Hardware Security for Device Authentication in the Smart Grid

Andrew J. Paverd and Andrew P. Martin

Department of Computer Science, University of Oxford, UK
{andrew.paverd,andrew.martin}@cs.ox.ac.uk

Abstract. Secure communication between devices is a key aspect of smart grid security. In the future *smart home* environment, various smart devices, appliances and energy management systems will communicate with each other via the home network. In order to achieve mutual authentication, each device will have a private cryptographic key which must be protected against theft or misuse. Current mechanisms for protecting such keys exist but generally require interaction with the user. This makes them unsuitable for the smart grid context due to the high degree of automation involved in the smart grid. To address this challenge, we have designed, implemented and tested a system that provides hardware security for device private keys using Trusted Computing technologies. Using DRTM late-launch functionality, our system ensures that the private key is only available within a protected trusted environment on a specific device. Preliminary implementation and testing has demonstrated that our system can operate successfully in unattended environments such as the smart grid.

1 Introduction

In the various visions of the future smart grid, it is universally acknowledged that residential customers will be a key aspect of the overall system, especially since the residential sector accounts for more than a quarter of final energy consumption in Europe [1]. The widely-cited NIST Framework and Roadmap for Smart Grid Interoperability Standards [2] defines the residential *Customer* domain as one of the seven primary domains in the smart grid conceptual reference model as shown in Fig. 1. The future *smart home* environment will generally include a smart meter, an Energy Services Interface (ESI), and an Energy Management System (EMS). Naturally, it will also include various smart devices, appliances and other customer equipment. The communication between these systems will be important for facilitating automated energy management within the home.

Due to the nature of the communicated information, the provision of *secure* communication functionality is a primary requirement of the smart grid. This is particularly important in the home environment where communication often takes place over wireless networks or the public Internet. Secure communication usually requires mutual authentication of the communicating entities. The widely-used Transport Layer Security protocol (TLS) provides this using

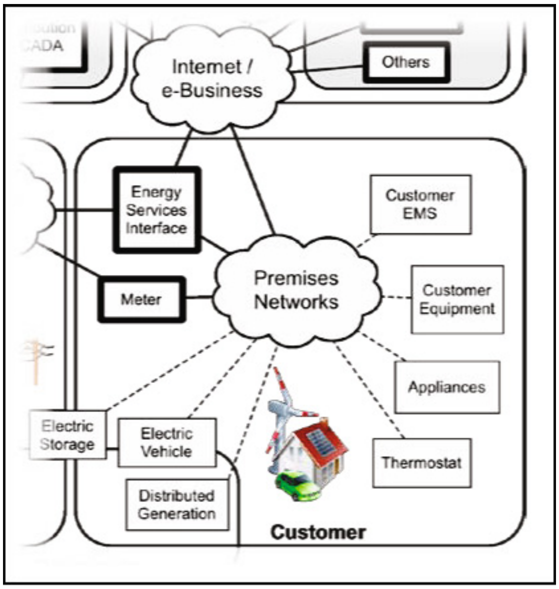


Fig. 1. Customer Domain from the NIST Conceptual Reference Model [2]

asymmetric cryptography and a Public Key Infrastructure (PKI). This approach could facilitate secure device authentication in the smart grid by providing each device with a unique private key [3,4]. However, on general purpose computational systems such as PCs, the security of this private key could be threatened by malicious software (malware) or unauthorized access. Therefore, a suitable mechanism is required for protecting these device private keys.

One of the key features of the smart grid is that it will ultimately facilitate automated management of energy consuming devices and systems. In the home environment, smart devices will report their current state and power consumption information to the EMS which will deactivate or reactivate specific devices based on varying energy prices or demand response signals. This inherently means that smart grid systems must be designed to function without requiring user interaction since a user may not always be present. This poses a particular challenge for the protection of private cryptographic keys because existing mainstream approaches for protecting keys have not been designed to support unattended operation.

This paper provides two main contributions: Firstly, we show that the problem of protecting device private keys in the smart grid context is not adequately solved using existing mainstream approaches because of the automated nature of the smart grid. Secondly, we demonstrate that this problem can be solved using Trusted Computing and we present an architecture and proof of concept implementation for achieving this. This paper is organized as follows: Section 2 discusses existing approaches for protecting private keys and provides background information on Trusted Computing. Section 3 specifies the security and privacy requirements as well as the functional requirements for a solution to this problem. We present the architectural design of our system in Section 4 and evaluate our proof of concept implementation in Section 5. Section 6 summarizes related work and Section 7 presents our conclusions.

2 Background

2.1 Existing Approaches

There are currently four main approaches for protecting device private keys:

No Specific Security. The most basic option is to simply store the key unencrypted so that it can be used at any time. The major vulnerability of this type of approach is that the key is not protected from any malware or unauthorized access to the device. There are cases in which this vulnerability is not applicable such as some embedded systems which are incapable of running malware and do not support remote access. However, for most devices in the smart home, these threats pose a serious risk to unprotected private keys.

Software Obfuscation. Software obfuscation techniques allow for a cryptographic key to be hidden within a specific software binary. Although the key is not encrypted, the obfuscation increases the work required to find and obtain the key since the attacker must usually reverse engineer the software binary. In this approach, the same obfuscated key will be shared by all systems running the specific software binary. This key is usually used as a master key to encrypt a key store containing the device's unique private key. However it is acknowledged that software obfuscation provides only limited security [5]. If the obfuscation is circumvented, the private keys of all devices would become vulnerable. Assuming this breach is detected, it would necessitate the replacement of the obfuscated key across all devices.

Encryption with a User Secret. The most common approach is to encrypt private keys with a user-supplied secret such as a password or PIN. When required, the user enters this secret and the key is temporarily decrypted into volatile memory. However, whilst in this decrypted state, the key can still be compromised by an attacker exploiting vulnerabilities in the OS or the application using the key. Instead of decrypting the key every time it is required, some key manager systems allow the user to decrypt the key once at startup and then keep it in volatile memory for future use. However, this still requires an initial user interaction and increases the window of vulnerability for application or OS exploitation. The user-supplied secret is also vulnerable to social engineering attacks which could ultimately compromise the security of the key. In the smart grid context, private keys would be used to authenticate devices rather than users so ideally the protection of these keys should not involve the user.

Hardware Security for Private Keys. The third main approach involves the use of specialized security hardware such as a smart card or Hardware Security Module (HSM) to protect private keys. There are a variety of solutions of this type but most follow the principle that the private key can only be accessed by the HSM. If implemented correctly, this type of approach virtually eliminates the possibility of an attacker obtaining the private key. When required, the HSM uses this key for cryptographic operations and outputs the result. However, systems of this type require additional security mechanisms to ensure that requests do not originate from an attacker. For example, smart cards generally require the user to enter a PIN to confirm an operation. A more sophisticated approach for protecting information using hardware security has been provided by the Trusted Computing Group (TCG) as explained in the next subsection.

2.2 Trusted Computing

The Trusted Computing Group (TCG) has defined various technologies and processes that are collectively known as Trusted Computing (TC). Many of these are based on the Trusted Platform Module (TPM), a standards-based cryptographic co-processor [6]. At present, the TPM is integrated into the majority of business-focussed PCs as a discrete hardware module. It is envisioned that a TPM could also be included in future smart devices and home appliances.

The TPM allows a system to create a record of its software state using a set of secure Platform Configuration Registers (PCRs) on the TPM. Each PCR stores a SHA-1 hash value which can not be directly written but can be *extended* by the TPM by concatenating the existing value with the new input and storing the hash of the result. In TCG terminology, software can be *measured* by computing the hash of the complete software element and extending it into a PCR. For a *measured boot*, every piece of software is measured by the preceding software before being executed. This forms a *chain of trust* starting from the platform's initial startup.

The TPM provides secure storage using a unique asymmetric Storage Root Key (SRK) of which the private part never leaves the TPM. The SRK can encrypt any number of symmetric keys which can in turn encrypt data. In this way it is possible to *bind* data to a specific platform based on the SRK. It is also possible to *seal* data to a specific software state in which case the TPM will not decrypt the data unless the PCRs match some predefined value.

TC provides another potential solution to the original problem of protecting device private keys. After a measured boot, the TPM can seal the device private key to a specific secure software state. However, the current TCG approach to sealed storage has various shortcomings [7]. For example, the installation of a patch or update results in a different set of PCR values and so would prevent the TPM from unsealing data. To address this challenge, an alternative approach has been implemented by major microprocessor vendors in the form of a Dynamic Root of Trust for Measurement (DRTM).

2.3 Dynamic Root of Trust for Measurement

The concept of a *late-launch* allows the system to startup in an unmeasured state and then transition into a measured state [8]. This is said to provide a Dynamic Root of Trust for Measurement (DRTM) which can be used as a trust anchor for subsequent operations. The *Flicker* research project [9] uses this capability to provide an isolated and protected execution environment on modern x86 platforms. When invoked, Flicker suspends the host OS and initiates a late-launch using a special CPU instruction. This resets a subset of the PCRs (known as the Dynamic PCRs), disables Direct Memory Access (DMA) and partially resets the CPU. Flicker then executes a Piece of Application Logic (PAL) which is a small section of code provided by an application. Due to the partial CPU reset, this PAL is unaffected by any software which was executed on the system before the late-launch. The PAL can therefore be said to execute in a Trusted Execution Environment (TEE). Flicker allows any application to perform certain operations within this TEE by encapsulating the operations as a PAL. By using late-launch and the dynamic PCRs, Flicker also makes it possible to seal and unseal data within a specific PAL irrespective of the overall state of the host system. However, the functionality of a PAL is inherently limited because the PAL should not use any libraries or hardware drivers from the host OS since these may have been compromised before the late-launch. Once the PAL has completed execution, the memory used by the PAL is cleared and the host OS is resumed from its suspended state. The PCRs are also extended with a well-known value to indicate that the TEE is no longer active.

3 Security, Privacy and Functional Requirements

As introduced in Section 1, secure communication between devices is an important aspect of smart grid security. In the smart home environment, each smart device will have a unique private key that is used for mutual authentication [3,4]. In this context, the primary goal of an attacker would be to intercept or modify network communication or to impersonate a legitimate device. This could be part of a cyber-physical attack which, if successful, could have serious consequences in the smart grid environment. Assuming that it is infeasible to break the cryptographic protocols themselves, the attacker would try to obtain a device's private key in order to carry out this attack. On general-purpose systems such as PCs, it is already the case that the security of this key could be threatened by malware or unauthorized access. Additionally, since this key is a form of strong device identification, a secondary attacker objective might be to misuse the device private key in order to identify the device to an unauthorized third party. For example, signatures from this key could be used for tracking the device or its user and thus constitutes a threat to user privacy. Therefore, a mechanism is required for protecting the device private key. Furthermore, in order to be used in the smart grid environment, this mechanism must be able to operate without requiring user interaction. Based on this context, the mechanism for protecting device private keys must satisfy the following requirements.

Security & Privacy Requirements

SR-1: The device private key can only be used on a specific physical device or system.

SR-2: The device private key can only be used by a specific set of software applications.

SR-3: The device private key can only be used for authenticating the device to authorized network end-points.

Functional Requirements

FR-1: The mechanism must facilitate the use of the private key in the standard TLS handshake protocol.

FR-2: After the key initialization phase, the mechanism must not require further interaction with the user.

Based on the above requirements, it can be seen that the existing mainstream approaches for protecting device private keys are not well suited for use in a smart grid environment. Whilst it is possible to satisfy the functional requirements by simply storing the key in unencrypted form, this leaves the key vulnerable to being compromised by malware or unauthorized access to the system and therefore does not fulfil any of the security requirements. Using obfuscation techniques to protect the key would satisfy the functional requirements and SR-2 provided the obfuscation is not compromised. In this approach, the applications must be trusted to not violate SR-1 and SR-3. Encrypting the key with a user-supplied secret satisfies SR-1 and potentially SR-2 and SR-3 since the user should only provide the secret to trusted applications for purposes of authenticating to authorized network end-points. However, due to the required user interaction, this type of approach does not satisfy FR-2 and therefore cannot be widely used in an automated smart grid environment. The use of hardware security could theoretically satisfy all the security, privacy and functional requirements. However, previous systems of this type have not fully achieved this objective in a practical manner. For example, a key can be bound to a specific platform (SR-1) but this does not restrict the use of the key (SR-2 & SR-3). Sealing the key to a specific platform state fulfils all the requirements but is arguably not yet practical on unmanaged systems in the home environment due to the various shortcomings of sealed storage [7].

4 System Architecture

Based on the defined requirements, we have designed a system to protect device private keys in the smart grid context. The fundamental concept is that the device private key can only be used from within a trusted execution environment (TEE). For our proof of concept implementation, this TEE is provided by the open-source *Flicker* project [9] described in Section 2.3. Our system has two main phases of operation: the key initialization phase and the TLS handshake.

4.1 Key Initialization

The objective of the key initialization phase is to generate the device private key within the trusted environment and to establish restrictions on the use of this key. This process must take place before the device connects to the network.

In order to restrict the use of the device private key to specific applications, we use the Linux Integrity Measurement Architecture (IMA) subsystem [10]. With IMA activated, the Linux kernel computes a measurement hash of each software application before it is run and stores this measurement in a secure log. The use of IMA for identifying specific applications has been described by Bugiel and Ekberg [11]. In a similar manner, our system uses IMA to specify a trusted application that may use the device private key. The hash measurement of this application is obtained from the IMA subsystem and provided as input to the trusted environment. Additionally the user can input a Certificate Authority (CA) certificate that will be used to restrict the use of the key.

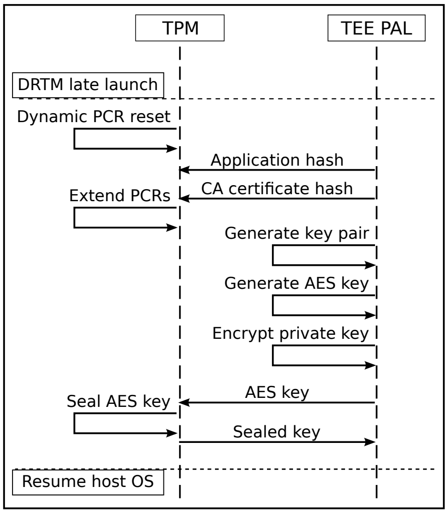
As shown in Fig. 2a, when the system switches to the trusted environment, the dynamic PCRs are reset and the measurement of the PAL is implicitly extended into one of these PCRs. The PAL then extends the PCRs with the hash measurement of the specified application and the hash of the CA certificate. The PAL generates the asymmetric key pair of which the private part is the device private key. The PAL also generates a symmetric key and encrypts the device private key using AES. The AES key is then sealed by the TPM using the SRK and the current PCR values. This intermediate AES step is necessary because the device key is too large to be sealed directly by the TPM.

Once the PAL has completed its execution, all secrets are erased from memory and the PCRs are extended with well-known values. The PAL outputs both the AES key and the device private key as encrypted data structures which can be safely stored in non-volatile memory. The generated public key is output to the host environment to be included in the device certificate. The end result of the key initialization phase is that a device private key has been generated and encrypted within the trusted environment. This private key can only be decrypted or used by the same physical TPM and only by the same trusted environment since the PCR values must match their predefined state.

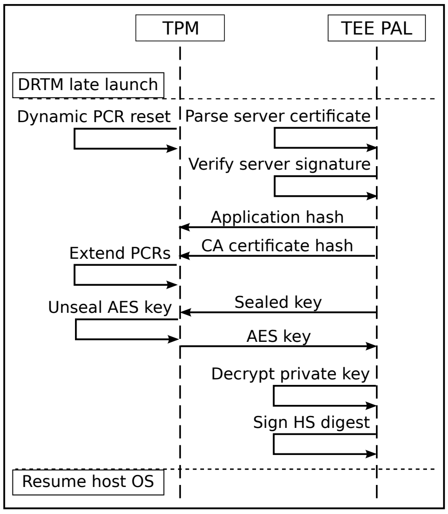
4.2 TLS Handshake Protocol

In a mutually authenticated TLS handshake, the client proves its identity by using its private key to sign a digest of the preceding handshake messages and sends this to the server as the *'client verify'* message. Fig. 2b shows how our system is integrated into this handshake protocol. We present our system from the perspective of the client device in the TLS handshake but the same architecture could be used on the server.

Before the trusted environment is launched, the preceding messages from the TLS handshake are provided as input for the PAL. This input causes the Flicker system to query the IMA subsystem for the measurement hash of the application that supplied the input. In contrast to the key initialization phase, this



(a) Key Initialization Process



(b) Enhanced TLS Handshake Protocol

Fig. 2. Sequences of interaction between the TEE and the TPM

application hash is automatically input to the PAL and so cannot be modified by external applications.

When the system initiates the late-launch procedure, the same series of measurement steps is performed as in the key initialization phase. The dynamic PCRs are reset, and one of these PCRs is extended with the measurement hash of the PAL, the application hash and the hash of the supplied CA certificate. The PAL does not need to check the application hash or CA certificate because if these differ from the key initialization phase, the PCR values will be different and the TPM will not unseal the required key.

The PAL will check that the end-point to which the connection is being established has provided a certificate signed by this CA. The server certificate is sent to the client before the late-launch and so it can be parsed from the list of previous handshake messages within the trusted environment. Additionally, the server's certificate is used to verify the server's signature in the handshake messages. If both of these verification steps are successful, the PAL can be sure that the digest it will sign is part of a TLS handshake with a permitted network end-point.

If the PCR values are correct, the TPM can unseal the AES key from the sealed key data structure and use it to decrypt the device private key. The TLS handshake digest is then signed using the private key. Once the trusted environment returns control to the host OS, the signature from the Flicker output is sent to the server to complete the handshake according to the standard TLS protocol.

5 Proof of Concept Implementation and Evaluation

We have implemented a proof of concept version of this system by modifying the PolarSSL TLS library[1] and the Flicker system [9]. Flicker was modified to interface with the IMA subsystem to obtain the measurement hash value of the application that provides input to the TEE. PolarSSL was modified to initiate the late-launch of the TEE whenever the private key is required. We used the OpenVPN-NL application[2] to demonstrate that our system can successfully establish a secure communication channel between devices without requiring user interaction.

5.1 Performance Evaluation

We obtained performance benchmarks for our system using a Dell Optiplex 990 desktop system with an Intel Core i5-2400 CPU running at 3.10 GHz and a version 1.2 TPM. Table 1 shows the average times and standard deviations of the various operations in the TLS handshake phase of the system (averaged over 3000 measurements). By far the longest operation is the unsealing of the AES key. This is the only operation for which the TPM itself is required to perform an asymmetric cryptographic operation. All other asymmetric cryptography is performed by the main application processor within the trusted environment. Since the TPM is not usually implemented as a high-performance device it has been noted that this type of TPM operation usually incurs a high performance cost [9,11]. Although it is important to minimize the time required for system execution, in the context of an unattended system, the only requirement is that the time taken should be within acceptable limits for completing the TLS handshake. This is achievable as demonstrated by the use of this proof of concept system with the OpenVPN-NL application.

Table 1. Benchmarks of various operations

Operation	**Time (ms)**	**[Std Dev]**
Extend PCR	8.96	[0.02]
Verify server certificate	0.30	[0.00]
Verify server signature	0.23	[0.00]
Unseal AES key	956.57	[0.30]
Decrypt RSA key	0.02	[0.00]
Sign TLS digest	2.09	[0.00]
Total PAL time	**982.91**	**[3.48]**

[1] `http://polarssl.org/`

[2] `https://openvpn.fox-it.com/`

5.2 Security Evaluation

A potential vulnerability of this system is that the Integrity Measurement Architecture (IMA) subsystem is not part of the trusted execution environment. In order to identify the application requesting use of the key, this subsystem depends on various components of the host OS including the file-system and process management subsystem. This dependency is necessary to bridge the semantic gap between the trusted environment and the host OS. It is possible that the IMA subsystem could be compromised by sophisticated malware or unauthorized access at the highest level of privilege. However, even in this worst-case scenario, the attacker would be able to misuse key but would not be able to obtain the key since it is protected by the trusted environment. Although this type of attack would break SR-2, the trusted execution environment and the TPM would still enforce SR-1 and SR-3.

The security properties provided by the trusted execution environment and the TPM itself could be compromised by an attacker who has direct physical access to the device. However, this would require sophisticated hardware tools and techniques such as eavesdropping on the hardware communication bus between the CPU and the TPM. It is assumed that for an attacker to have this level of physical access, the device must be in the attacker's possession and so its private key should anyway be revoked by the home network.

6 Related Work

In the *AutHoNe* project, Kinkelin et al. [12] have described how a non-migratable TPM key could be used as the private key of the *Home CA*. Since this key is not sealed to a particular platform state, the user is required to enter a PIN before the key can be used. They also recognize the possibility that malware could misuse this key and suggest that the Home CA should be run in a trustworthy execution environment. Similarly, Kuntze et al. [13] have described a system for using non-migratable TPM keys for device identification in smart grid environments. However, their system does not perform any verification when the key is used.

One of the examples presented by McCune et al. [9] is that Flicker could be used to protect the private key of a CA. In a similar manner to our system, the key was sealed using the TPM and the signing operation took place within the trusted environment. However, no verification was done on either the requesting application or the information being signed. Since unauthorized use of the system could be rectified at a later stage, the focus of the Flicker example was on the minimization of the TCB size and the attestation of the PAL to a third-party. Our system differs in that the primary focus is on preventing any unauthorized use of the private key and automating the process of using this key.

Bugiel and Ekberg [11] have developed an application specific implementation of a Mobile Trusted Module (MTM). Their implementation is also based on Flicker and uses IMA for application identification. Although their system provides a wider variety of functionality, this increases the size and complexity of the components which are executed in the trusted environment. In contrast, our system is designed to perform a specific operation and so can include additional verification procedures to achieve the security and privacy requirements.

Various solutions have been proposed using a TC measured boot process and virtualization to separate the TCB from untrusted applications. The *TruWallet* system by Gajek et al. [14] provides a means of securely storing users' web authentication credentials by using a security kernel to isolate the *Wallet* functionality from the untrusted OS and web browser. Cesena et al. [15] have proposed a similar approach by using a sealing proxy which is again part of the static TCB.

A *TPM Engine* has been developed for OpenSSL[3]. This engine supports the generation of TPM keys and the use of these keys in an SSL or TLS handshake. However, since it does not include a trusted execution environment, the OpenSSL TPM Engine does not support the verification procedures used in our system.

Goldman et al. [16] as well as Gasmi et al. [17] have presented tangentially related work about the combination of TC technologies and TLS communication. Goldman et al. [16] provide a solution for linking TCG attestation to secure tunnel endpoints to ensure the integrity of the endpoint and prevent relay attacks. Gasmi et al. [17] take this further and provide an implementation proposal for the combination of TC technologies and the TLS protocol. However, both of these systems depend on static roots of trust and so are arguably less practical for use on unmanaged home devices and appliances.

7 Conclusion

Secure communication between devices is a critical aspect of smart grid security, particularly in the smart home environment. In the smart home, each device would have a unique private cryptographic key used for device authentication. These keys must therefore be protected against theft or misuse. Although various mechanisms exist for protecting private keys, most of these require interaction with the user and so are not suitable for use in the smart grid environment due to the level of automation required. We have proposed a system architecture for protecting device private keys in this context using Trusted Computing technologies. In our system, the private key never leaves the trusted execution environment and can only be used by a predefined set of applications. In order to address potential privacy concerns arising from this type of strong device authentication, our system ensures that the key can only be used to establish connections to authorized network end-points. We have developed a proof of concept implementation of our system and demonstrated that our architecture satisfies the defined security, privacy and functional requirements. Overall, the architecture and proof of concept implementation presented in this work are promising steps towards protecting device private keys and achieving secure communication in the smart grid.

[3] `https://github.com/ThomasHabets/openssl-tpm-engine`

Acknowledgements. The research described in this paper was conducted as part of the *Future Home Networks and Services* project at the University of Oxford, funded by British Telecommunications. The first author has also received funding from the Foreign and Commonwealth Office of the United Kingdom as a Chevening Scholar.

References

1. European Commission: Eurostat: Final Energy Consumption, by Sector (2010)
2. National Institute of Standards and Technology (NIST): NIST Special Publication 1108R2: NIST Framework and Roadmap for Smart Grid Interoperability Standards, Release 2.0. Technical report (2012)
3. Baumeister, T.: Adapting PKI for the smart grid. In: 2011 IEEE International Conference on Smart Grid Communications (SmartGridComm), pp. 249–254 (2011)
4. Metke, A.R., Ekl, R.L.: Security Technology for Smart Grid Networks. IEEE Transactions on Smart Grid 1(1), 99–107 (2010)
5. Nützel, J., Beyer, A.: How to Increase the Security of Digital Rights Management Systems Without Affecting Consumer's Security. In: Müller, G. (ed.) ETRICS 2006. LNCS, vol. 3995, pp. 368–380. Springer, Heidelberg (2006)
6. Trusted Computing Group: TPM Main Specifications Part 1: Design principles, Part 2: TPM structures, Part 3: Commands. Version 1.2, Revision 116 (2011)
7. Kühn, U., Kursawe, K., Lucks, S., Sadeghi, A.-R., Stüble, C.: Secure data management in trusted computing. In: Rao, J.R., Sunar, B. (eds.) CHES 2005. LNCS, vol. 3659, pp. 324–338. Springer, Heidelberg (2005)
8. Intel: Intel Trusted Execution Technology (Intel TXT): Measured Launch Environment Developer's Guide. Technical report (2011)
9. McCune, J.M., Parno, B.J., Perrig, A., Reiter, M.K., Isozaki, H.: Flicker: an execution infrastructure for TCB minimization. In: Eurosys 2008 Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems, vol. 42, pp. 315–328 (April 2008)
10. Sailer, R., Zhang, X., Jaeger, T., van Doorn, L.: Design and implementation of a TCG-based integrity measurement architecture. In: Proceedings of the 13th Conference on USENIX Security Symposium, vol. 13. USENIX Association (2004)
11. Bugiel, S., Ekberg, J.E.: Implementing an application-specific credential platform using late-launched mobile trusted module. In: Proceedings of the Fifth ACM Workshop on Scalable Trusted Computing, STC 2010, pp. 21–30. ACM Press, New York (2010)
12. Kinkelin, H., Holz, R., Niedermayer, H., Mittelberger, S., Carle, G.: On Using TPM for Secure Identities in Future Home Networks. In: Security in NGNs and the Future Internet, vol. 3, pp. 1–13 (January 2010)
13. Kuntze, N., Rudolph, C., Bente, I., Vieweg, J., von Helden, J.: Interoperable device identification in Smart-Grid environments. In: 2011 IEEE Power and Energy Society General Meeting, pp. 1–7. IEEE (July 2011)
14. Gajek, S., Löhr, H., Sadeghi, A.R., Winandy, M.: TruWallet: trustworthy and migratable wallet-based web authentication. In: Proceedings of the 2009 ACM Workshop on Scalable Trusted Computing, STC 2009, pp. 19–28. ACM (2009)

15. Cesena, E., Ramunno, G., Vernizzi, D.: Secure storage using a sealing proxy. In: Proceedings of the 1st European Workshop on System Security, EUROSEC 2008, pp. 27–34. ACM Press, New York (2008)
16. Goldman, K., Perez, R., Sailer, R.: Linking remote attestation to secure tunnel endpoints. In: Proceedings of the First ACM Workshop on Scalable Trusted Computing, STC 2006, pp. 21–24. ACM Press, New York (2006)
17. Gasmi, Y., Sadeghi, A.R., Stewin, P., Unger, M., Asokan, N.: Beyond secure channels. In: Proceedings of the 2007 ACM Workshop on Scalable Trusted Computing, STC 2007, pp. 30–40. ACM Press, New York (2007)