Implementation of a Trusted Ticket System

Andreas Leicher[1], Nicolai Kuntze[2], and Andreas U. Schmidt[3]

[1] Johann Wolfgang Goethe-Universität, Frankfurt am Main, Germany
leicher@cs.uni-frankfurt.de
[2] Fraunhofer–Institute for Secure Information Technology, Rheinstraße 75, 64295 Darmstadt, Germany
nicolai.kuntze@sit.fraunhofer.de
[3] CREATE-NET Research Center, Via alla Cascata 56/D, 38100 Trento, Italy
andreas.schmidt@create-net.org

Abstract. Trusted Computing is a security technology which enables the establishment of trust between multiple parties. Previous work showed that Trusted Computing technology can be used to build tickets, a core concept of Identity Management Systems. Relying solely on the Trusted Platform Module we will demonstrate how this technology can be used in the context of Kerberos for an implementation variant of Identity Management.

1 Introduction

Trusted Computing (TC) as defined by the Trusted Computing Group (TCG) is usually seen as a protection technology centred on single devices. Yet, viewed as a platform-neutral security infrastructure, TC offers ways to establish trust between entities that are otherwise separated by technical boundaries, e.g., different access technologies and access control structures. Not surprisingly, some concepts of TC are rather similar to Identity management (IDM) and federation.

In previous work we have presented a concept for how to use TC within ticket systems such as Kerberos [1], and have also shown that identity federation between different provider domains can be supported by TC [9]. The present paper reports on the progress in this field, in particular a concrete realisation and integration in an existing authentication system, namely Kerberos. The combination of attestation of a client system's trustworthiness with user authentication and authorisation is a key issue. We show that this combination is efficient in a generic demonstration environment for TC-based applications. Part of this work is done within the EU FP7 project NanoDataCenters as a base for trust in distributed environments.

Section 2 provides some essential background on TC technology. Section 3 describes the demonstration environment we put together building on and combining available open source projects. Section 4 details the concepts for a TC-enabled ticket system architecture and describes the most important use cases, while Section 5 sketches the concrete integration into the Kerberos framework. We conclude with Section 6.

D. Gritzalis and J. Lopez (Eds.): SEC 2009, IFIP AICT 297, pp. 152–163, 2009.

2 Trusted Computing Essentials

The idea of building security into open, connected systems by using computing platforms enhanced by security-relevant functionality in protected places has a long history, rooted in the study by the Rand Corporation [13].

The TCG is the main industrial effort to standardise TC technology. Trust as defined by the TCG means that an entity always behaves in the expected manner for the intended purpose. The trust anchor, called Trusted Platform Module (TPM), offers various functions related to security. Each TPM is bound to a certain environment and together they form a trusted platform (TP) from which the TPM cannot be removed.

To prove trustworthiness of a TP to an external party, or verifier, processes called (remote) attestation and according protocols have been envisaged. They transport measurement values and data necessary to retrace the system state from them, so called measurement logs, to the verifier. The data is uniquely and verifiable bound to a particular platform, e.g. by a digital signature. Remote attestation can be supported by a PKI structure for instance to protect a platform owner's privacy by revealing the platform identity only to a trusted third party. The following technical details are taken from [17]. More can also be found in [14].

For the TPM to issue an assertion about the system state, two attestation protocols are available. As the uniqueness of every TPM leads to privacy concerns, they provide pseudonymity, resp., anonymity. Both protocols rest on Attestation Identity Keys (AIKs) which are placeholders for the EK. An AIK is a 1024 bit RSA key whose private portion is sealed inside the TPM. The simpler protocol Remote Attestation (RA) offers pseudonymity employing a trusted third party, the Privacy CA (PCA), which issues a credential stating that the respective AIK is generated by a sound TPM within a valid platform. The system state is measured by a reporting process with the TPM as central reporting authority receiving measurement values and calculating a unique representation of the state using hash values. For this, the TPM has several Platform Configuration Registers (PCR). Beginning with the system boot each component reports a measurement value, e.g., a hash value over the BIOS, to the TPM and stores it in a log file. During RA the communication partner acting as verifier receives this log file and the corresponding PCR value. The verifier can then decide if the device is in a configuration which is trustworthy from his perspective. Apart from RA, the TCG has defined Direct Anonymous Attestation. This involved protocol is based on a zero knowledge proof but due to certain constraints of the hardware it is not implemented in current TPMs.

AIKs are crucial for applications since they can not only be used, according to TCG standards, to attest the origin and authenticity of a trust measurement, but also to authenticate other keys and data generated by the TPM. Before an AIK can testify the authenticity of any data, a PCA has to issue a credential for it. This credential together with the AIK can therefore be used as an identity for this platform. Using the AIK as a signing key for arbitrary data is not directly possible but we have shown elsewhere how to circumvent this limitation [14,12].

3 Trusted Demonstration Environment

In order to develop and test various ideas and concepts of TC, we set up a Trusted Demonstration Environment. The goal was to design a system in which it is possible, without the need of a physical TPM, to access all desired TPM functions. In order to emulate a TPM in software, we used the TPM emulator from [5]. To simulate a complete system we decided to build upon Virtualisation. Therefore we established a connection between the emulated TPM and QEMU [3], thus enabling virtual machines to execute TPM applications and commands.

The **TPM emulator** enables to access and review the internal operations in the TPM, which made it a very powerful tool for analysis, testing and debugging. The emulator consists of three parts: an implementation of the TPM Device Driver Library (TDDL), a kernel module (tpmd_dev) and the TPM emulator daemon. As specified by the TCG, TDDL provides a convenient way to access the TPM from applications. By substituting this library, applications that use the TDDL are forced to use the TPM emulator instead of a hardware TPM. For those applications and libraries that access the TPM directly, the kernel module tpmd_dev simulates a hardware TPM by forwarding all messages directly to the TPM emulator daemon, which is the main component of the emulator. The tpmd listens on a Unix socket and waits for incoming commands. At current, most of the commands specified by the TCG are supported by the emulator. The installation of the TPM emulator is quite straightforward, compiling from source version 0.5.1. To prevent that QEMU gets disconnected from the emulator before the guest OS is up, as long as no TPM commands are issued during the boot process, we decided to change TPM_COMMAND_TIMEOUT in tpmd.c to a higher value (3000, default: 30).

To establish the connection between the tpmd and QEMU to gain a **virtualised client environment** we used a patch from [4]. We modified the patch to work with the current QEMU source version 0.9.1 [2]. The patch allows QEMU to connect to the Unix socket created by tpmd via command line option, and registers a new IO port inside QEMU and forwards all commands to the socket and thus to the TPM emulator.

We set up a virtual machine in QEMU using a standard debian distribution (etch). To communicate with the TPM the kernel (version 2.6.24-3) was recompiled, adding the IMA patch from IBM [6]. We configured the kernel to support TPMs In addition we enabled IMA.

IMA stands for Integrated Measurement Architecture and enables measurement and logging of every file that the kernel loads. Measurement is done in two steps: before execution, the SHA-1 hash of every file and library is measured and written to PCR 10 of the TPM using "extend" as in equation 1. Additionally the measured file and its SHA-1 hash gets logged in the /sys/kernel/security/ima/ascii_runtime_measurements file.

$$\mathbf{PCR10} = \mathbf{SHA\text{-}1}\,(\mathbf{PCR10}||\mathbf{SHA\text{-}1}\,(\text{file})) \tag{1}$$

As a main concept of authenticated boot, measurement through IMA allows for a later attestation to a remote system. It should be noted that every file gets measured only once, upon first execution. While the measurement list contains all files that executed since the system boot, it is not a list of the current running configuration of the system. Thus IMA cannot implement an actual run-time measurement. Using emulation of both, the TPM and the machine connecting to it, enabled us to set up on a single host multiple clients each of them with its own TPM. Integrated in this framework is QEMU's capability to connect the clients to a virtual network and thus enabling a complete client/server infrastructure.

In order to access the TPM inside the client, we developed several applications in Java, based on the framework TPM/J from [7] implementing a **client-server infrastructure**. Around various little tools for creating and storing keys for later use, we implemented the complete process of Remote Attestation. In particular our environment can create an arbitrary number of AIKs and, by connecting to our PCA, realised as a single module written in Java, we can certify our AIKs.

The process of AIK certification is shown in figure 1. First, the client creates a new AIK using the TPM_MakeIdentity command. This creates the key and the TPM_IDENTITY_CONTENTS structure which contains the public AIK. This structure is then signed with the AIK and sent to the PCA together with the public EK and its certificate.

Our realisation of the PCA creates a random nonce and encrypts both, the public AIK and the nonce with the EK. The client then has to load the AIK into the TPM, decrypt the nonce using TPM_ActivateIdentity, and finally checks if the decrypted public AIK corresponds to the loaded AIK. Then the decrypted random nonce is sent back to the PCA. This is used as a handshake operation — extending the TCG's specification of this protocol — to ensure that the PCA communicates with the TPM that generated the AIK. The corresponding security weakness in the TCG's specified protocol was noted in [8], and we followed the solution proposed there. The handshake operation generates a reliable link between the EK (stored in the TPM) and the AIK. Omitting the handshake poses several risks. If an attacker can get hold of a public EK and the credentials, he can request PCA credentials for arbitrary RSA keys. These keys are not necessarily bound to the TPM and can be created without a TPM. Thus, one can imagine DoS attacks on the PCA. If a policy enables the PCA to issue only a limited amount of certificates for a certain TPM, or users are charged for the issuance of the certificates, this leads to further attacks. Note that the attacker will not be able to decrypt the credentials for the given AIK, as they can only be decrypted by the private EK which will not leave the TPM.

The PCA then verifies the correctness of the EK certificate and validates the AIK signature on the TPM_IDENTITY_CONTENTS. After generating the AIK certificate cert(AIK,PCA_{cert}), the PCA encrypts it using a symmetric key K. To ensure that only the requesting client can access the certificate, the key K, together with the hash of the public AIK is encrypted with the public EK in the TPM_EK_BLOB structure. The encrypted certificate and the encrypted TPM_EK_BLOB are then sent back to the client. The client's TPM can

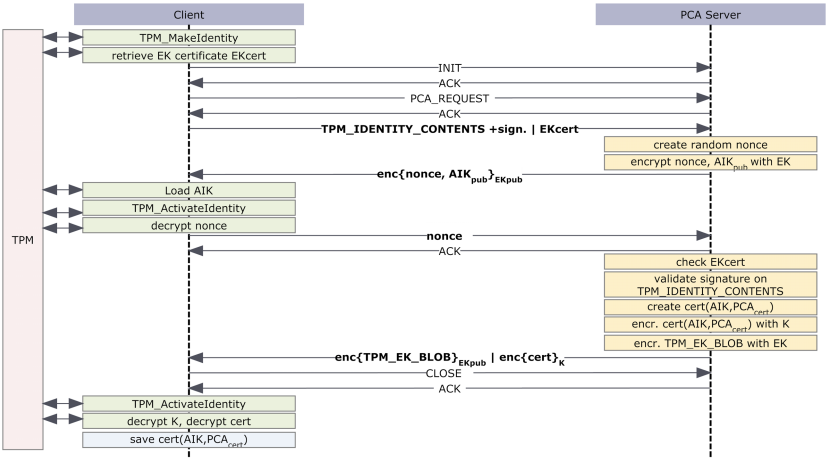

Fig. 1. Flow of the AIK certification protocol

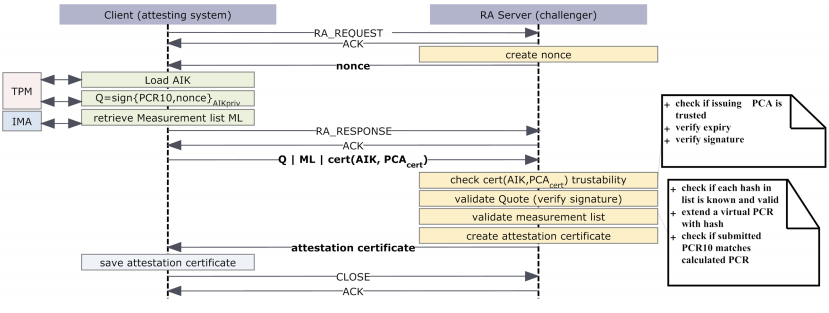

Fig. 2. Flow of the RA protocol

decrypt the symmetric key K using the TPM_ActivateIdentity command and can thus decrypt the cert(AIK,PCA_{cert}) which is stored for further use. With these certified AIKs it is possible to connect to our Remote Attestation (RA) server. The client sends the IMA measurement list together with the current value of PCR 10 signed with the AIK to the server. The RA server then checks if the AIK certificate is valid and issued by a trusted PCA. The RA server validates the measurement list in the following way: first, every entry in the measurement list is checked if it is contained in a database of known hash-values for programs defined as trusted. If the client executes a program that is considered untrusted, the hash will not be found in the database or in case of a virus/malware modifying a trusted program, the hash will be different. In this case, the client is considered untrusted and the attestation fails. If the hash value matches the known (trusted) value, a virtual PCR is extended as described in equation 1. After examining all entries in the measurement list, the RA server checks if the virtual PCR matches the submitted value from client's TPM. This

procedure ensures that the submitted measurement list has not been tampered with. The client receives a certificate from the RA server with a timestamp. This certificate can then be used in a following connection to a service provider. The RA process is shown in figure 2. In the current demonstration environment we are not yet able to access the TPM emulator through the QEMU BIOS, preventing a measurement of the boot process. A patched version of the boot loader GRUB [10] exists and is installed in the client. As soon as a working patch for the QEMU BIOS exists, we will be able to establish a chain of trust starting at boot time of the virtual machine.

4 Usage Concepts

Our goal is to integrate the TC concepts in an **IDM** Environment. IDM is concerned with the management of user credentials and the means by which users use them to access different (online) services. A real life person's identity is formed of all attributes that belong to the individual such as name, address, hobbies, banking accounts. The identity can be split into partial identities which consist only of a subset of information, e.g. a drivers license contains name, date of birth, a picture and the type of vehicle a person is entitled to drive, whereas a credit card account contains name, account number and a list of last purchases. Internet usage increases the number of identities, represented by, e.g., different accounts for mail, online-shops, auctions and so on. Every identity contains different types of information about the subject.

One goal is the establishment of trust domains where participants can trust each other. In traditional scenarios trust is based on the fact that the participants know each other, e.g., because they belong to the same company. As the customer-business relation shifts from physical to electronic means it is necessary to develop and establish new ways of trust relationships between enterprises and their customers.

In general, IDM covers several aspects: (i) **Trust** is linked to a set of identity credentials, allowing an individual to be part of a trust domain. (ii) **Anonymity** and **pseudonymity** play an important role in IDM. The level of identity needed for a relationship in a trusted domain has to be considered. (iii) **Authentication** is needed to prove that a claimed identity really belongs to the agent. Examples are passwords, biometric devices or smartcards. (iv) **Authorisation** describes the process of either granting or denying access to a certain service or resource. (v) **Integrity** ensures that a message cannot be changed once it has been sent. (vi) **Non-repudiation** means the evidence for the existence of a certain message can be provided.

Several use cases of IDM can be imagined and are currently being widely promoted. In most web based services, the user needs to sign in for an account in order to access the service. This implies the need for an account management on behalf of the service provider. In an IDM scenario, the provider only has to care about authorisation and has to establish a trust relationship to an identity provider. The client only has to retrieve a ticket incorporating his identity from

the identity provider. He then uses the obtained ticket and presents it to the provider without the need of an additional registration. This lowers entry barriers for users that want to access the service.

In existing IDM solutions one security concern is the phishing of the login information needed to sign up with the identity provider. Once this information gets stolen, all login information to services will be exposed. With our solution, the tickets will be build on the client's hardware TPM. As the tickets get bound to the user's device, an interception of login information is rendered useless. In addition, the spread of different, somehow loosely linked partial information leads to certain risks. As users tend to use same login information when registering with different services, gathering and linking informations contained in different identities can lead to a complete profile of an individual. This poses huge privacy concerns potentially enables further attacks such as identity theft where an attacker obtains enough key pieces of personal information to impersonate someone else [15].

We take an approach of trust, such that by transmitting authentication data, an agent enters the domain of trust of a principal. Within authentication, the trustworthiness of the agent can be attested, thus making a statement about the agent's identity and its state. The token used for authentication and attestation is called credential. In existing IDM solutions such as Kerberos, these credentials are embodied in (software) tickets. Using TC concepts we will build trust credentials that rely on the agent's TPM. By the means of mutual agreements trust can be established across different domains. In such an architecture every domain consists of (at least one) identity provider and multiple service providers. An acquired ticket can only be used inside the particular domain. In order to enable ticket usage across multiple domains, either cross certification (one CA signing the public key of the other CA) or a spanning CA can be used. This leads to a high technical overhead if many domains are involved. With the approach of using TC architecture we are be able to establish trust between different domains.

We build our trusted ticket system upon the identities embodied in the AIKs certified by a PCA. The tickets are generated locally on the user's device and the process of ticket acquisition and redemption is bound to the user's hardware TPM chip. The tickets are only be usable from the user's device, and thus prevent attacks that rely on copying the ticket (e.g. phishing). We establish an access control scheme allowing the user to access multiple services by maintaining non-repudiation (by the chain of trust), accountability (by the PCA) and pseudonymity (by the separation of duties), see [12,11].

As described in [14] we create the trusted tickets using an indirection. We use the notation cert(*entity, certificate*) for the **credential of the certified entity**. It is the union of the entity signed with the certificate's private key and the public key $certificate_{pub}$. The credential is verified by checking the signature.

First, the AIK representing a certain identity is certified by a PCA, yielding the AIK certificate cert(AIK,$\mathrm{PCA}_{\mathrm{cert}}$). As AIKs cannot be used to sign arbitrary data, we create a new RSA key pair in the TPM using

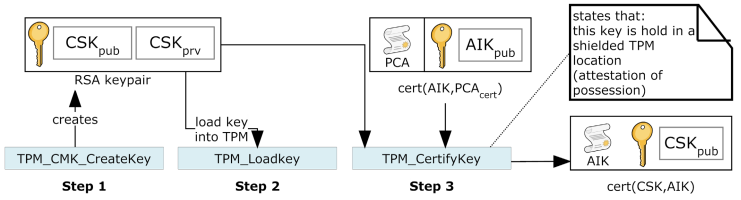

Fig. 3. Creation of the Certified Signing Key (CSK)

the TPM_CMK_CreateKey command. The resulting key pair is loaded into the TPM via the TPM_LoadKey command and then certified by the TPM_CertifyKey command. Using this indirection we are able to create for each AIK a certified key. This so called certified signing key (CSK) can then be used to sign arbitrary data. In order to access a service, a service request R is signed with the CSK and cert(R,CSK) is obtained. The service request R, together with cert(R,CSK), cert(CSK,AIK) and cert(AIK,PCA_{cert}) build the credential chain which is transferred to the Service Provider. This data embodies the ticket. By verifying the chain an authorisation decision can be made and access to the service can be granted. The process of CSK generation is detailed in [14] and sketched in Figure 3.

5 Trusted Tickets in Kerberos

This section exhibits how the CSKs can be generated in a Kerberos IDM environment and how the obtained tickets can be integrated into Kerberos.

The **Kerberos** authentication consists of three phases. First, the client requests a ticket granting ticket (TGT) from the Authentication Server (AS). The AS sends the TGT together with an encrypted session key back to the client. Only the client can decrypt the session key with his password. In the next phase, the client uses the TGT to request a service ticket (ST) from the Ticket Granting Server (TGS). The response contains the ST together with a second session key encrypted with the first session key. Finally the client uses the ST to access an application server providing the service. The whole process is shown in Figure 4.

The Kerberos protocol provides the *authorization-data* field which can be used to embed authorisation information into a Kerberos ticket. Referring to [16, p. 57], the usage of the *authorization-data* field is optional. We take advantage of this field and use it to include the information needed to build the trusted ticket. The type of this field is set to AD-IF-RELEVANT, so that servers, that don't understand the embedded information will be able to ignore the included data.

In our design, the AS takes the PCA functionality of signing and thus certifying AIKs. Therefore the client initiates the session by sending the request including authorisation data, the public part of the AIK, the EKC and the cert(AIK,EK) to the AS. By including the cert(AIK,EK) the client states that the AIK has been generated by the TPM hardware the EKC belongs to. The

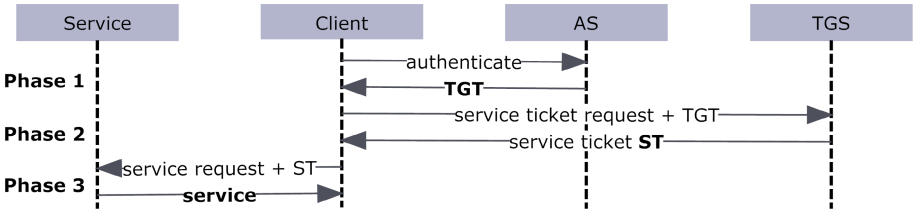

Fig. 4. Flow of the Kerberos protocol

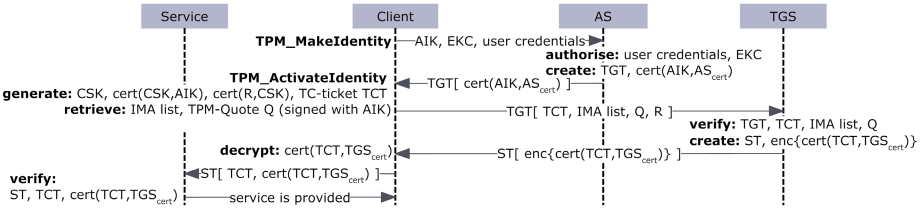

Fig. 5. Protocol of a TC enabled Kerberos Infrastructure

AS then checks the authorisation data provided by the client. By verifying the EKC, the AS verifies that the TPM is implemented by a trustworthy (at least known) manufacturer. Further, by checking the cert(AIK,EK) it can be verified that the AIK belongs to that single TPM. Upon success, the AS issues a certificate cert(AIK,AS_{cert}), binding the AIK to the respective identity. The certificate is encapsulated in the *authorization data* field of the TGT. When the client receives the TGT, he can extract the cert(AIK,AS_{cert}) from the ticket. By using the AIK certificate, the client is able to create a CSK belonging to this AIK by using the process described in Section 4.

When the client wants to access a certain service, he requests an ST. The client signs the request message R with the CSK and can thus build the credential chain cert(R,CSK), cert(CSK,AIK), cert(AIK,AS_{cert}). This credential chain is called TC-Ticket (TCT). Verifying the TCT means to check if the chain resolves to a trustworthy issuer (which is in our case the AS). In order to make a statement about the platform configuration, the client retrieves the IMA measurement list and a TPM quote of the PCR-10, signed with the AIK. The client then sends the service request R, the TCT, the IMA list and the quote enclosed in the TGT to the Ticket Granting Server.

Upon receipt, the TGS first verifies the TGT. If it is issued by a trustworthy AS, the TGS verifies the TCT. Therefore policies for authorisation and agreements on trust have to be established between AS and TGS in advance. Then the TGS will go on with the process of remote attestation as detailed in 3. The TGS can then create the certificate cert(TCT,TGS_{cert}), stating that the TCT comes from a valid client in a trustworthy state. Note that the certificate has to be equipped with a timestamp, making it valid only for a short period of time.

Otherwise, the system state could change in a significant way. In order to render eavesdropping useless, the TGS will encrypt the certificate with the public part of the CSK via the Tspi_Data_Bind Operation. Only the client that is in possession of the CSK (and the TPM it was created with) can then decrypt the certificate via the Tspi_Data_Unbind method.

In the case of a successful verification, the TGS issues the ST in which the encrypted cert(TCT,TGS_{cert}) is included. The TGS can resolve the credential chain to a single AIK that is certified by the AS. This allows for anonymity of the person using the TGS, as the identity is only known by and revealed to the AS. Only AS is able to de-anonymise users as the platform credentials provide the necessary information. In case of misbehaving users, AS could reveal the personal identity to the TGS.

After decrypting the received certificate from the TGS, the client is able to use the obtained ST together with its own created TCT and the newly received cert(TCT,TGS_{cert}) to access a service. The service provider has to verify if (i) the ST is issued by a trustworthy TGS, (ii) the TCT resolves to a trustworthy issuer and (iii) the cert(TCT,TGS_{cert}) belongs to the TCT and comes from a trustworthy TGS. The service is then provided to the client.

Note that the cert(TCT,TGS_{cert}) contains the signed service request as well as a timestamp. This also allows for protection against double spending of the ticket. The service provider therefore has to keep track of redeemed tickets. As they do not contain any data about the personal identity of the client, only a limited amount of information can be gathered. The client is able to generate a new AIK for every identity he wants to use.

6 Conclusions

We have shown how to integrate the concept of trusted tickets in the Kerberos protocol as an existing IDM solution. The complete process, from authentication over ticket generation to ticket redemption at the service provider is shown. A proof of concept integration into the Kerberos protocol is given. By integrating PCA functionality into the Kerberos AS and remote attestation done by the TGS, we are able to issue tickets bound to the client platform. By the separation between AS and TGS we showed how pseudonymity for the client can be achieved. In some scenarios it might be required to charge the client for accessing a certain service. As mentioned in [14], an additional charging for the use of certain services could easily be implemented by extending the protocol on the part of the AS and TGS entities. Upon issuing a ST for a certain service, the TGS then requests a charging for this ticket at the AS. The AS in turn can then initiate the charging for this ticket at a third party charging service. Upon charging, the charging provider must be able to identify the user by credit card account or similar means. Data protection laws can prevent the charging provider from disclosing identity related information. While this offers a protection for several identity based attacks, such as profiling, identity theft, phishing attacks, TC-based tickets additionally enable a binding of the tickets

to the client's hardware. This is an important key in providing security against eavesdropping. The tickets are completely built on the TPM's basic functions. As in other TC applications like Digital Rights Management (DRM), the usage of a certain service is bound to the TPM. If the client's TPM fails, the tickets are no longer be valid. Thus there is no (financial) loss on the side of the service provider. In contrary to DRM where protected content is rendered unprotected and can cause monetary loss to the owner.

Our concept allows to implement multiple service access using one identity (AIK) to retrieve multiple STs. Every instance in the protocol is able to verify the chain of trust upwards to a trustworthy issuer. This concept maintains non-repudiation throughout the whole protocol. In order to enable usage of the tickets in different identity domains there have to be agreements on trust between the service providers and the respective AS and TGS servers. In this usage scenario, protection against multiple spending has to be implemented. Further usage of TC concepts could include an adaption of the attestation protocol which allows service providers to report their status to the clients. Such usage could prevent malicious service providers from stealing customer data, further increasing the security of online transactions.

References

1. Kerberos: The Network Authentication Protocol, `http://web.mit.edu/Kerberos/`
2. Nanodatacenters / Results / Security Experimentation Environment, `http://nanodatacenters.eu/`
3. QEMU, `http://bellard.org/qemu/`
4. [Qemu-devel] [PATCH] Add TPM support, `http://www.mail-archive.com/qemu-devel@nongnu.org/msg13408.html`
5. Software-based TPM Emulator, `http://tpm-emulator.berlios.de/`
6. SourceForge.net: Integrity Measurement Architecture (IMA), `http://sourceforge.net/projects/linux-ima`
7. TPM/J Java-based API for the Trusted Platform Module (TPM), `http://projects.csail.mit.edu/tc/tpmj/`
8. Gürgens, S., Rudolph, C.: AIK Certification. Technical report, Fraunhofer SIT / BSI. 13 (April 2006) (unpublished)
9. Trusted Computing Group: Home, `https://www.trustedcomputinggroup.org/home`
10. Trusted GRUB, `http://trousers.sourceforge.net/grub.html`
11. Fichtinger, B.: Trusted infrastructures for identities. Master's thesis, Fachhochschule Hagenberg, Austria (May 2007)
12. Fichtinger, B., Herrmann, E., Kuntze, N., Schmidt, A.U.: Trusted infrastructures for identities. In: Grimm, R., Hass, B. (eds.) Proc. 5th Internat. Workshop Virtual Goods, Koblenz, Hauppauge, New York, October 11-13, 2007. Nova Publishers (2008)
13. Gasser, M., Goldstein, A., Kaufman, C., Lampson, B.: The Digital Distributed System Security Architecture. In: Proc. 12th NIST-NCSC National Computer Security Conference, pp. 305–319 (1989)

14. Kuntze, N., Mähler, D., Schmidt, A.U.: Employing Trusted Computing for the forward pricing of pseudonyms in reputation systems. In: Proc. Axmedis 2006, Atti del Convegno, pp. 145–149. Firenze University Press (2006)
15. Liberty Alliance. Whitepaper: Identity Theft Primer (December 2005)
16. Neuman, C., Yu, T., Hartman, S., Raeburn, K.: The Kerberos Network Authentication Service (V5). RFC 4120, updated by RFCs 4537, 5021
17. TCG. TCG TPM Specification Version 1.2 Revision 103. Technical report, tcg (2007), Trusted Computing Group (retrieved February 29, 2008), `https://www.trustedcomputinggroup.org/groups/tpm/`