR

Fig. 2. Attacking the integrity reporting protocol

Figure 2 depicts the attack against the integrity reporting protocol. The challenging party A wants to securely validate the integrity of the attesting malicious system M. The malicious system M itself transfers all messages from A to the honest client C. The protocol is the same as the protocol shown in figure 1 except for step 2a and 4, in which only the messages are forwarded.

Figure 3 simplifies the attack on the presented protocol by reducing it to the transferred messages. This figure shows the challenger (platform A) that wants to attest the client (M) before protected data is transferred. Platform M and C are working collaboratively. Platform M is authenticated on the basis of the provided information through platform C, this is simplified through step 5 and 6. The figure also shows that the attack cannot be prevented with the use of a secure mutual authenticated TLS channel, since this protocol only authenticates a certain user. An attacker may be able to extract the X.509 certificate and the corresponding keys for a mutual authentication. Since standard TLS-applications, e.g., web browsers, support the export of the keys by the platform owner, he may transfer the certificates and the private keys to the non-conformant host. The server (platform A) then authenticates the client based on the certificates and sends an attestation request to the malicious platform, which answers the request in the previously described manner.

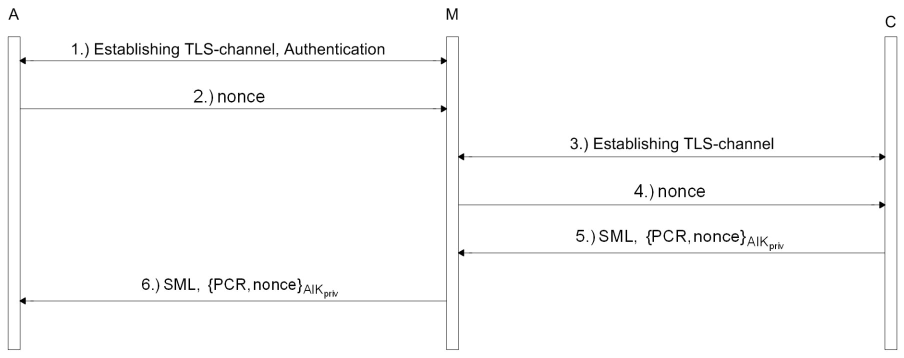
A
M
C
1.) Establishing TLS-channel, Authentication
2.) nonce
3.) Establishing TLS-channel
4.) nonce
5.) SML, {PCR, nonce}$_{AIK_{priv}}$
6.) SML, {PCR, nonce}$_{AIK_{priv}}$

Fig. 3. Collaborative masquerading attack on RA

The shortcoming of the integrity reporting protocol is caused by the restricted usage possibilities of AIKs. According to [Gro06] the AIKs can exclusively be used for proving the authenticity of a platform. This means AIKs can neither be directly used to establish secure channels nor to authenticate communication partners. Therefore, the challenger cannot be sure whether the received message belongs to the attesting system. He only knows that he received a message from a genuine TPM.

Additionally, the possession of multiple AIKs is possible, the only requirement is, that the corresponding certificate must be certified by a trusted Privacy-

CA thus belonging to a valid EK. As a consequence, the challenging party cannot map AIKs to users.

Even if the server is in possession of the AIK_{pub} certificate and forbids the creation of new AIKs masquerading attacks cannot be prevented, since the attesting system M pretends that it is in possession of the corresponding AIK_{priv} by using the integrity values with a valid AIK signature from platform C (Steps 5 and 6 in Figure 3). The challenging system is therefore not able to detect this attack in step 5a (Figure 2), since the attesting system delivers all information that identifies it as platform C. After the platform is authorized, the malicious platform is assumed to be trusted.

5 Evaluation of possible approaches

In this section we present some approaches that appear to solve the problem. We show for each of these approaches why they fail to prevent the described attack.

5.1 Verifying RA-challenges

Since the attacker cannot modify the software on the honest platform, the honest platform may decide which RA-challenge it will answer and which it will reject. However, this decision process becomes quite complex when many different processes require an attestation.

On way to decide this is by authenticating the sender of the query, e.g., by using certificates. In this case, the server's certificate must be included in the integrity measurement of the client software to detect a manipulation on that certificate.

The TCG specification proposes that all software components are measured and stored in the PCRs. Additionally, all corresponding events that occur are recorded in the SML, which is later transferred to the challenger. Therefore, the transferred platform configuration includes all processes that are running on the client machine. This is necessary to make an explicit statement of the platform configuration.

Any arbitrary software component that can answer RA-request can respond to the malicious RA-challenge. An attacker can use such a software to circumvent the verification of the RA-request of the client software. However, the verifying party detects that this software component is listed in the SML, where the correct DRM-software is also listed. On the downside, the SML does not tell which software issued the answer to the RA. The attacker can either use a software supplied by himself to answer the RA or he can misuse an existing application that serves a legitimate purpose. In the second case, the issuing software appears to be trustworthy, because its software vendor might offer valid SML-reference values. Regardless of which client software responds to the RA-Request, the only requirement is that the nonce, which was sent from the server-application (platform A), is used for the attestation. Hence, bugs or design flaws in the client software which have the result that every RA-request is answered, affect other trusted client software. One approach is to forbid the installation of other client software components than the DRM-software. However, this leads to high restrictions which can hardly be accepted in open environments.

5.2 Using shared secrets

One method may be to exchange a shared secret key between server and client, which is generated in the TPM. This shared secret key must be marked as a non-migratable key and the public part must be transferred to the server. This key is used to securely exchange a key for the following communication. The problem is, that this public key must be transferred through a protected second channel, otherwise the server is not sure that the key belongs to the correct communication partner.

5.3 Using AIK as session key

Another method is to use the AIK_{pub} as encryption key to exchange the shared key for the following communication. In this case, the following traffic cannot be decrypted by the attacker, since he does not have the private portion of the AIK, which is stored in the protected storage of TPM. But, the AIK must not be used as encryption key as specified by the TCG [Gro06].

6 A robust Integrity Reporting Protocol

To overcome the described attack, the malicious host, who is placed between the server and the collaborative host must be excluded from the following communication flow. Therefore, cryptographic keys must be used and included in the signed messages from C in the response phase. The specification of the *TPM_Quote* command allows us to include additional data and to sign it with the AIK. Thereby, it is possible to use the challenge response messages for the key-exchange.

One possibility is to adopt the Diffie-Hellman (DH) key exchange protocol [DH76] and to integrate it into the integrity reporting protocol. One may also use RSA [RSA83] and generate the corresponding keys inside the TPM. Subsequently, these keys are used to exchange a shared session key which is used to encrypt the upcoming communication. Since the TPM does not support symmetric encryption, the CPU must be used for that purpose. The main difference between both approaches is that the DH parameters must be generated by the CPU whilst the RSA keys are generated by the TPM. Both approaches are equivalent in terms of security, since they both offer a secure way to exchange a key. Computing the RSA key with the TPM offers no advantage compared to computing the DH parameters with the CPU, since the shared communication key must be passed to the CPU anyway. In both cases the required random numbers can be computed by the secure random number generator of the TPM.

1a.	A	: create a non-predictable 160bit *nonce*
1b.	A	: GenerateKey(K_{pub}^{A}, K_{priv}^{A})
2.	A → C	: ChallengeRequest($nonce$,K_{pub}^{A})
3a.	C	: GenerateKey(K_{pub}^{C}, K_{priv}^{C})
3b.	C	: loadkey(AIK_{priv})
3c.	C	: retrieve $Quote = sig\{PCR, SHA1(nonce, K_{pub}^{C})\}_{AIK_{priv}}$
3d.	C	: get stored measurement log (SML)
3e.	C	: ComputeSessionKey(K^{AC})
4.	C → A	: ChallengeResponse($Quote$, K_{pub}^{C}, SML) and $cert(AIK_{pub})$
5a.	A	: validate $cert(AIK_{pub})$
5b.	A	: validate $sig\{PCR, SHA1(nonce, K_{pub}^{C})\}_{AIK_{priv}}$
5c.	A	: validate $nonce$ and SML using PCR
5d.	A	: ComputeSessionKey(K^{AC})
6.	A	: create a non-predictable 160bit $nonce_1$
7.	A → C	: ChallengeRequest($nonce_1$)
8.	C	: compute $Response = enc\{nonce_1\}_{K^{AC}}$
9.	A → C	: ChallengeResponse($Response$)
10.	A	: validate $nonce_1$

Fig. 4. Enhanced integrity reporting protocol

To protect the integrity reporting protocol against masquerading attacks, we enhance it with a key agreement protocol. The modified integrity reporting protocol is shown in figure 4 with the extension to use Diffie-Hellman parameters. It is essential for the protocol that both parties agree to one common generator g and one common group m. The asymmetric keys are then generated and computed as described in [DH76].

In step 1b, A generates $K_{pub}^{A} = g^{a}\ mod\ m$, while a is the private part of K^{A}. The major enhancement is that the attesting party generates a public key K_{pub}^{C} in step 3a, includes the generated key together with the PCRs and the *nonce* in the *Quote* message and signs the *Quote* with the AIK. The public key K_{pub}^{C} is generated using the client's CPU and injected as *external data* into the *TPM_Quote* operation. Since the *TPM_Quote* command only allows to include 160bits of external data, the package must be reduced to fit the 160bit length by applying $SHA1$. Because C is running a trusted OS with a trustworthy platform configuration the private part of the key is not accessible to a potential malicious client. Finally, in step 5d the challenging party computes the shared session key K^{AC} by using its own private part and the public part of C. After the session key has been computed, A verifies whether C is also in possession of this shared key, by executing a second challenge-response authentication which is carried out in steps 6 - 10.

6.1 Protocol Discussion

The presented protocol prevents an attacker from spoofing his malicious software configuration since all following messages are encrypted with the computed session key. It is also impossible for M to compute an own session key between him and A since his software is in a compromised state and his TPM is providing malicious platform configurations to A. Given that, the malicious host has no access to the private part of the generated session key which is stored on platform C. The generated symmetric session key K^{AC} has to be used to encrypt all following data, which can only be decrypted on the machine which possesses the session key. This session key can not be transferred to the malicious host by the platform owner, since the extraction, e.g., by memory dump or by modifying the system software, would lead to a non-conformant system state which will be detected in the attestation phase.

It may be possible for the malicious client M to substitute K^{A}_{pub} with his own public key K^{B}_{pub} and then transfers this key to C. Since M cannot modify the response from C, A generates the shared session key K^{AC} which is later used for the encryption. The second challenge response authentication detects this substitution since neither M nor C are in possession of the correct K^{AC} and are therefore not able to encrypt the delivered $nonce_1$. It may also be possible that B modifies the common generator g, or the common group m. The substitution of K^{A}_{pub} and the modification of g or m do not lead to a security problem, since in both cases the attacker would not be able to perform the second challenge-response authentication.

One could also create a bound keypair and use *TPM_CertifyKey* to prove that the corresponding private key is held in a trusted TPM. This asymmetric key is then used to exchange a shared symmetrical key between both parties. However, this approach is less performant compared to our solution since it requires three additional operations: The generation of an asymmetrical bound keypair, the signing of this keypair with *TPM_CertifyKey* and the signing of the symmetrical keypair with the generated bound keypair. Moreover, using CertifyKey would lead to additional signature verification procedures.

7 Related Work

Since the specifications of the TCG are still in progress, many open issues exists. There is a large number of work focusing on the concepts of trusted computing. [SZJvD04] presents a comprehensive prototype based on trusted computing technologies. In particular, it comes up with an architecture for integrity measurement, which contains the integrity reporting protocol, which we enhanced in this paper to make it resistant against masquerading attacks.

Terra [GPC+03] provides an approach for remote attestation. It supports the integrity measurement of virtual machines providing runtime environment for sets of processes. The approach does not build up on TPM and does not provide a protocol for integrity reporting to remote entities, which are the main differences to our work.

Our work is based on the assumption that a trusted OS provides us with the measurement of all executed code, i.e. binary attestation. In contrast to that, [HCF04,SS04] focus on semantic attestation based on attesting the behavior of software components. However, the idea behind our protocol can easily be applied to these approaches.

[BLP05] proposes the integration of key exchange protocols into Direct Anonymous Attestation (DAA) [BCC04] in P2P networks, which is basically similar to our approach. However, the objectives of the integration of key exchange protocols are different, since [BLP05] aims at building stable identities in P2P networks. Additionally, the presented approach does not feature integrity reporting and can not be directly applied to remote attestation.

Another related work is [GPS06] which aims at building secure tunnels between endpoints. But this approach adds a new platform property certificate, which links the AIK to the TLS-Certificate. Moreover, the presented approach focuses on server attestation, which needs in turn an additional trusted CA that offers the platform property certificate. In contrast to that, our approach focuses on client attestation without an additional trusted CA, since we directly bind the cryptographic channel to the AIK.

8 Conclusions

This paper shows that masquerading attacks against integrity reporting protocols are possible. This evolves because the AIK does not reveal whether the attesting system is the one, which really provides the measured values. This shortcoming may become relevant when remote attestation is used to guarantee the trustworthiness of client systems, such as DRM-clients. To address this issue, we have presented a robust protocol which prevents masquerading attacks. For that purpose we add a key agreement scheme into an integrity reporting protocol. The resulting protocol guarantees that the communication after the remote attestation is authentic, i.e. the challenger communicates with the system that provided the measurement values.

References

Bas06. European Multilaterally Secure Computing Base. Towards trustworthy systems with open standards and trusted computing. http://www.emscb.de/, 2006.

BCC04. Ernie Brickell, Jan Camenisch, and Liqun Chen. Direct anonymous attestation. In *CCS '04: Proceedings of the 11th ACM conference on Computer and communications security*, pages 132–145, New York, NY, USA, 2004. ACM Press.

BCP$^+$03. Boris Balacheff, Liquin Chen, Siani Pearson, David Plaquin, and Graeme Proudler. *Trusted Computing Platforms.* Hewlett-Packard Company, 2003.

BLP05. Shane Balfe, Amit D. Lakhani, and Kenneth G. Paterson. Trusted computing: Providing security for peer-to-peer networks. In *Proceedings of the Fifth IEEE International Conference on Peer-to-Peer Computing (P2P'05)*, 2005.

BM92. Steven M. Bellovin and Michael Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *Proc. IEEE Symposium on Research in Security and Privacy*, pages 72–84, 1992.

DH76. Whitfield Diffie and Martin Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.

GPC$^+$03. Tal Garfinkel, Ben Pfaff, Jim Chow, Mendel Rosenblum, and Dan Boneh. Terra: a virtual machine-based platform for trusted computing. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 193–206, New York, NY, USA, 2003. ACM Press.

GPS06. Kenneth Goldman, Ronald Perez, and Reiner Sailer. Linking remote attestation to secure tunnel endpoints. In *First ACM Workshop on Scalable Trusted Computing*, Fairfax, Virginia, November 2006.

Gro06. Trusted Computing Group. Trusted Platform Module (TPM) specifications. Technical report, `https://www.trustedcomputinggroup.org/specs/TPM`, 2006.

HCF04. Vivek Haldar, Deepak Chandra, and Michael Franz. Semantic remote attestation: A virtual machine directed approach to trusted computing. In *USENIX Virtual Machine Research and Technology Symposium*, 2004 2004.

LSNS03. Qiong Liu, Reihaneh Safavi-Naini, and Nicholas Paul Sheppard. Digital rights management for content distribution. In *ACSW Frontiers '03: Proceedings of the Australasian information security workshop conference on ACSW frontiers 2003*, pages 49–58, Darlinghurst, Australia, Australia, 2003. Australian Computer Society, Inc.

RC05. Jason F. Reid and William J. Caelli. DRM, trusted computing and operating system architecture. In *ACSW Frontiers '05: Proceedings of the 2005 Australasian workshop on Grid computing and e-research*, pages 127–136, Darlinghurst, Australia, Australia, 2005. Australian Computer Society, Inc.

RMAW99. W. M. Row, Donald. J. Morton, B. L. Adams, and A. H. Wright. Security issues in small linux networks. In *SAC '99: Proceedings of the 1999 ACM symposium on Applied computing*, pages 506–510, New York, NY, USA, 1999. ACM Press.

RSA83. Ronald L. Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 26(1):96–99, 1983.

SJZvD04. Reiner Sailer, Trent Jaeger, Xiaolan Zhang, and Leendert van Doorn. Attestation-based policy enforcement for remote access. In *CCS '04: Proceedings of the 11th ACM conference on Computer and communications security*, pages 308–317, New York, NY, USA, 2004. ACM Press.

SS04. Ahmad-Reza Sadeghi and Christian Stueble. Property-based attestation for computing platforms: caring about properties, not mechanisms. In *NSPW '04: Proceedings of the 2004 workshop on New security paradigms*, pages 67–77, New York, NY, USA, 2004. ACM Press.

SvDW04. Reiner Sailer, Leendert van Doorn, and James P. Ward. The role of TPM in Enterprise Security. *Datenschutz und Datensicherheit (DuD)*, September 2004.

SZJvD04. Reiner Sailer, Xiaolan Zhang, Trent Jaeger, and Leendert van Doorn. Design and Implementation of a TCG-based Integrity Measurement Architecture. In *13th USENIX Security Symposium.* IBM T. J. Watson Research Center, August 2004.