

# Hades – Hardware Assisted Document Security

Patrick Röder\*, Frederic Stumpf\*\*, Ralf Grewe, Claudia Eckert

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

**Abstract.** In today’s business world, electronic documents have nearly replaced paper documents. As electronic documents are easier to steal and to replicate, we have to deal with the resulting risks. Consequently, many different approaches were developed to protect electronic documents. However, these software based solutions cannot assure that their mechanisms nor the underlying platform have not been compromised, e.g., by an attacker or by malware. Moreover, documents are usually encrypted, and the corresponding keys are protected from unauthorized access by software mechanisms, which can be circumvented by a skilled attacker. To overcome these shortcomings, we present a hardware based approach using the Trusted Platform Module (TPM) to detect and handle a compromised platform and to protect the encryption keys with the provided hardware mechanisms. These hardware mechanisms also bind the documents to the machine they are stored on, which ensures that these documents can only be accessed on that machine.

## 1 Introduction

In the modern business world, more and more electronic documents are used to replace paper documents for higher flexibility and efficiency. Electronic documents are easier to create, modify and share with others, additionally they offer new possibilities such as machine-supported searching and reusing of content. However, using electronic documents introduces new kinds of risks and vulnerabilities, such as data theft and industry espionage, since electronic documents are easier to steal and to replicate than paper documents. According to a survey of the FBI [GLLR05], information theft and unauthorized access alone are responsible for a major part of damages caused by computer crime, accounting for a financial loss of over \$62 million in 2005 in the United States.

According to [GLLR05] about 46% of the successful attacks are performed by inside users who have legitimate access to a system. This type of attacker

---

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

\*\* The author is supported by the German Federal Ministry of Education and Research under grant 01AK706C, project *TrustCaps*.

is more dangerous than an outside attacker, since they can misuse their legitimate authorizations. Besides that, documents can also be sent by accident to an unauthorized recipient, which must be prevented as well.

Therefore, several approaches [Mic03,IGD01,YC04] have been developed to prevent the theft of confidential information. Typically, these approaches protect documents from unauthorized access by encrypting them and storing the corresponding keys in a secret place on the same computer. This method achieves security by obscurity and can be broken by a skilled attacker. This attacker could retrieve the keys to decrypt the documents, which highlights the importance of protecting these keys.

Additionally, these approaches assume that the enforcement mechanisms have not been modified and are working correctly. These mechanisms assure that the security policies [Sch00], which define rules to permit or deny data access, are enforced correctly. Disabling these mechanisms either by modifying the underlying operating system or the enforcement software itself can result in a loss of confidentiality of the protected documents. The modification could be performed by an inside attacker or an outside attacker, e.g., by injecting malware. A very critical attack of this type is presented in [KWV<sup>+</sup>06], which shows that malware can be disguised perfectly in many cases. Therefore it is essential to detect unauthorized modifications of the software which performs policy enforcement.

When documents are shared, they are often transferred over a network. Therefore, the integrity of a remote system must be verified as well before sending confidential documents to a potentially compromised system. Otherwise a security incident could occur.

In this paper, we present a client-server approach which overcomes the shortcomings outlined above by using the methods and principles offered by the Trusted Computing Group (TCG) [Tru06]. We use the remote platform attestation to make sure that the system of the recipient is not compromised. This mechanism assures the server that the client is in a trustworthy state. To prevent theft or leakage of confidential documents by modifying the local enforcement mechanisms, we seal all locally stored documents to ensure that these documents can only be accessed if the system is in the same trustworthy state as at the time of sealing. The tamper evident Trusted Platform Module protects all cryptographic keys to make unauthorized access to these keys much more difficult than protecting them only with software measures.

The remainder of this paper is organized as follows: We provide some background information on the TCG concepts in section 2. Section 3 discusses related work, whereas section 4 explains our approach followed by its evaluation in section 5. We conclude in section 6 and discuss future work in section 7.

## 2 Background on TCG mechanisms

This section gives an overview of the mechanisms described by the TCG. For a more detailed description, we refer to the TPM specification [Gro06] or [BCP<sup>+</sup>03].

The core of the TCG mechanisms is the Trusted Platform Module, which is basically a smart card 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.

The TPM can create and store cryptographic keys, both symmetric and asymmetric. These keys can either be 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 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 so that they must not be used outside the TPM. Moreover, it can measure the integrity of the software components which are involved in the boot process and store the corresponding hash values in secure hardware registers, which are called *Platform Configuration Registers* (PCRs). The TCG assumes that a trusted operating system<sup>1</sup> measures the hash value of every executable started after the boot process. Each hash value is concatenated with the previous value of a specific PCR, hashed and then stored again. The trusted OS maintains a journal of the started processes to enable a later verification of the combined hash values.

The PCR values can be used together with the journal to attest the platform's state to a remote party, which is called *Remote Platform Attestation* (RPA). To make sure that these values are authentic, they are signed with a non-migratable TPM key. The remote platform can compare these values with reference values to see whether the platform is in a trustworthy state. As a result, the term *system state* in this context describes which BIOS, system kernel and user processes were executed on the system up to the RPA, but not the runtime state or the processed data.

Furthermore, the TPM can encrypt local data with an encryption key which is computed from the current platform state. This concept is referred to as *sealing*. As a result, the local data can only be decrypted if the platform state has not been changed. Since many attacks are based on the modification of the local system time, the TPM offers a secure tick counter to have a trusted time source.

### 3 Related work

This work is related to *Digital Rights Management* (DRM), which is an approach to prevent illegal distribution of paid content. In contrast to DRM, this work focuses on the protection of confidential documents. To express the similarity to

<sup>1</sup> A trusted OS is not part of the TCG specification. For a description of a trusted OS confer [GRB03].

DRM, we refer to our approach as *Information Rights Management (IRM)*. In the following, we present three approaches which we think are representative in this field.

CIPRESS [IGD01] builds a local quarantine zone by encrypting all local files with a machine specific key, which can optionally be stored on a tamper resistant hardware module, the Elkey crypto board [CI06]. CIPRESS offers only coarse grained access rights, since there is no access control on application level. Moreover, the security of CIPRESS is based on the assumption that the client system is not compromised, since it offers no mechanism to detect or to handle a compromised system state.

Microsoft's Rights Management Services [Mic03] offer much more finely grained access rights compared to CIPRESS. These access rights are embedded in a signed usage license, are transferred together with the document, and are enforced by a local client software, which is assumed to be not compromised. Again, an unauthorized modification of the client system is neither detected nor handled. Moreover, the encryption keys are stored together with the encrypted documents without further protection.

In the Display-Only File Server (DOFS) architecture [YC04], all documents remain on the server and are accessed by executing the corresponding applications on the server. Only the display content of these applications is transferred to the client using windows terminal services. DOFS offers only coarsely grained access rights, since there is no access control on the application level. The approach is limited to usage scenarios with a permanent connection to the DOFS server. Moreover, a modified client is neither detected nor handled.

Besides these three approaches, there are many others such as [Inc04] or [Aut01]. All of these lack mechanisms to detect a compromised system state, which could lead to an unauthorized information transfer. The mechanisms specified by the Trusted Computing Group (TCG) provide a possible solution for the described problem.

In the following, we list some related research projects which make use of the TCG mechanisms to detect and handle unauthorized system modifications. We use some of their results for our architecture *Hades*. TrustedGRUB [App06b] is a bootloader that uses the TPM to extend integrity measurements to the bootloader and the OS kernel. Perseus [PRS<sup>+</sup>01] and Turaya [LP06] are microkernel-based trusted operating systems that can be combined with TrustedGRUB to establish a trusted computing base (for details, cf. [BCP<sup>+</sup>03]). The Bear/Enforcer project [MSWM03] includes a TPM-enabled Linux Security Module (LSM) to compare hash values of applications with reference values. A similar approach is used in the Integrity Measurement Architecture (IMA) [SZJvD04] developed by IBM, which performs integrity measurements for all started processes to enable a remote attestation. The concepts provided in [Rei04] are very similar to Hades, since they determine access rights of the client depending on the result of the RPA. In contrast to Hades, this work does not focus on Information Rights Management.

In summary, the software approaches presented above suffer from different shortcomings, which could be avoided by using the TCG mechanisms. Unfortunately, the current work in the area of trusted computing is focused on the operating system level. Additional approaches based on the work at the OS level are required to use the TCG mechanisms more specifically. In closure we present an application level TCG-based approach for IRM in the following section.

## 4 Hades

In this section, we present our approach to Information Rights Management called *Hades*, which is an abbreviation for *Hardware Assisted Document Security*. We have also chosen the name in reference to the Greek god of the underworld, who strictly forbade his subjects to leave his domain and would become quite enraged when anyone tried to steal his prey from him.

We use the TCG mechanisms on the application level, since this enables us to adapt these mechanisms to the requirements of the application. For example, we can seal only confidential local files selectively, instead of sealing all files, which would result in an enormous performance-overhead.

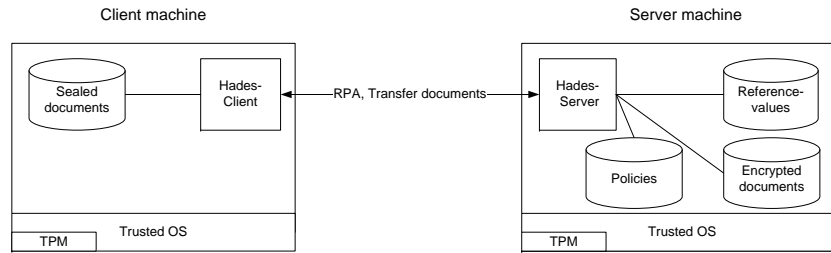
However, we cannot implement integrity measurements of the started processes on the application level, which is required for the RPA. For this purpose, we need an operating system such as the Integrity Measurement Architecture from IBM to perform these measurements after the boot process. Every time a new process is started, the hash value of the corresponding binary is written to a Platform Configuration Register of the TPM chip. Whenever the server requests an RPA, the client sends these values to the server, which verifies whether these values are listed on a white-list. Thus, the state of the machine captured by an RPA depends on the applications that were started on the client.

In contrast, the state definition which is used for sealing is different. This state definition consists of the kernel, the BIOS and the Hades client software. Therefore, the modification of other software does not affect the unsealing of the documents.

### 4.1 Architecture overview

We use a client-server architecture with a centralized server in our approach, because a central server is easier to protect and to monitor compared to a decentralized system. In addition, keeping the policies in a consistent state is a less complex task in a centralized system. On the downside, a central server is more critical towards failures and attacks, since the entire system depends on it. The components of our system are illustrated in figure 1 and are described in the following.

The server stores the policies (see section 4.2) as well as the documents, which are initially created locally on the client and then uploaded to the server to share them with other users. To reduce the complexity of the system, the documents can not be modified directly on the server, instead they must be



**Fig. 1.** Components of the Hades architecture

retrieved from the server and modified on the client. Before transmitting the document to the client, the server checks whether the client is authorized to access the requested document. After that, the client enforces the embedded policy of a document. Documents can not be transferred directly from client to client, instead all documents must be sent to and retrieved from the server. This offers a higher level of security, because controlling the information flow at a central point is less complex and therefore less error-prone than a decentralized system.

## 4.2 Policies

We use the access matrix model [GD72] to model access rights for the documents, since this model is easy to implement and widely used. Note that our architecture is not limited to the matrix model, any other model such as RBAC can be used as well. The owner of a document can define for each subject, referenced by a unique identifier, whether he or she can view, print, edit or remotely store documents. Additionally, the copy and paste commands can be disabled selectively to prohibit information transfer on this channel. In some cases it can be permitted to have copy and paste enabled to offer a higher level of usability. To prevent information leakage, we deny copy and paste to applications other than our client, which restricts information flow to documents controlled by our system. As a consequence, protected information can not easily be transferred outside the domain of our architecture.

A unique identifier is assigned to every document upon creation to make documents independent of their local file names. This identifier is integrated into the policy to link the policy to a specific document. When such a document is transferred to a client, a copy of its policy is embedded into the document. This enables users to work without a connection to the server, e.g., when a user is mobile. The embedded policy is signed with the servers TPM to be able to detect unauthorized modifications of the policy. The signature of the policy is verified when a document is processed by the client. If this verification fails, access to the document is denied. To keep the policies consistent, we only allow the owner of a document to edit the master copy of the policy on the server.

The client software checks periodically for updates of the policies when a connection to the server is available. This mechanism propagates changed policies to all distributed versions of a document. For each document, its owner can specify a maximum offline time, which is the maximum time a document can be accessed without checking for a policy update. This is relevant when there is no connection to the server and the policies can not be checked for an update. If the maximum time has elapsed and the policies were not updated, further access to this document is denied. To prevent bypassing this mechanism by modifying the local system time, we specify Hades to use the secure tick counter of the TPM as a trusted source for relative time.

### 4.3 Client-Server communication

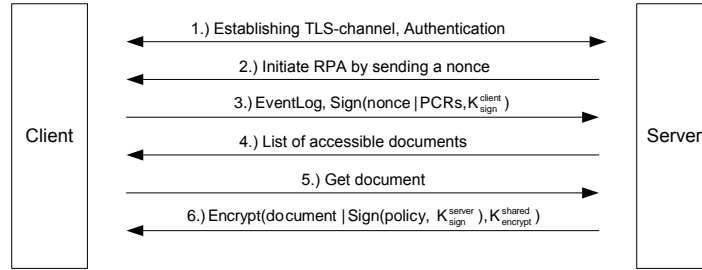
The communication between client and server is protected by TLS [IETF99] with mutual authentication via certificates, which greatly reduces the risk of spoofing attacks. The certificates require that client and server use a common Public Key Infrastructure (PKI). To achieve a higher level of security compared to software solutions, we additionally use the security mechanisms of the TPM. Therefore, a migratable symmetric key ( $K_{encrypt}^{shared}$ ) is generated in the client's TPM and transferred to the server's TPM whenever a new client is added to the system. We refer to this pre-shared key as the *client-key*, which is used to have a further TPM secured encryption in addition to the TLS protocol. In contrast to the TLS certificates, the client-key is more difficult to extract from the client by an attacker, e.g., to spoof a client, because it is stored in the protected domain of the TPM.

Client and server interact with each other using three different protocols. The first protocol is used to retrieve a document from the server. A second protocol checks whether there are policy updates on the server for that document. And finally, a third protocol is used to store these documents on the server.

Subsequently, we describe the protocol for document retrieval in detail, since it is the most complex and the other protocols are conceptually similar. The clients state is checked with an RPA to make sure that it is trustworthy before sending a confidential document to the client. We do not attest the servers state, because we assume the server to be trustworthy due to its higher level of protection. Moreover, the server runs only a small set of software, which reduces the system complexity and therefore the number of possible software vulnerabilities. The protocol for document retrieval is illustrated in figure 2 and takes several steps:

1. A mutually authenticated TLS-connection is established.
2. The server initiates an RPA by sending a nonce to the client.
3. The client answers the request by signing the nonce and the current PCR values with his TPM signing key ( $K_{sign}^{client}$ ). Additionally, the client sends the journal of the started processes to the server, which verifies the clients state.
4. The server sends the list of accessible documents to the client.
5. The client requests a document from the list.

6. The server signs the document's policy with a non-migratable TPM signing key ( $K_{sign}^{server}$ ). Next, the server encrypts the document and the signed policy with the symmetric client-key ( $K_{encrypt}^{shared}$ ) and sends this encrypted package to the client.



**Fig. 2.** Protocol for document retrieval

#### 4.4 Hades – Server

All documents on the server are encrypted with a non-migratable key by the TPM to prevent unauthorized information gain. This key is generated by the TPM when the server is initially set up. Before sending a document, the server performs an RPA of the client, decrypts the document and then re-encrypts it with the client-key, which is protected by the server's TPM. The server also maintains the master copy of each documents policy, therefore the clients must update their local policies periodically. The corresponding network communication is secured by the TLS protocol.

The server can generate a view of all accessible documents based on the clients authorizations, which makes the selection of accessible documents easier.

#### 4.5 Hades – Client

The client enforces the cached version of a document's policy. Therefore, the client checks for each mode of access (viewing, modifying, ...) whether the user has the required privileges. As a result, specific actions are allowed or denied depending on the policy.

Documents need not to be stored on hard disk after they were retrieved from the server. Instead, they can be viewed and modified in the clients memory, which prevents possible unauthorized information gain by accessing local data. Furthermore, documents can be stored on the client to enable access for later situations without a network connection. Documents are sealed when they are stored on the client to bind them to the machine and its specific state. This

reduces the risk of information gain by unauthorized access to locally stored data, because the document can only be decrypted on the corresponding machine in its unmodified state. The editor immediately checks for a policy update after opening a sealed document to ensure that the policy is up to date. The access to a document is controlled by the client software in a non-compromised system state. An attacker must modify these control mechanisms to access the documents, but the modification of these mechanisms would result in a changed system state. Thus, even if an attacker modifies the control mechanisms, he can not access the documents, because the unsealing of the documents is only possible in the uncompromised state.

#### 4.6 Implementation

This section presents our prototype of Hades. All parts of our implementation, except for the TPM driver, are implemented in Java. We modified the TPM driver developed by the University of Bochum [App06a] to access the TPM using the Java Native Interface (JNI). The owner password of the TPM is stored in the client software, which must be executed with root privileges to access the TPM. A Hades user should have no root privileges, but can use our client with the *setuid* mechanism, which enables non-root users to run software with root permissions. Before using the client software, the user must perform a password authentication, which is also used to decrypt the client’s private key of the corresponding TLS certificate.

Figure 3 shows the Hades editor while specifying the appropriate access rights for a file before saving it on the server. The user can specify for all other system users, who are represented by the distinguished name as denoted in their certificate, whether they have access to the file and define their access rights for the file.

We used a TPM chip version 1.1b from Infineon in our implementation. The current prototype does not check the maximum offline time with a secure tick counter, because this chip version does not offer the required feature. Newer TPM chips of version 1.2 support this feature, but were not available when we started the implementation.

In our current version, we do not have a trusted operating system that performs integrity measurements on running software. Instead, we simulate an RPA by sending pre-defined reference values to the server. We are currently working on the integration of the IMA (see section 3) into our architecture to obtain measured values. The current version of our prototype runs on a Gentoo Linux with kernel version 2.6. The machine on which our prototype was tested is equipped with 512 MB of RAM and an Athlon XP2000+ CPU.

## 5 Evaluation

In this section, we evaluate our prototype and its architecture under security and performance aspects. In addition to the security features presented above,

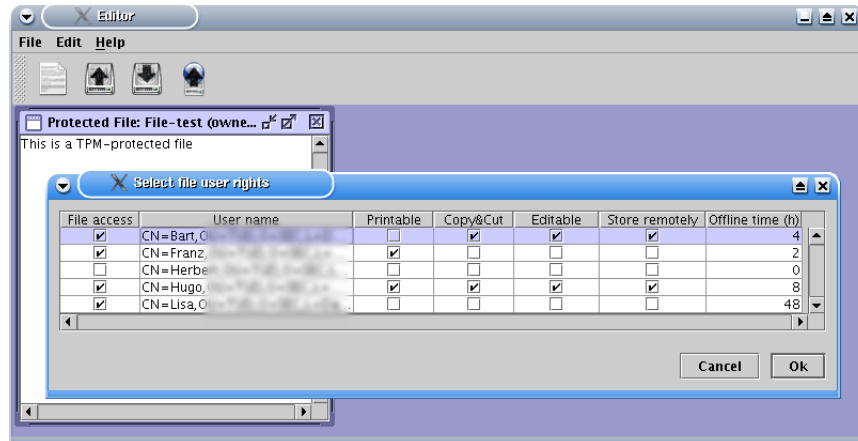


Fig. 3. Screenshot of the Hades editor

we still see some open issues. Authorized changes to the system state, like software updates, result in inaccessible documents if they are not handled, e.g., by unsealing and resealing the documents. Maintaining reference values is a tedious task and becomes even more complex if the system's software has a high level of dynamics due to frequent updates and patches. Moreover, the access to the TPM must be protected by the operating system, otherwise an attacker could use the TPM to decrypt or unseal the protected data. In our current prototype, access to the TPM requires root privileges and the owner password of the TPM. This password is hardcoded in the client software. If an attacker gains root access and retrieves the password from our software, he can use the TPM to unseal all stored documents. The access control mechanisms of the (trusted) operating system must be set up in a way that prevents this type of attacks. The modification of the operating system to bypass its control mechanisms and retrieve the hardcoded TPM password results in a different system state and therefore makes unsealing impossible. We further assume that the user of the Hades architecture has no root privileges and uses a client computer that is configured by a system administrator, who has to be trustworthy, since he has full control of the client system, its software and the corresponding reference values. This assumption also includes that the user is different from the platform owner. Analogous attacks, such as taking a picture of the screen, are very difficult to handle with technical measures. Finally, the TPM is specified to be only tamper evident and not tamper resistant, which means that physical attacks may still be possible for a skilled attacker.

Another evaluation criterion is performance. The Hades architecture encrypts all documents with a client specific key and performs a remote attestation before transferring a document to the client. As a consequence, the server must execute three asymmetric cryptographic TPM instructions (decrypt, encrypt and sign) to send a document to a client. Symmetric cryptographic instructions are less

critical than asymmetric ones, since the symmetric encryption are not performed by the TPM chip and can therefore be executed very fast. The symmetric encryption or decryption of a document with a size of 10 MB takes fractions of a second, which is sufficiently fast for typical business documents. In contrast to that, a typical asymmetric cryptographic TPM instructions such as signing the policy or decrypting a symmetric key takes about one second on average.

Opening a file from a remote server takes 3.3 seconds on average, whereas storing a document on the server takes 2.0 seconds. Thereby about more than 90% of the time is spent for TPM instructions. Even though our prototype is fast enough for many scenarios, the performance will be suitable for large business environments too when TPM chips get faster, especially when they are integrated in the CPU. Alternatively, the TPM can be omitted at the server to achieve a higher level of performance. However, without a TPM in the server one must either completely rely on the server's high degree of protection or use other mechanisms to protect the server's keys, e.g., USB key tokens.

## 6 Conclusions

In this paper, we have presented an architecture with hardware assisted protection mechanisms against unauthorized access to confidential documents. We used the features of the TPM chip to gain a higher level of security compared to pure software solutions. Attacks that are based on a compromised client state are successfully prevented by remote attestation and by sealing the locally stored documents. Documents can also be edited in memory without saving them to hard disk to avoid the risk that they are stolen from the client. If documents are saved on hard disk they are encrypted with keys that reside in the protected storage of the TPM. Therefore these keys are much more difficult to extract compared to keys saved on the client without TPM protection. Hades supports finely grained access rights, because the control is performed on the application level. The policies of our architecture are stored on the server side. Clients check these policies periodically for updates and deny further access in case of revoked access rights. To allow usage of the documents without a connection to the server, the author of a document can specify the maximum duration where no policy update is required. If the policies are not updated within the given time frame, further access is denied. Therefore, this mechanism enforces something similar to an expiration date.

## 7 Future Work

We are currently working on the extension of our architecture to perform integrity measurements during runtime, which is required to perform an RPA that inspects the system state as defined in section 2. We investigate whether we can combine existing approaches like IMA [SZJvD04] to close this gap. Moreover,

we are thinking about a solution to handle authorized modifications to the system state, e.g., BIOS updates. In this case, sealed documents must be unsealed, stored temporarily in a secure way and then resealed in the new system state.

We are also investigating other scenarios where our architecture can be applied. One such application is e-commerce, where we have to guarantee that the software of a communication partner is free of malware, which could undermine the security of the transaction process. We are currently working on a similar architecture for e-commerce in a DFG<sup>2</sup> research project called *TrustCaps*. The expiration dates of documents provided through our architecture are also important in the field of e-commerce, where privacy related information of a client may only be stored as long as they are needed for the transaction process.

## References

- App06a. Applied Data Security Group, University of Bochum. Linux TPM Driver, 2006.
- App06b. Applied Data Security Group, University of Bochum. TrustedGRUB. [http://www.prosecco.rub.de/trusted\\_grub\\_details.html](http://www.prosecco.rub.de/trusted_grub_details.html), May 2006.
- Aut01. Authentica Inc. PageRecall: The Key to Document Protection. <http://www.authentica.com/>, 2001.
- BCP<sup>+</sup>03. Boris Balacheff, Liqun Chen, Siani Pearson, David Plaquin, and Graeme Proudler. *Trusted Computing Platforms*. Hewlett-Packard Company, 2003.
- CI06. CE-Infosys. Elkey Crypto Board. [http://www.ce-infosys.com/CeiProducts\\_Elkey.asp](http://www.ce-infosys.com/CeiProducts_Elkey.asp), 2006.
- GD72. G. S. Graham and P. J. Denning. Protection - Principles and Practice. In *Proceedings of the Spring Joint Computer Reference*, volume 40, pages 417–429, 1972.
- GLLR05. Lawrence A. Gordon, Martin P. Loeb, William Lucyshyn, and Robert Richardson. 2005 CSI/FBI Computer Crime and Security Survey. Technical report, Computer Security Institute, 2005.
- GRB03. Tal Garfinkel, Mendel Rosenblum, and Dan Boneh. Flexible OS Support and Applications for Trusted Computing. In *Proceedings of the 9th Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, May 2003.
- Gro06. Trusted Computing Group. Trusted Platform Module (TPM) specifications. Technical report, <https://www.trustedcomputinggroup.org/specs/TPM>, 2006.
- IET99. IETF TLS Working Group. The TLS Protocol, RFC 2246. <http://www.ietf.org/internet-drafts/draft-ietf-tls-rfc4346-bis-00.txt>, January 1999.
- IGD01. Fraunhofer IGD. CIPRESS White Paper - An Overview of the System, Its Concepts, and Implementation in the Microsoft Windows NT Operating System Family Environment. Technical report, Februray 2001.
- Inc04. Adobe Systems Incorporated. A primer on electronic document security - How document control and digital signatures protect electronic documents. [http://www.adobe.com/security/pdfs/acrobat\\_security\\_wp.pdf](http://www.adobe.com/security/pdfs/acrobat_security_wp.pdf), November 2004.

---

<sup>2</sup> The Deutsche Forschungsgemeinschaft (German Research Foundation) is the central, self-governing research funding organisation that promotes research at universities and other publicly financed research institutions in Germany

- KWV<sup>+</sup>06. Samuel T. King, Peter M. Chen Yi-Min Wang, Chad Verbowski, Helen J. Wang, and Jacob R. Lorch. SubVirt: Implementing malware with virtual machines. In *Proceedings of the 2006 IEEE Symposium on Security and Privacy*, May 2006.
- LP06. M. Linnemann and N. Pohlmann. Die vertrauenswürdige Sicherheitsplattform Turaya. In *DACH Security 2006*. Patrick Horster, syssec Verlag, 2006.
- Mic03. Technical Overview of Windows Rights Management Services for Windows Server 2003. White paper. Technical report, Microsoft Corporation, November 2003.
- MSWM03. John Marchesini, Sean W. Smith, Omen Wild, and Rich MacDonald. Experimenting with TCPA/TCG Hardware, Or: How I Learned to Stop Worrying and Love The Bear. Technical Report TR2003-476, Department of Computer Science, Dartmouth College, December 2003.
- PRS<sup>+</sup>01. Birgit Pfitzmann, James Riordan, Christian Stübke, Michael Waidner, and Arnd Weber. The PERSEUS System Architecture. In *VIS 2001, Sicherheit in komplexen IT-Infrastrukturen*. Vieweg Verlag, 2001.
- Rei04. Reiner Sailer and Trent Jaeger and 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.
- Sch00. Fred B. Schneider. Enforceable Security Policies. *ACM Transactions on Information and System Security*, 3(1):30–50, 2000.
- 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.
- Tru06. Trusted Computing Group. <https://www.trustedcomputinggroup.org/>, 2006.
- YC04. Yang Yu and Tzi-cker Chiueh. Display-Only File Server: A Solution against Information Theft Due to Insider Attack. In *DRM '04: Proceedings of the 4th ACM workshop on Digital rights management*, pages 31–39, New York, NY, USA, 2004. ACM Press.