A Novel Architecture for a Secure Update of Cryptographic Engines on Trusted Platform Module

***Abstract*—Trusted computing is gaining an increasing acceptance in the industry and finding its way to cloud computing. With this penetration, the question arises whether the concept of hard-wired security modules will cope with the increasing sophistication and security requirements of future IT systems and the ever expanding threats and violations. So far, embedding cryptographic hardware engines into the Trusted Platform Module (TPM) has been regarded as a security feature. However, new developments in cryptanalysis, side-channel analysis, and the emergence of novel powerful computing systems, such as quantum computers, can render this approach useless. Given that, the question arises: Do we have to throw away all TPMs and loose the data protected by them, if someday a cryptographic engine on the TPM becomes insecure? To address this question, we present a novel architecture called Sustainable Trusted Platform Module (STPM), which guarantees a secure update of the TPM cryptographic engines without compromising the system's trustworthiness. The STPM architecture has been implemented as a proof-of-concept on top of a Xilinx Virtex-5 FPGA platform, demonstrating a test case with an update of the fundamental hash engine of the TPM.**

***Keywords*-Field programmable gate array, Cryptography, Trusted Platform Module, Secure Update, Trustworthiness**

I. Introduction and Related Work

Trusted Computing is an emerging technology, developed and promoted by the Trusted Computing Group (TCG), which aims at building trustworthy computing platforms. The Trusted Platform Module (TPM) forms the root-of-trust while executing critical security functions such as integrity measurement, remote attestation, binding, and sealing. Today's TPMs are micro-controller based chips with hard-wired engines for various cryptographic schemes such as RSA, SHA-1, and HMAC as specified in TPM component architecture by TCG [1]. Using hard-wired cryptographic engines, storing cryptographic root keys on hardware, and tying itself to the motherboard, the TPM provides hardware-based security to the system artifacts such as data, certificates, passwords and other cryptographic keys. For example, in a personal computer (PC) without TPM, encrypted data and encryption keys are usually stored on the same hard drive. In contrast, a TPM-based PC stores the encryption keys on the TPM and prevents an unauthorized access to the data. Although the TPM is predominantly used in workstations and servers [2], some approaches already exist to map the specification to embedded systems, reconfigurable architectures, and mobile devices [3], [4], [5]. TCG also publishes platform specific profiles used as a common yard stick for evaluating devices that incorporate TCG technology.

However, it is well-known that cryptographic schemes, also those embedded in TPMs, have always been subject to persistent cryptanalysis and recently to side-channel analysis either by malicious attackers or by the research community. For instance, Wang et al. [6] showed the collision search attacks for the SHA-1 algorithm and Finke et al. [7] conducted a side-channel attack on the RSA key generator. Further, in [8], Bruschi et al. have presented a replay attack during the execution of the TPM authorization protocol that compromises the correct behavior of the trusted platform. Also, Sadeghi et al. have tested several TPM chips for compliance to TCG specifications and were able to find weaknesses with those chips as described in [9].

Considering these and other threats and violations, and following the general recommendation regarding the necessity of updating cryptographic algorithms, e.g., those published by NIST [10], the idea of hard-wiring the TPM security engines began to be questioned. This can be seen from the specification of the next generation TPM (called TPM.next) by TCG. In particular, this specification allows the replacement of cryptographic algorithms in case of a compromise or even for boosting the TPM performance [11]. In spite of this agreement on their necessity, technical solutions for the update of TPM cryptographic engines have not been proposed in the literature, so far.

In this paper we present a novel generic architecture for updating the cryptographic engines used by the TPM in case of a compromise. For this an in-depth analysis of current TPMs, their security functions, and threats is carried out first. Based on this analysis a novel architecture for TPM, which we refer to as the Sustainable Trusted Platform Module (STPM) as depicted in Figure 1, is proposed. Relying on this architecture, an update procedure is then defined. A comprehensive test case for the SHA-1 engine update on STPM is presented and evaluated next. To regain the trust in the system, fundamental post-update measures are specified.

II. Threats to the TPM and their Implications

This section discusses the threats to the TPM and their implications using a threat matrix. Also, the security functions provided by the TPM are described to group them with the corresponding cryptographic engines of the TPM.

A. Threat Matrix

Table I gives an overview of the individual threats a TPM has to deal with, the affected components, and the countermeasures to the posed threats.

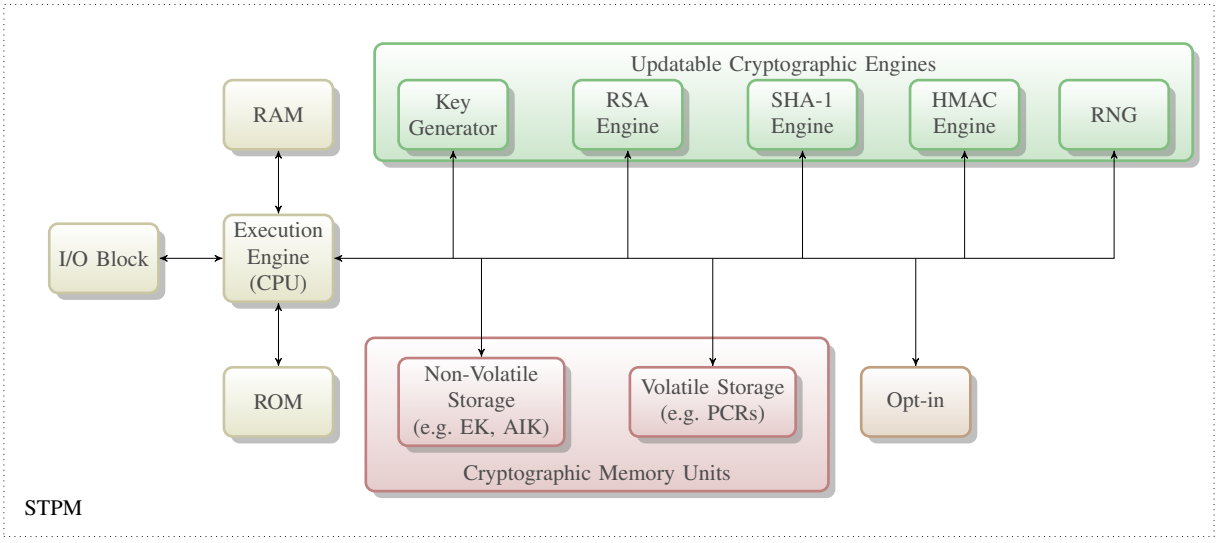

Fig. 1: Proposed Sustainable TPM Architecture

B. Security Functions and Cryptographic Engines

TPM provides the following four main security functions:

Integrity Measurement: With this function the platform configuration and the processes running on it are measured by determining their hash values. These values are stored inside the Platform Configuration Registers (PCRs). A history of events is kept in the Stored Measurement Log (SML).

Remote Attestation: The trustworthy reporting of the platform configuration to a remote challenger is called remote attestation. The TPM provides a set of PCR values representing the system state, which are signed using the Attestation Identity Key (AIK). The remote party is able to compare the measurement results with reference values that indicate the trusted platform configurations. These values are usually listed in a public Reference Measurement List (RML).

Binding: This function binds data (usually the cryptographic keys) to a given platform. It uses asymmetric encryption to store the keys in the key hierarchy managed by the particular TPM.

Sealing: This function expands the binding function by tying it to the platform state. Therefore, the data which has been encrypted at some platform state, can only be decrypted if the platform is exactly in the same state.

All the aforementioned security functions rely on one or more cryptographic engines. Therefore, breaking one of these engines leads to a damage of at least one of the security functions as detailed in Table II. For instance, if the SHA-1 engine is attacked as presented in [6], neither of the security functions will be available except for the binding function. Note that the HMAC engine is not listed in Table II as it does not directly support any of the main security functions. However, HMAC is essential for command authentication and therefore it is intrinsically essential for all the security functions. HMAC uses the same hash engine already available on the TPM. Thus, in the case of violation by the hash engine, all the security functions are compromised, as commands can not be securely issued anymore.

III. Requirements for an Algorithm Update on STPM

This section addresses the architectural and security requirements for an STPM design. A specification of the platform, definition of the protection profile, and the assumptions for an update environment using the STPM are presented.

A. Platform Specification

The task of updating a cryptographic algorithm on a TPM requires a special architecture to support secure updates. It is known that a conventional TPM is in general an Application Specific Integrated Circuit (ASIC) implementation and therefore cannot be updated after deployment. Given that, the STPM should feature dynamic (updatable) regions for loading new cryptographic engines. In contrast, the execution engine to process the TPM commands and the update algorithm have to run uninterruptedly and therefore they should reside in a static (non-updatable) region. In addition, the sensitive cryptographic key information, such as the Endorsement Key (EK), the Storage Root Key (SRK), and the Certificates [12], are stored in the non-volatile memory in a conventional TPM. Further, the PCRs storing the platform configuration are resettable and should be stored in volatile memory. Therefore, the STPM architecture features both static and dynamic regions, along with volatile and non-volatile memory.

Although reconfigurable devices such as FPGAs seem to be a candidate platform for an STPM design, they do not satisfy all the stated requirements. Specifically, most modern FPGAs are SRAM-based and do not support permanent storage. Some FPGAs, such as the Spartan-3AN family from Xilinx, contain on board non-volatile flash memory to keep configuration data. Such FPGAs, however, belong to the low-price class and do not support partial reconfiguration as needed for

Threat	Affected Components and Functions	Countermeasure
Side-Channel attacks on RSA	RSA, keygen., Volatile and Non-Volatile Memory	Side-Channel aware RSA Implementation
Weak RSA Implementation	RSA, keygen., Volatile and Non-Volatile Memory	RSA with longer key
Broken RSA	RSA, keygen., Volatile and Non-Volatile Memory	Non RSA solution
Broken RSA Key Generator	EK, SRK, Key Hierarchy	New Key Generator
Broken SHA-1	PCRs, Attestation, TPM_SHA1 commands	New Hash function
Side-Channel attacks on HMAC	Secret Key, User_Auth_Data, Integrity	Side-Channel aware HMAC Implementation
Broken HMAC	User_Auth_Data, Integrity	New HMAC
Side-Channel attacks on TRNG	Random Numbers, keygen., Nonces	New RNG (such as CSRNG)
Insecure Communication Interface (LPC)	Keys, PCRs	Secure Communication Interface & Protocols
Broken Firmware (ROM)	All TPM_Commands	New Firmware

TABLE I: Threats and Necessary Countermeasures

Security Function	Involved Cryptographic Engines
Integrity Measurement	SHA-1
Remote Attestation	RSA, SHA-1
Binding	RSA
Sealing	RSA, SHA-1

TABLE II: Mapping of TPM Security Functions to TPM Building Blocks

the STPM architecture. Structured ASICs [13] contain both static and reconfigurable parts. This type of integrated circuit, however, is mask-programmable, an operation which can only be performed by the chip vendor. The target platform for STPM must therefore be a novel one which combines the techniques of ASICs, parital reconfiguration, and hosts volatile and non-volatile memories.

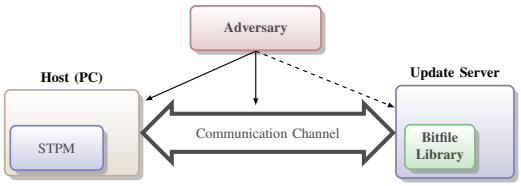

Fig. 2: Adversarial Model

For the purposes of an in-depth analysis and a prototypical design evaluation, we consider an FPGA-based STPM that includes measures to protect external non-volatile memory as presented in the sequel.

B. Protection Profile

A special protection profile is considered for the FPGA-based STPM architecture to achieve a secure algorithm update and additionally maintain the security level of a conventional TPM. The lack of non-volatile memory (NVM) on the FPGA can be circumvented by applying the protocol as proposed by Schellekens et al. in [14]. Using their approach an authenticated communication between the external multi-programmable NVM and the system-on-chip is established. Although an external memory integration is not always the case, in future implementations of the STPM the non-volatile memory will reside on the chip itself. Also, in [15], Feller et al. have presented an algorithm for secure and authenticated updates in an FPGA-based embedded system. Their approach uses a block cipher (AES) based authentication scheme to protect the IPs (intellectual property) transferred from an update server.

C. Update Environment and Adversarial Model

During STPM update, two parties are involved: the STPM residing in a computer system and the update server maintaining the bitfile library, see Figure 2. The computer system that embeds the STPM is assumed to be part of an enterprise environment and is administered by the dedicated staff. The update server delivers the new cryptographic engines to be loaded onto the STPM over an insecure network. It is administered by the STPM manufacturer, who initiates and provides all the update services. Further, the update server is assumed to be run in a secure environment. Thus, attacks on the server itself are not addressed in this paper. The adversary in this model can be an active attacker who may interfere with the communication channel and perform attacks on the STPM. According to the taxonomy of attacks given by Popp [16], we assume that the attacker is able to perform both classical cryptanalysis and implementation attacks. While the classical cryptanalysis includes attacks such as cloning, replay attacks, and malicious updates, the implementation attacks exploit side-channel analysis, probing attacks, fault analysis, and reverse engineering.

The communication between the STPM and the update server exploits well-established encryption and authentication

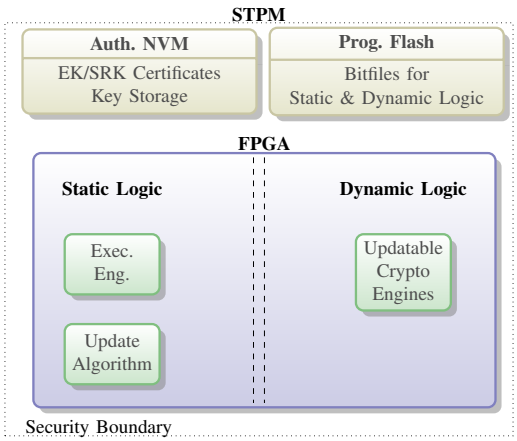

Fig. 3: STPM Architecture based on an FPGA

techniques to overcome classical cryptanalysis attacks. However, possible physical and implementation attacks on the STPM during shipping and after deployment [17] are not considered in the scope of this paper.

D. The STPM Architecture

Based on the secure design requirements as mentioned in the previous sections, a novel specific architecture for the STPM using an SRAM-based FPGA is described in the following. The FPGA inside the STPM supports the partial reconfiguration feature, to accommodate an uninterruptedly running static region and a run-time reconfigurable dynamic region simultaneously, as depicted in Figure 3. The dynamic region holds the locations for updatable cryptographic engines. In contrast, the static region of the FPGA is loaded upon initialization with the non-updatable components. Additional components, i.e., non-volatile memory (NVM) and programmable flash required for an update, are also available inside the STPM. The persistent cryptographic data is stored in NVM and the bitfiles for loading into static and dynamic regions of the FPGA are stored in a flash memory. Although we present two different types of memory for storage purposes here, the NVM and the flash memory may technically reside within one physical component. The storage for bitfiles is rewritable to support updating the bitfiles from an update server as needed. All the three components of the STPM, i.e., the FPGA, the authenticated NVM, and the flash, reside in a single secure package inside the PC. This secure package, often referred to as security boundary, avoids physical attacks on the STPM and the communication between different components inside the STPM. The most important feature of the STPM is to securely support hardware update (i.e., cryptographic engine update) similar to the well-known software update on a PC. Thereby, it provides a flexible and scalable design of cryptographic engines on the STPM, which is in general not the case with conventional TPMs.

IV. Test Case: Update of the SHA-1 Engine

This section presents and discusses an important test-case relating to the update of the SHA-1 engine on the STPM using a special data format and a well-defined update algorithm. In addition to measuring the platform state, the SHA-1 engine is used for computing signatures and creating key blobs. For example, in the remote attestation process, a digital signature is computed for the AIK by the privacy certification authority (a trusted third party). AIK is an asymmetric session key from the attester, which can be validated later with the help of a trusted third party by providing the AIK certificate. The private part of this AIK is used for signing the PCRs, which are sent to the challenger along with SML and the AIK certificate. The challenger takes the PCR values and the AIK certificate for verification and for establishing the communication with the attester. If the SHA-1 engine is broken, the newly appended PCR content is not valid any more and all the generated signatures and key blobs become useless. Additionally, some TPM implementations feature a HMAC engine, which uses the same SHA-1 engine to check the integrity and authenticity of received packages and to authenticate TPM commands.

A. Update Algorithm

In the following, a description of the update data format is given first followed by an explanation of the update algorithm itself. For the purpose of authenticating the update data (U_{DATA}), we use a cipher-based message authentication code (CMAC) as described in [18]. The cipher used in this particular case is the AES block cipher and the corresponding hash function is the AES-hash.

For the next specification we introduce the following notation:

$config$	The configuration file for the new hash engine
k	Previously exchanged symmetric encryption key for IP protection
$E(k, M)$	Symmetric encryption of message M with key k
k_{hmac}	Key used for generating the HMAC
$CMAC(k_{hmac}, M)$	CMAC of message M computed with key k_{hmac}

Using this notation, the update data is defined as:

$$U_{DATA} \Leftrightarrow (E(k, config), CMAC((k_{hmac}, E(k, config))))$$

Algorithm 1 details the procedure for loading the new hash engine onto the STPM. The CMAC $CMAC((k_{hmac}, E(k, config)))$ is computed using the key k_{hmac} on the server side and sent to the STPM along with the encrypted configuration $E(k, config)$. Verifying this CMAC by the STPM requires the storage of the HMAC key k_{hmac} in the non-volatile memory within the STPM. To decrypt the update data, the STPM must be equipped both with the symmetric cryptographic engine (AES) and the corresponding symmetric key k. In addition to the symmetric key and the HMAC key, the non-volatile memory has to store

the STPM keys and certificates as depicted in Figure 3. Upon receiving the update data, they are checked for integrity and authenticity using CMAC. Then the update data is decrypted in order to subsequently configure the STPM with the new hash engine. Finally, an activation of this engine occurs.

Algorithm 1 HASH Engine Update Algorithm

Require: U_{DATA}, symmetric key k, hmac key k_{hmac}
1: Load U_{DATA}
2: Compute $CMAC((k_{hmac}, E(k, config)))$ and compare with attached CMAC
3: Decrypt $E(k, config)$
4: Configure STPM with $config$
5: Activate new hash engine
6: **return** $Update\_complete$.

B. Updating the SHA-1 Engine on the STPM

Partial reconfiguration (PR) is supported by several SRAM-based FPGAs. PR means that a portion of the FPGA's fabric can be reconfigured while the rest is executing the previous configuration. For that purpose the FPGA is floorplaned into dynamic regions, also denoted as partial reconfigurable regions (PRR), and into a static region. Modules placed inside the PRR are denoted as partial reconfigurable modules (PRM). The PRMs assigned to a PRR are mutually exclusive, i.e., only one PRM can be assigned to a given PRR at a given time.

In Figure 4 the dynamic logic of the FPGA contains two PRRs, so, two distinct PRMs may be placed in each of these PRRs. The static logic of the FPGA is loaded with the application parts, which should run uninterruptedly such as the controller responsible for loading PRMs. In the proposed architecture, the static logic of the FPGA within the STPM contains all the non-updatable components such as the update algorithm and the execution engine. The dynamic logic contains the regions for loading new replacements of cryptographic engines, see Figure 3 and Figure 4 for details.

Recall that the partial bitfiles (new cryptographic engines) to be loaded into the dynamic region are available from the update server only, see Figure 4. The update process starts with transferring the partial bitfiles to the STPM flash memory. The full bitfile for the static region is already available in the flash memory. The full bitfile is first loaded into the static region upon initialization. Then, the partial bitfiles are transferred to the dynamic region. A subsequent replacement of partial bitfiles is done according to the steps given in Algorithm 1.

C. Implementation Results

A proof-of-concept implementation for the SHA-1 update on STPM is evaluated on a Xilinx Virtex-5 LX110T FPGA. Table III summarizes the overall resource consumption of the static logic implementation, i.e., the update algorithm and the execution engine. The major parts of this algorithm are the controller for command execution and the AES core performing all cryptographic tasks. The internal configuration access port (ICAP) is utilized for loading the new cryptograhic engines into the dynamic logic. Also, the control flow and the corresponding command execution to provide the TPM functionality, can be run on a softcore processor such as Xilinx's MicroBlaze. All together, the static logic consumes only a fraction of the available FPGA resources.

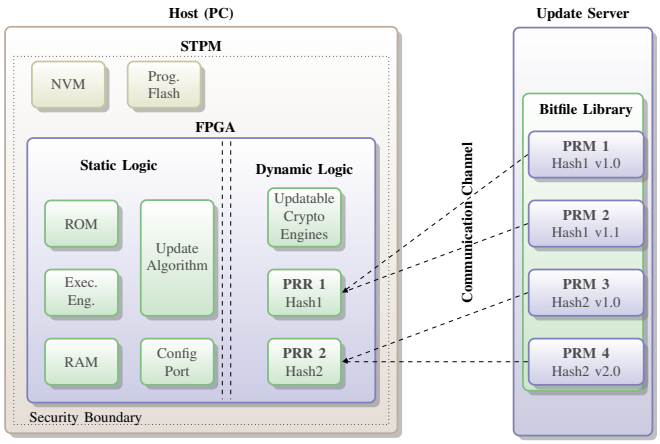

Fig. 4: Updating the SHA-1 Engine on the STPM

	Reg.	LUT	36Kbit BRAM
AES-128	524	899	5
Hash-Core	289	138	0
HMAC-Core	294	184	0
Controller	662	453	0
PR ICAP	170	168	2
Update Algorithm	1939	1842	7
MicroBlaze CPU (Exe. Engine)	2326	2704	4
Total Static Logic	4265	4546	11

TABLE III: Resource Consumption of Static Logic of STPM

	Register	LUT	36Kbit BRAM	max Freq. (MHz)
RSA	1275	1924	0	78.927
SHA-1	1013	1754	0	156.145
HMAC	1722	2353	0	156.145
RNG	1424	1248	5	283.688
Total TPM Engines	5434	7279	5	

TABLE IV: Resource Consumption of Cryptographic Engines

Table IV summarizes the overall resource consumption of all the updatable cryptographic algorithms as depicted in Figure 1. These algorithms are loaded into the dynamic logic of the FPGA during the STPM operation. This logic region of the FPGA occupies the major portion of the FPGA because

the static logic is very small in size. Therefore, the new crypto algorithms to be loaded are provided with enough resources. Thus, with reference to Table III and Table IV, the current STPM design implementation occupies only a few of the available resources on the considered FPGA.

V. Regaining Trust

In this section we describe how to regain trust in the system after a hash engine has been updated. In Table II it was outlined that all the security functions except for binding are directly dependent on the hash engine. Therefore, to maintain a trusted environment, we present the necessary measures based on the specific requirements of each security function for trust enforcement.

A. Integrity Measurement

Integrity measurement is a basic function to compute the hash values stored in PCRs, which represent the current system state. In the case of an engine update, this representation of the system state is no more valid, because it had been computed with the old hash engine. This leads to an undefined representation of the system state, where it cannot be determined whether the system is in a trustworthy state or not. Even if there are no changes in hardware or software, the current PCR contents will differ from future values computed with the updated hash engine. Also, due to the fact that the measurement has been performed using the old hash engine, the PCR contents are prone to collision attacks. However, after updating the hash engine, the whole platform configuration is recomputed and stored in the PCRs. Because most of the PCRs are not resettable, the system has to be rebooted before the new measurements can be considered as trustworthy. To indicate the trustworthiness, the system state has again to be compared to the reference values.

B. Remote Attestation

A trusted platform is able to attest the current system state to any requester. Usually, the trusted system states are stored in RMLs (Reference Measurement Lists) generated by the IT-department. In the case of an update of the hash engine, these RMLs become invalid. Therefore, the reference list has to be recomputed by means of the updated hash engine. Furthermore, the contents of the PCRs have to be reevaluated in order to reflect the update in the stored hash values. Remote attestation is the only function, which is able to directly distinguish between trustworthy and untrustworthy system states.

C. Binding

Binding is used to encrypt sensible data with keys, which are available on a specific TPM only. The binding function can be considered to be independent of the trust someone puts into the system. Of course, the confidential data has to be protected properly for which the binding procedure itself does not provide any measures to reflect the trustworthiness of the current state. Therefore, the update of the hash engine does not affect the binding function directly.

D. Sealing

Sealing uses the current system state to protect sensible data. Sealed data can only be unsealed if the system is in the same state as during the sealing procedure. After an update of the hash engine the data cannot be unsealed, because the PCR content has changed. To tackle this problem, a resealing process must be performed as detailed below.

E. Data Resealing

Data resealing is necessary after a hash engine update to maintain the availability of sealed data. Data resealing of remote data is necessary if the PCR values change, but the system is still in a trusted state. This usually happens only when software updates were performed.

As the sealing function relies on the content of PCRs, we propose a procedure to reseal data to the updated PCR values that represent the current state of the system.

$$reseal \Leftrightarrow unseal(Hash_{old}) + seal(Hash_{new}) \qquad (1)$$

Equation 1 shows that prior to sealing the data with the new hash engine ($Hash_{new}$), it has to be unsealed first by using the outdated hash function ($Hash_{old}$). But the PCRs currently contain the measurements of the system state with the outdated hash engine. Therefore, to perform resealing, the current system state has to be made available in both representations computed by $Hash_{old}$ and $Hash_{new}$, respectively. To achieve this, along with the two hash engines, two sets of PCRs (located in volatile memory), are needed to store both representations of the system state. For this purpose the STPM architecture features the operation of two hash engines in parallel, as detailed in Figure 4, to accelerate resealing. After a reboot action, the resealing procedure is performed according to Equation 1.

Although the data resealing aspect has been considered by Kühn et al. in [19], they concentrate mainly on the hardware and software life cycle as commonly found in enterprises. Their procedures assume that the system state is trustworthy even after a software update, which does not cover the case of a hash engine hardware update that affects the integrity measurement process itself. Therefore, to cope with the update of components of the TPM such as the hash engine, the procedures presented in [19] are not applicable. Instead, new dedicated measures as described in this work are required.

F. HMAC

Every STPM command is being authenticated using the HMAC function based on the protocols OSAP and OIAP as specified by TCG. As the HMAC function uses the hash engine, it has to be fitted to the new hash engine in the case of an update. Secure communication to the STPM can be continued using two different approaches. First, all new messages, which have an HMAC attached, have to use the new hash function to create the authentication code. Running sessions using OIAP or OSAP can be terminated, if the hash function has been exchanged and the session has to be reinitiated afterwards. The overhead introduced by this

procedure is tolerable, because the algorithm update will not happen too frequently. The second approach is to use the mode of operation introduced by Bellare et al. in [20]. They presented a proof for the usage of HMAC functions even though the hash engine looses its collision resistance property. Therefore, running sessions can still exploit the old hash engine, while newly initiated sessions operate the updated hash engine.

G. Signatures

Signatures are used during the remote attestation process to attest the state of the system to a remote challenger. For this, the PCR values are signed using the AIK and sent to the challenger for verification. In general, signature creation uses the hash engine, as presented in [21], whereas collisions of the hash engine directly translate to forgeries. However, in this paper the signatures are created using the mode of operation presented in [22] and therefore are considered to be valid despite the fact that the hash engine is not collision resistant. Even though the digital signatures are still valid, new messages should be signed using the new hash engine. The signatures have to carry some information showing the algorithm that was used to create it, as done in X.509 [23] certificates. Also, all certificates that are used during the boot sequence of the system might have to be updated to fit the new signature algorithm.

VI. Conclusion

A novel architecture for updating the cryptographic engines embedded in a TPM was presented in this paper. We first analyzed possible threats to the TPM and the implications of these threats on its security functions. From this analysis, we investigated the architectural and security requirements for performing an update of cryptographic engines on the STPM. Relying on the proposed STPM architecture, we defined a special format for update data and designed an algorithm for securely updating the cryptographic engines. In addition to providing an updatability feature, the STPM stays secure against the possible attacks, which were formulated using assumptions on both update environment and the adversaries. Later, using the update algorithm and the STPM, a complete test case for updating the SHA-1 engine with the resource consumption values on a Virtex5 FPGA has been presented. Based on the implications caused by the update of the hash engine, we performed an analysis of the steps to be taken to regain the trust in the platform. The implications resulting from updating other cryptographic engines are part of future work.

References

[1] Trusted Computing Group, Incorporated, "TCG specification architecture overview," http://www.trustedcomputinggroup.org/resources/tcg_architecture_overview_version_14, 2007.

[2] IBM Research, Inc., "Virtual Trusted Platform Module," http://domino.research.ibm.com/comm/research_projects.nsf/pages/ssd_vtpm.index.html, 2008.

[3] T. Eisenbarth, T. Guneysu, C. Paar, A. Sadeghi, M. Wolf, and R. Tessier, "Establishing Chain of Trust in Reconfigurable Hardware," in *Proceedings of IEEE Symposium on Field-Programmable Custom Computing Machines*, 2007, pp. 289–290.

[4] B. Glas, A. Klimm, O. Sander, K. Muller-Glaser, and J. Becker, "A System Architecture for Reconfigurable Trusted Platforms," in *Proceedings of IEEE/ACM Int. Conference on Design, Automation and Test in Europe*, 2008, pp. 541–544.

[5] J.-E. Ekberg and M. Kylänpää, "Mobile Trusted Module (MTM) - an Introduction," Nokia Research Center, Helsinki, Tech. Rep. NRC-TR-2007-015, Nov. 2007, http://research.nokia.com/files/NRCTR2007015.pdf.

[6] X. Wang, Y. Yin, and H. Yu, "Finding collisions in the full SHA-1," in *Advances in Cryptology–CRYPTO 2005*. Springer, 2005, pp. 17–36.

[7] T. Finke, M. Gebhardt, and W. Schindler, "A New Side-Channel Attack on RSA Prime Generation," in *Proceedings of the 11th International Workshop on Cryptographic Hardware and Embedded Systems*. Springer-Verlag, 2009, pp. 141–155.

[8] D. Bruschi, L. Cavallaro, A. Lanzi, and M. Monga, "Replay attack in TCG specification and solution," in *Proceedings of the 21st Annual IEEE Computer Security Applications Conference*, 2005, pp. 11–137.

[9] A. Sadeghi, M. Selhorst, C. Stüble, C. Wachsmann, and M. Winandy, "TCG inside?: A note on TPM specification compliance," in *Proceedings of the first ACM workshop on Scalable trusted computing*. ACM, 2006, pp. 47–56.

[10] National Institute of Standards and Technology, *Recommendation for the Transitioning of Cryptographic Algorithms and Key Lengths*, http://csrc.nist.gov/publications/drafts/800-131/draft-sp800-131_spd-june2010.pdf, 2010.

[11] Trusted Computing Group, Incorporated, "Features under consideration for the next generation of tpm," http://www.trustedcomputinggroup.org/resources/summary_of_features_under_consideration_for_the_next_generation_of_tpm, 2009.

[12] ——, "Trusted Platform Module (TPM) specifications," http://www.trustedcomputinggroup.org/resources/tpm_main_specification, 2010.

[13] K. Wu and Y. Tsai, "Structured ASIC, evolution or revolution?" in *Proceedings of the ACM International Symposium on Physical Design*. ACM, 2004, pp. 103–106.

[14] D. Schellekens, P. Tuyls, and B. Preneel, "Embedded Trusted Computing with Authenticated Non-Volatile Memory," in *Proceedings of Trusted Computing - Challenges and Applications*, 2008, pp. 60–74.

[15] T. Feller, S. Malipatlolla, D. Meister, and S. A. Huss, "TinyTPM: A Lightweight Module aimed to IP Protection and Trusted Embedded Platforms," in *Proceedings of IEEE International Symposium on Hardware Oriented Security and Trust (HOST 2011)*, Jun. 2011.

[16] T. Popp, "An Introduction to Implementation Attacks and Countermeasures," in *Proceedings of IEEE/ACM International Conference on Formal Methods and Models for Co-Design (MEMOCODE'09)*, 2009, pp. 108 –115.

[17] Trusted Computing Group, Incorporated, "TCG protection profile pc client specific trusted platform module tpm family 1.2; level 2," http://www.commoncriteriaportal.org/files/ppfiles/pp0030b.pdf, 2008.

[18] National Institute of Standards and Technology, *Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication*, http://csrc.nist.gov/publications/nistpubs/800-38B/SP_800-38B.pdf, 2005.

[19] U. Kühn, K. Kursawe, S. Lucks, A.-R. Sadeghi, and C. Stüble, "Secure Data Management in Trusted Computing," in *Proceedings of Cryptographic Hardware and Embedded Systems*, 2005, pp. 324–338.

[20] M. Bellare, "New proofs for NMAC and HMAC: Security without collision-resistance," in *Proceedings of Advances in Cryptology (CRYPTO'06)*, 2006, pp. 602–619.

[21] National Institute of Standards and Technology, "Digital Signature Standard (DSS)," National Institute of Standards and Technology, Tech. Rep., June 2009. [Online]. Available: http://csrc.nist.gov/publications/fips/fips186-3/fips_186-3.pdf

[22] S. Halevi and H. Krawczyk, "Strengthening digital signatures via randomized hashing," *Advances in Cryptology*, pp. 41–59, 2006.

[23] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," RFC 5280 (Proposed Standard), Internet Engineering Task Force, May 2008. [Online]. Available: http://www.ietf.org/rfc/rfc5280.txt