Journal of Universal Computer Science, vol. 16, no. 4 (2010), 507-518
submitted: 10/10/09, accepted: 21/12/09, appeared: 28/2/10 © J.UCS

On Reliable Platform Configuration Change Reporting Mechanisms for Trusted Computing Enabled Platforms

Kurt Dietrich
(Institute for Applied Information Processing and Communications (IAIK)
Graz University of Technology
Inffeldgasse 16a, 8010 Graz, Austria
Kurt.Dietrich@iaik.tugraz.at)

Abstract: One of the most important use-cases of Trusted Computing is *Remote Attestation*. It allows platforms to get a trustworthy proof of the loaded software and current configuration of certain remote platforms, thereby enabling them to make decisions about the remote platforms' trust status. Common concepts like Internet Protocol security or Transport Layer Security make these decisions based on shared secrets or certificates issued by third parties. Unlike remote attestation, these concepts do not take the current configuration or currently loaded software of the platforms into account. Consequently, combining remote attestation and existing secure channel concepts can solve the long lasting problem of secure channels that have to rely on insecure channel endpoints. Although this gap can now be closed by Trusted Computing, one important problem remains unsolved: A platform's configuration changes every time new software is loaded. Consequently, a reliable and in-time method to provide a proof for this configuration change - especially on multiprocess machines - is required to signal the platforms involved in the communication that a configuration change of the respectively other platform has taken place. Our research results show that a simple reporting mechanism can be integrated into current Trusted Platform Modules and Transport Layer Security implementations with a few additional Trusted Platform Modules commands and a few extensions to the TLS protocol.

Key Words: Trusted computing, platform configuration reporting, secure channels, TLS, Remote Attestation
Category: L.4, K.6.5

1 Introduction

Secure channel technologies - like Transport Layer Security (TLS) - are crucial components of many today's security concepts. This technology is used in many applications e.g. netbanking, e-commerce, e-government etc.. Modern concepts, like Trusted Computing, can considerably improve the security of these channels by providing reliable information about the current configuration of the channel's endpoints.

Many research projects are working to find an answer to the question how TPMs and Trusted Computing features could be integrated into secure channel technologies like transport layer security (TLS) or IP security (IPSec). In all these approaches, attestation information of a platform's configuration is presented to the remote platform and a secure channel is only established if the platform accepts the attested configuration. However, this attestation information only contains the configuration changes (i.e. software loaded) that have been recorded so far. Once the secure channel is opened, it is hard to detect a change in the configuration of a platform by a remote party. Detecting these changes is important as a platform that passed the attestation process with a valid configuration could change into a configuration that is not accepted by the host. This change into an invalid state could be triggered by loading a virus or Trojan Horse that could do substantial damage to the platform before the connection can be closed.

The common technique to detect such changes is to periodically read the configuration from the platform configuration registers (PCRs) or to periodically perform a TPM_Quote operation [TCG 2007]. However, executing a TPM_Quote command and validating the information is time consuming. Depending on the platforms's policy, each time a Quote is requested, a new attestation identity key (AIK) has to be created. To worsen the situation, a newly created AIK also has to be newly certified, which involves requesting an AIK certificate from a privacy CA.

These steps have to be performed for every newly established secure connection. When considering the situation on a common platform where multiple processes are concurrently running and opening channels to remote platforms, the situation is even worse. Multiple concurrent TLS sessions are active on the same platform, and different AIKs might be employed for each session. Consequently, each session has to load its own AIK(s) into the TPM prior to performing a quote operation.

For server platforms that have to handle hundred to thousands of connections at the same time, this procedure without a doubt leads to a great bottleneck and is, therefore, not applicable on these platforms. The ongoing loading and un-loading of AIKs also involves asymmetric cryptography. Another drawback of this technique is the missing assurance that a platform change is delivered in-time - malicious software could be loaded on a platform without notice of the local platform. Consequently, a new efficient mechanism that is able to report configuration changes at the time when they occur and which can be applied on heavy duty server platforms is required.

In this paper, we discuss two approaches for simple, yet reliable and secure reporting mechanisms of configuration changes that can easily be integrated into existing trusted computing enhanced TLS frameworks. We propose to modify the TLS mac computation [Dierks and Allen 1999] which is used for integrity protection of the transported data as follows: instead of computing the mac in software on the platform's CPU, we propose computing the mac within the TPM. This allows a platform to recognize configuration changes of the remote platform. By integrating the PCR values in the computation of the mac-key, changes in the configuration directly influence the mac-value of the TLS records. Consequently, the verifying platform can detect these changes and has - as the computation is performed inside the TPM - a reliable proof of this fact.

In this paper, extensions and alternatives to our previous proposal discussed in [Dietrich 2008] are presented. We discuss a different method for computing the mac-key inside the TPM and analyse the advantages and disadvantages of both approaches. All the proposed approaches are implemented in our proof-of-concept prototypes. Finally, we give a discussion of the set of commands we integrated in our prototypes.

Throughout the remainder of the article, we use the term *trusted channel* for Trusted Computing enhanced secure connections.

The paper is organized as follows: Section 1.1 gives an overview of related work and additional background information on trusted computing enhanced secure channels. This is followed by a discussion of our concept in detail in Section 2 and the required modifications of the TPM and the TLS specification in Section 2.6. Section 2.1 gives an overview of our second approach for modifying a TPM to support configuration change reporting including a discussion of additional commands that have been added to the TPM specification for the second approach. Moreover, we discuss how TPMs can be modified to support TLS client authentication.

Finally, we give a comparison of both mac-key generation concepts, summarize the results and give an outlook on future investigations and improvements of the concepts.

1.1 Related Work

A lot of different publications concerning trusted channels exist. Although all of these publications discuss the integration of Trusted Computing technology into secure channels, however, none of them address the problem of reporting a configuration change while a trusted channel is open.

The publication of Goldman et al [Goldman, Perez, Sailer 2006] analyses several methods of linking server endpoint validation with the TCG's proposed remote attestation. They investigate relay attacks where a compromised server might relay a remote attestation quote from a trusted server to an untrusted one. Furthermore, they introduce the idea of a *platform property certificate* that links attestation identity keys to platform endpoint properties. This approach focuses on virtualized environments and allows fast endpoint certificate revocation and creation with application dependent security properties. However, they do not address the problem of state change reporting.

Trusted Network Connect (TNC) is a concept proposed by the TCG which is working on a reference architecture that focuses on policy enforcement and authentication for granting network access. The architecture is basically generic and is based on collecting and verifying integrity information of the communication partners. Which integrity information is actually included is not discussed, as it is part of the policy and depends on the invoked platforms. Moreover, the framework focuses on policy enforcement and does not necessarily rely on remote attestation and TPM support. Like Goldman et al, they do not consider configuration changes while trusted channels are open. Furthermore, they do not address the problem of linking the channels to the certain TPM.

The first approach addressing the report of configuration changes is discussed in [Gasmi 2007]. The authors propose an implementation that reliably determines the trustworthiness of the communication endpoints and show how the implementation can be combined with secure channel technology. In contrast to other proposals, they try to address the problem of configuration change reporting. They propose to store the session keys in the trusted computing base and restricting access to them based on the platforms configuration. They add an extra TLS message that notifies the remote peer about a configuration change. Furthermore, they define an additional TLS extension (*state_change_extension*) that carries the encrypted configuration information. This information is exchanged between the communication partners in case the configuration changes. However, this approach requires extra messages and extra components to report the configuration state changes through the secure channel. Moreover, they do not create this proof inside the TPM and, therefore, do not have an implicit proof of a change in the TLS protocol.

Stumpf et. al. present a concept that binds a Diffie-Hellman key to a platform configuration [Stumpf 2006]. In their approach, they incorporate a public Diffie-Hellman key in the TPM_Quote as external data. However, their concept is susceptible to relay attacks as this approach is not resistant to man-in-the-middle attacks during the remote attestation process and they do not address the problem of configuration change reporting.

2 Secure Platform Configuration Change Reporting

In this section, we give a brief summary of our platform configuration reporting approach as discussed in [Dietrich 2008] which uses a modified message authentication code calculation process. In common TLS implementations, the data that is sent through the secure channel is split into separated records. For each of these records, an integrity protection value is calculated using the record data and a secret key - the mac-key - as input. The mac-key is only known to the local and the remote platform, which provides authentication and integrity protection for the transmitted data records. The mac-key derived by the TLS stack is sent to the TPM that takes the key as input and finally computes $TLS_{write\_mac\_final} = h(TLS_{write\_mac} || PCR_{hash} || s_\nu)$ [1] (see Figure 1).

Every record that is sent to the TPM causes the TPM to re-calculate the mac-key. The HMAC value of the record is then computed from the mac-key, the record data and a sequence counter: $MAC_{record} = HMAC(mac\_key || record_{data} || sequence\_number)$. This process is done on both endpoints, the remote and the local TLS endpoint. Consequently, both endpoints have the same key material and are able to sign and verify the symmetric signature on the TLS record data.

One can easily see that every change in the platform configuration directly affects the mac-key and, consequently, the value of the hmac value of the TLS records.

This modified procedure for deriving the mac-key has the following consequences: In case of a configuration change, a remote platform will not be able to verify the hmac [Menezes 1997] values created with this key because it still uses the key that was derived from the previous configuration.

[1] S_ν is a shared secret between local and remote host that provides a proof that the mac-key calculation was done inside a TPM. See [Dietrich 2008] for details how this value is exchanged between the platforms and installed in the TPMs.

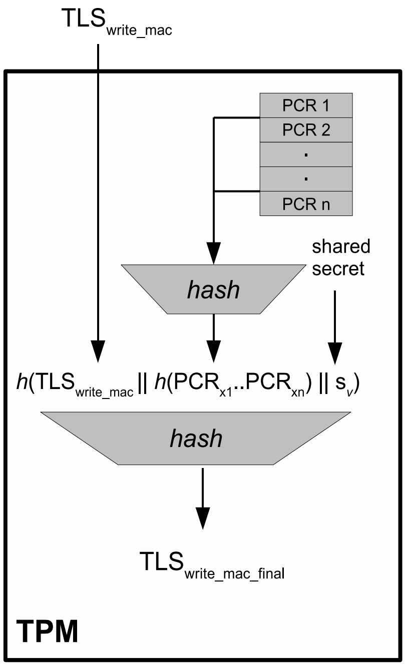

Figure 1: hmac Key Derivation

If the hmac cannot be verified, that can mean three things. First, the transmitted records have been modified during transmission. Second, the configuration of the remote platform has changed since the last received record and third, the hmac computation was done outside of the TPM.

In these cases, the verifying peer has to react to this event, which means that it can simply send a TLS *alert* message and close the connection or renegotiate new session parameters. If the newly negotiated parameters and the new configuration are valid, the secure channel remains open, otherwise the channel is closed.

Furthermore, the derivation function includes a shared secret (s_ν) that is only known to the TPM of the sending platform and the TLS stack of the receiving platform. With this procedure, the sending platform can prove that the mac-key computiation was done inside its TPM.

2.1 Alternative MAC Key Derivation

In our original approach, we used the mac-key from the standard TLS key derivation process to further derive the final mac-key inside the TPM. A different approach could be to perform the common key derivation directly inside the TPM instead of using the output of this function. The TLS specification defines this derivation process as

$$key = PRF(master\ secret, "key\ expansion", random_{server} + random_{client}). \tag{1}$$

The mac-key is derived from a master secret (MS) which is derived from a pre-master secret. Details about this process can be found in [Dierks and Allen 1999]. The session key and the mac-key material are derived from the MS. The key material is computed as follows:$key = PRF(master\ secret, "key\ expansion", server_{random} + client_{random})$ where the pseudo random function (PRF) is defined as: $PRF(secret, label, seed) = MD5(S_l, label + seed) \oplus SHA1(S_r, label + seed)$. (where S_l denotes the leftmost bits of the master secret and S_r denotes the rightmost bits of the master secret).

The TPM could then compute the mac-key via

$$\begin{aligned} key =&PRF(master\ secret, "key\ expansion", random_{server} + random_{client}, \\ &h(PCR_{x_1}..PCR_{x_n})). \end{aligned} \tag{2}$$

In this case, the PRF must be modified that way that it includes the hash of the PCR registers. This could be achieved in two ways: The hash of the PCR registers could either be computed by

$$\begin{aligned} PRF(secret, label, seed, h(PCRs) =&MD5(S_1 \Big|\Big| label + seed \Big|\Big| h(PCRs)) \oplus \\ &SHA1(S_2 \Big|\Big| label + seed \Big|\Big| h(PCRs)). \end{aligned} \tag{3}$$

or

$$\begin{aligned} PRF(secret, label, seed, PCRs) =&h(MD5(S_1 \Big|\Big| label + seed) \oplus \\ &SHA1(S_2 \Big|\Big| label + seed) \Big|\Big| PCR_{x_1}...PCR_{x_n} \Big|\Big| s_\nu). \end{aligned} \tag{4}$$

The first PRF requires the computation of an additional hash operation whereas in the second PRF the content of the PCRs is hashed together with secret, label and seed. For our approach, we chose the second approach as it is the easiest one to implement and requires only one hash calculation to be performed when a PCR is extended.

In contrast to our original approach, we derive and store the session key inside the TPM. Storing the key obsoletes the requirement of sending the mac-key computed in the TLS stack to the TPM which reduces the communication overhead as well as the processing latency when computing the final mac-key. Moreover, we use a counter inside the TPM to compute the TLS record sequence number. This counter is increased everytime a new record is sent to the TPM.

Although this approach requires an additional hash engine inside the TPM, we can benefit from this fact as it allows us to use MD5 based ciphersuites in addition to the SHA1 based ciphersuites.

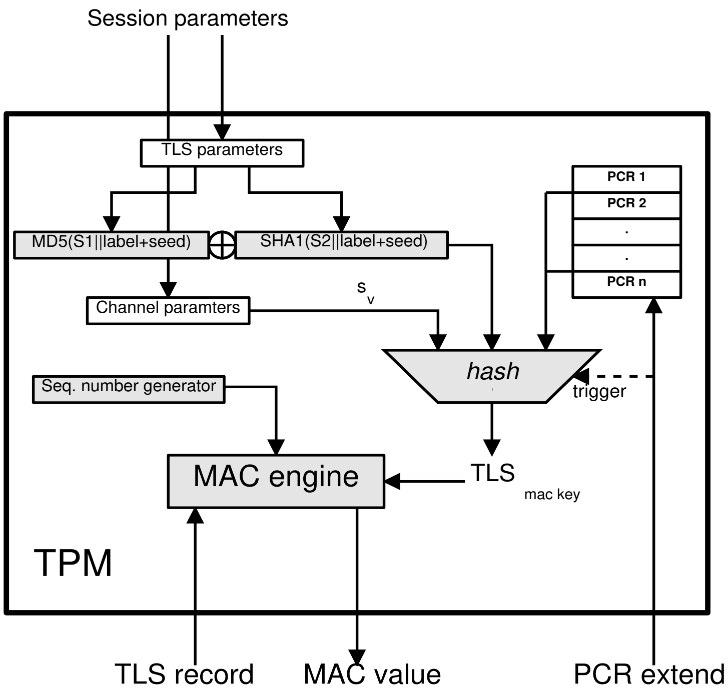

Figure 2: Alternative hmac Key Derivation

2.2 Session Parameter Setup

In this section, the required initialization and setup steps for the TPM and the TLS stack are discussed. For the first approach, the local TPM requires a TLS_{write_mac} which can be obtained from the TLS stack key derivation process. The local TLS stack, however, requires the initial TPM_QUOTE_INFO structure and the AIK certificate from the remote platform. The certificate is required to verify the authenticity of the remote TPM by validating the AIK certificate, whereas TPM_QUOTE_INFO contains the current configuration. This configuration information is required by the stack to compute $TLS_{read\_mac\_final}$. For this calculation, the TLS stack also requires TLS_{read_mac} which can additionally be obtained as described in [Rescorla 2001]. Consequently, TPM_QUOTE_INFO and the AIK certificate must be exchanged with the remote platform before $TLS_{write\_mac\_final}$ and $TLS_{read\_mac\_final}$ can be computed. Moreover, both platforms require a shared secret s_ν that has to be exchanged before the hmac computation can start. This is also true for the second approach. Moreover, this second approach has to initialize the TPM prior starting the TLS session. This requires the stack to send the master secret, client and server random to the TPM, as well.

2.3 Binding the TLS Channels to different Processes

The approach we have discussed in the previous sections focuses on single connections. On modern platforms, concurrent connections are often used - not only on server systems. Concurrent connections can be initiated by e.g. different processes within a single machine with one single TPM or, in virtualized environments, with many different virtual TPMs. In any case, an application must not be able to read or use s_ν from a different process under the assumption that the process isolation mechanism of the underlying operating system works flawlessly.

In order to achieve this access protection, the TPM must provide an authentication mechanism that can be used by each TLS stack instance. In our approach and prototypes, we use the standard authentication mechanisms and protocols e,g, OIAP, OSAP provided by the TPM specification.

2.4 TLS Client Authentication with TPMs

In common client authentication scenarios, the client credentials allowing access, for example to a company's VPN, are stored on the platform's disk where they are subject to manipulation or even theft. The protection of these credentials solely relies on the security services and protection mechanisms provided by the platform. Having a security element like a TPM on the platform, it seems reasonable to use the protection mechanism provided by TPMs. A simple approach could be using the bind mechanism of the TPM to bind the credentials to a certain platform state. However, that does not fully guarantee the nondisclosure of these credentials as they are decrypted and used in plain for the authentication process - TPMs provide shielded locations and protected capabilites which can provide a much better protection. Consequently, it is reasonable to store and use the credentials inside the TPM.

Basically, client authentication in TLS works as follows: During the handshake, the server requests a certificate from the client. The client sends a *Certificate Verify* message including the signature on the hash of the handshake messages sent so-far. The server is then able to verify the signature with the previously sent certificate and by comparing the hash value of the client messages recorded by the client with the hash value of the messages received by the client, thereby authenticating the client.

Instead of creating the signature within the TLS stack, the signature could be created inside the TPM, therewith providing a strong protection of the signature key. Although this method can also be achieved with common smart cards, the TPM and its DRM capabilities offer features for managing these credentials, which is hard to achieve with smart cards. Assuming a running trusted platform, network administrators could remotely distribute or delete the client credentials among their managed platforms. Furthermore, the use of the client credentials could be limited to a certain amount of uses simply by locking them to a monotonic counter. Moreover, authorization to use the credentials could be bound to a certain platform configuration, thereby preventing a succesfull connection with invalid configured platforms.

A proof-of-concept implementation of the client authentication commands for TPMs is discussed in the following paragraph.

2.5 TLS Client Authentication Commands

For our TLS supported client authentication prototype, we added the following commands to the TPM emulator:

- TPM_LoadClientAuthKey(TPM_KEY_HANDLE parent key handle, TPM_KEY tls_aut_key). This method loads the specified key into the TPM and returns a TPM_KEY_HANDLE structure.
- TPM_UpdateClientMsg(UINT32 client message size, BYTE[] client message). This command sends a single hash of a handshake message to the TPM. Consequently, this command is subsequently invoked for every message sent by the client.
- TPM_SignClientMsg(TPM_KEY_HANDLE tls_auth_key_handle). This command instructs the TPM to compute the signature over the handshake messages with the given key addressed by the key handle.

In order to use these commands, the TLS stack has to establish an OIAP session with the TPM. Furthermore, this approach does not require any modifications of the verification module of the TLS stack which is responsible for verifying the signature.

2.6 TPM Enhancements

The proposed mac-key computation algorithms require modifications of the current TPM specification. The TPM has to support a few additional commands which is the key derivation process as discussed in Section 2 and Section 2.1.

In order to implement our original concept, the TPM has to support the $TLS_{write\_mac\_final}$ key derivation process (as discussed in Section 2). The process can easily be integrated in existing TPM implementations as it only requires two additional hash calculations. As hash algorithms are available on common TPMs, new cryptographic algorithms are not required as long as the supported ciphersuites are limited to ciphersuites that use sha1 for integrity protection.

For our proof-of-concept prototype, we added the commands listed as follows:

- TPM_LoadSharedSecret(TPM_KEY_HANDLE aik key handle, UINT32 encrypted shared secret size, BYTE[] encrypted shared secret). This command initializes a new TLS session and downloads the encrypted secret s_ν from the remote platform and the handle to the AIK it was encrypted with. The response of the command is a shared secret of the type TPM_KEY_HANDLE, a handle to the shared secret within the TPM.

- TPM_ComputeMac(TPM_KEY_HANDLE shared secret handle, UINT32 keySize, BYTE[] HMAC key, TPM_PCR_SELECTION pcrs, UINT32 dataSize, BYTE[] data). This command instructs the TPM to compute the derived mac-key and mac-value of the provided data, including the shared secret adressed by *shared secret handle*. The result of this command is the TPM_DIGEST mac-value. When a session is closed, the shared secret should be deleted from the TPM.
- TPM_ClearSharedSecret(TPM_KEY_HANDLE shared secret handle) This command forces the TPM to clear the shared secret adressed by *shared secret handle*.

For our analysis, we implemented these commands as authorized as well as unauthorized commands. However, for a maximum level of security, applications should use the authorized set of commands.

For our second approach, we additionally implemented the following commands:

- TPM_SetSessionParameters(UINT32 master secret size, BYTE[] master secret, UINT32 client rnd size, BYTE[] client random, UINT32 server rnd size, BYTE[] server random, UINT32 enc shared secret size, BYTE[] encrypted shared secret, TPM_PCR_SELECTION pcrs). This command initialized a new TLS session by loading the master secret, the shared secret s_ν, client and server random and the PCR selection into the TPM. Moreover, this method resets the sequence counter.
- TPM_ComputeMac(UINT32 dataSize, BYTE[] data). This method computes and returns the mac value for the provided data.

3 Discussion

Our experiments showed that both approaches are feasible. However, the second approach requires some additional modifications of the TPM in comparison with the original approach. Moreover, the second approach requires more session specific data and additional algorithms to be stored inside the TPM. To be more detailed, the differences are: The TPM has to receive additional key derivation parameters i.e. master secret, client and server random and has to store it in the TPM every time a new session is initialized. Moreover, the TPM has to support the modified pseudo random function including the additional MD5 hash algorithm. Implementing additional hash algorithms and a PRF into the TPM reduces the flexibility of the original approach, where most of the algorithms are implemented in software. This software can easily be replaced in future TLS implementations when new algorithms are defined in the specifications. Nevertheless, due to the fewer data that has to be transferred to the TPM, the second approach has a performance advantage against the first approach.

From the security's point of view, both approaches provide a high level of security. All computations are done inside a tamper-proofed device. Moreover, manipulatation of the session parameters requires authetication which is employed by the TPMs internal authentication mechanisms and protocols (i.e. OIAP). A modification of the platforms software configuration results in a modification of one or more PCRs. The final session keys are derived from the current stage of PCRs, hence, the configuration change is reflected in the MAC of the TLS records. These records are encrypted so that manipulations are hard to apply. A possible attack could be to modify the encrypted data stream where an adversary could flip certain bits of the datastream. The attack would result in a denial of service as the receiving platform would have to reject the incoming packets and would have to negotiate new session parameters. This attack is not special to our approach as it is also a threat to current TLS connections. Hardware attacks on TPMs are not considered in our discussion as they are not part of the TCG's threat model.

3.1 A Note on Configuration Reporting with Mobile TPMs

Our concept for configuration change reporting is also applicable to mobile TPMs. The major difference between common TPMs and mobile TPMs (MTMs) is that, depending on the security features provided by the mobile platform, MTMs can be implemented only in software [Dietrich and Winter (2009)]. This property makes them an ideal basis for our approach as the mac computation of the TLS records is a resource demanding task as all records have to be sent to the TPM prior to mac calculation. This calculation can efficiently be achieved in software and, therefore, in software based MTMs.

4 Conclusion and Future Work

In this paper, we discussed two alternatives to provide a configuration change reporting mechanism for trusted computing enabled platforms. Our proposals showed how the mechanism for protecting TLS records can be used to provide a proof for configuration changes by calculating the mac inside the TPM. While the first approach is rather easy to integrate into common TPMs, the second one is more complex but shows a better runtime performance. Moreover, we proposed a method how a TPM could be used for TLS client authentication, thereby providing stronger protection of the client credentials as similar software implementations and without the requirement of additional hardware like smart cards.

As the TPM specification does not include the required features for this approach we had to define additional commands and modify existing TPM implementations. Our concepts are currently available only in software - nevertheless, they provide a valuable extension to software based TPM implementations. Such implementations include virtual TPMs, like the one used in virtualization techniques like XEN [Williams 2007] or para-virtualization technologies and mobile TPMs [TCG-MPWG (2007)].

The current prototype only supports the creation of authentication keys inside the TPM. Future versions could include the installation of externally created authentication credentials. Moreover, a more sophisticated authentication management than the existing one could be integrated.

References

[Dierks and Allen 1999] T. Dierks and C. Allen, Internet Engineering Task Force,"*The TLS Protocol Version 1.1*", RFC 4346 (Proposed Standard), 1999.

[Dietrich 2008] Kurt Dietrich, "*A Secure and Reliable Platform Configuration Change Reporting Mechanism for Trusted Computing Enhanced Secure Channels*", International Conference for Young Computer Scientists, p. 2137-2142, IEEE Computer Society, Los Alamitos, CA, USA, ISBN: 978-0-7695-3398-8, 2008

[Dietrich and Winter (2009)] Dietrich, K., Winter, J.: "Implementation aspects of mobile and embedded trusted computing."; L. Chen, C. J. Mitchell, A. Martin, eds., TRUST; volume 5471 of Lecture Notes in Computer Science; 29–44; Springer, 2009.

[Galbraith and Paterson (2008)] Galbraith, S. D., Paterson, K. G., eds.: Pairing-Based Cryptography - Pairing 2008, Second International Conference, Egham, UK, September 1-3, 2008. Proceedings; volume 5209 of Lecture Notes in Computer Science; Springer, 2008.

[Gasmi 2007] Yacine Gasmi and Ahmad-Reza Sadeghi and Patrick Stewin and Martin Unger and N. Asokan, "*Beyond secure channels*", STC '07: Proceedings of the 2007 ACM workshop on Scalable trusted computing, pages 30–40, 2007

[Goldman, Perez, Sailer 2006] Kenneth Goldman and Ronald Perez and Reiner Sailer, "*Linking remote attestation to secure tunnel endpoints*", STC '06: Proceedings of the first ACM workshop on Scalable trusted computing, pages 21–24, 2006

[JSR 139 Process (2004)] SUN Community Process JSR 139: "*J2ME(TM) Connected Limited Device Configuration (CLDC) Specification 1.1 Final Release*"; Specification available at: http://jcp.org/aboutJava/communityprocess/final/jsr139/index.html (2004).

[Kesselman (2000)] Kesselman, S. J.: Java Platform Performance: Strategies and Tactics; Addison Wesley, 2000.

[Krall (1998)] Krall, A.: "Efficient javavm just-in-time compilation"; International Conference on Parallel Architectures and Compilation Techniques; 205–212; 1998.

[Menezes 1997] Menezes, A. J. , Van Oorschot, P. C., Vanstone, S. A.: Handbook of applied cryptography; CRC Press series on discrete mathematics and its applications; CRC Press, Boca Raton, c1997; includes bibliographical references (p. 703-754) and index.

[Ortiz (2002)] Ortiz, E.: "Introduction to ota application provisioning"; Technical report; SUN Developer Network (2002); article available at: http://developers.sun.com/mobility/midp/articles/ota/.

[Rescorla 2001] Eric Rescorla, "*SSL and TLS: designing and building secure systems*",Addison-Wesley, ISBN:0201615983, 2001

[Stumpf 2006] F. Stumpf and O. Tafreschi and P. Röder and C. Eckert, "*A Robust Integrity Reporting Protocol for Remote Attestation*", Second Workshop on Advances in Trusted Computing (WATC '06 Fall), Tokyo, 2006

[SUN Microsystems (1997)] SUN Microsystems: "Java Native Interface Specification"; Available online at: http://java.sun.com/j2se/1.5.0/docs/guide/jni/spec/jniTOC.html (1997).

[Sun Microsystems (2002)] Sun Microsystems: "K Native Interface (KNI)"; Technical report; 4150 Network Circle Santa Clara, California 95054 (2002).

[TCG-MPWG (2007)] TCG-Mobile-Phone-Working-Group: "*TCG Mobile Trusted Module Sepecification Version 1 rev. 1.0*"; Specification available online at: https://www.trustedcomputinggroup.org/specs/mobilephone/tcg-mobile-trusted-modul e-1.0.pdf (2007).

[TCG 2007] Trusted Computing Group - TPM Working Group "TPM 1.2 Main Specification - Part 3 Commands", Specification available online at: https://www.trustedcomputinggroup.org/specs/TPM/mainP3Commandsrev103.zip, 2007

[Yellin, Lindholm (1999)] Yellin Frank, Lindholm Tim: "*The Java Virtual Machine Specification Second Edition*", Available online at: http://java.sun.com/docs/books/jvms/second_edition/html/VMSpecTOC.doc.html (1999).

[Williams 2007] David E. Williams and Juan R. Garcia, "*Virtualization with Xen: including XenEnterprise, XenServer, and XenExpress*", Syngress, ISBN:9781597491679, 2007