Reconfigurable Trusted Computing in Hardware

Thomas Eisenbarth[1], Tim Güneysu[1], Christof Paar[1],
Ahmad-Reza Sadeghi[1], Dries Schellekens[2], Marko Wolf[1]

[1]Horst Görtz Institute for IT Security, Ruhr-University of Bochum, Germany
[2]K.U. Leuven, ESAT/COSIC, Leuven-Heverlee, Belgium

ABSTRACT

Trusted Computing (TC) is an emerging technology towards building trustworthy computing platforms. The Trusted Computing Group (TCG) has proposed several specifications to implement TC functionalities by extensions to common computing platforms, particularly the underlying hardware with a Trusted Platform Module (TPM).

However, actual TPMs are mostly available for workstations and servers nowadays and rather for specific domain applications and not primarily for embedded systems. Further, the TPM specifications are becoming monolithic and more complex while the applications demand a scalable and flexible usage of TPM functionalities.

In this paper we propose a reconfigurable (hardware) architecture with TC functionalities where we focus on TPMs as proposed by the TCG specifically designed for embedded platforms. Our approach allows for (i) an efficient and scalable design and update of TPM functionalities, in particular for hardware-based crypto engines and accelerators, (ii) establishing a minimal trusted computing base in hardware, (iii) including the TPM as well as its functionalities into the chain of trust that enables to bind sensitive data to the underlying reconfigurable hardware, and (iv) designing a manufacturer independent TPM. We discuss possible implementations based on current FPGAs and point out the associated challenges, in particular with respect to protection of the internal TPM state since it must not be subject to manipulation, replay, and cloning.

Categories and Subject Descriptors: B.7.1 [Integrated Circuits]: Types and Design Styles; C.3 [Special-Purpose and Application-Based Systems]; E.3 [Data Encryption]

General Terms: Design, Measurement, Security.

Keywords: Field Programmable Gate Array (FPGA), Trusted Computing, Trusted Platform Module (TPM).

Our special thanks go to Jean-Pierre Seifert, Berk Sunar, Russell Tessier, and Pim Tuyls.

STC'07, November 2, 2007, Alexandria, Virginia, USA.

1. INTRODUCTION

Trusted Computing (TC) is a promising technology towards building trustworthy computing platforms. A recent initiative to implement TC by extending common computing platforms with hardware and software components is due to the Trusted Computing Group (TCG), a consortium of IT enterprises [24]. The TCG specified the Trusted Platform Module (TPM) which provides a small set of cryptographic and security functions, and is assumed to be the trust anchor in a computing platform. Currently, TPMs are implemented as dedicated crypto chip mounted on the main board of computing devices, and many vendors already ship their platforms equipped with TPM chips. The functionalities provided by the TPM allow to securely bind (sensitive) data to a specific platform meaning that the data is only accessible when the underlying platform has the valid and desired configuration.

However, there are several issues to deal with: first, actual TPM chips are currently mainly available for workstations and servers and rather for specific domain applications, in particular barely not for embedded systems.[1] Second, TPM specifications are continuously growing in size and complexity, and there is still no published analysis on the minimal TPM functionalities that are practically needed. In addition to this, TPM users have to completely trust implementations of TPM manufacturers, e.g., regarding the compliance to the TCG specification. This also demands the user to trust the TPM implementation that no malicious functionalities have been integrated (like trapdoors or Trojans). Finally, the TCG adversary model considers software attacks only, but manipulations on the *underlying* hardware can circumvent any whatsoever sophisticated software security measures. Currently, TPM chips are connected to the I/O system with an unprotected interface that can be eavesdropped and manipulated easily [16].

In this paper, we address most of these issues by proposing a reconfigurable architecture in hardware that allows a scalable and flexible usage of trusted computing functionalities. Our implementation proposal is based on Field programmable Gate Arrays (FPGA). FPGAs are reconfigurable hardware devices offering a flexible solution for integrated hardware architectures. Their size and capabilities have greatly evolved during the last years and have made

[1]At least there exist proposals from Brizek et al. [6] and the TCG [24] for a tailored TPM to support also mobile devices and further approaches [3] to implement TPM hardware functionality into isolated software sandboxes (which in turn would require a fully trustworthy CPU).

them competitive to static ASIC chips. Particularly, recent FPGAs provide a sufficient gate complexity to create complete (Configurable) System on a Chip (CSoC) environments since microprocessor soft cores, crypto accelerators and high-throughput I/O components can be included in a single FPGA design. These flexible FPGA applications are synthesized into bit streams and stored in an external PROM or Flash memory. On power-up of an SRAM-based FPGA[2], the bit stream is loaded into the device since the loaded hardware configuration is lost after system shutdown.

To our knowledge, there has been no proposal for building TC capabilities (e.g., TPM functionalities) in reconfigurable hardware architectures. Our approach allows, amongst other things, to bind a reconfigurable application to the underlying TPM and even to bind any higher layer software to the whole reconfigurable architecture. Based on the asymmetric means of an TCG-conform TPM, this can be used as an effective and flexible protection of Intellectual Property (IP) to provide device-specific application software.

We believe that FPGA devices can provide a promising basis for a variety of TC applications in embedded system environments. For enabling TC functionality on these devices, todays FPGA architectures must be extended and enhanced but the technologies for the required modifications are already available. Please note that we do not primarily focus on the integration of large microprocessors (μP) like commercial Intel Core 2 Duo or AMD Opteron into an FPGA. In fact, our approach assumes embedded applications running on small μPs like ARM known from mobile phones and PDAs.

Contribution. In this paper we propose solutions to extend reconfigurable hardware architectures with Trusted Computing functionalities, e.g., for use in embedded systems. In particular, our architecture allows to include the TPM implementation itself into the so-called chain of trust. Although we aim at solutions compliant to the TCG proposed TPM specification, our architecture can be deployed independently from TCG approach for future developments of TC technology. Besides a vendor-independent and flexible integration of a TPM in embedded systems, our approach provides the advantage to reduce of the trusted computing base to the bare minimum according to the application's needs. This includes specific functionalities to allow for effective protection scenarios with hardware and software IP (on FPGAs).

Outline. In the subsequent Section 3 we introduce the main aspects of the Trusted Computing technology as proposed by the TCG and its weaknesses. Section 4 presents our idea and design for trusted reconfigurable hardware architectures providing TC capabilities, including the resulting advantages. Finally, Section 5 introduces implementation aspects and challenges to be dealt with, while Section 2 provides related work.

2. RELATED WORK

In [15] the authors already identified various challenges with regard to TPM maintenance in case of hardware or software updates. However, the approach does not consider FPGA-based TPMs just as [5] where the authors present a TPM chip with an integrated CRTM. However, it remains open how they manage it to let the platform initialization begin at the CRTM-enabled TPM (unlike hitherto platform initializations starting at the BIOS or CPU), the presented solution is still vulnerable to attacks on the communication link between TPM and the system. Simpson and Schaumont [23] provide an IP protection scheme for FPGAs where hardware and software components authenticate each other based on a Physically Unclonable Function (PUF). While [23] considers PUF as blackbox, Guajardo et al. [12] give an implementation for the PUF and also extend the FPGA-based IP protection protocol. However, both proposals require the availability of an external Trusted Party as well as the integration of a static PUF in the FPGA. This significantly cuts the flexibility with respect to what we are able to provide with a TC-based solution. Drimer [10] proposes an FPGA bit stream authentication mechanism based on two parallel AES engines. However, his approach does not provide any further functionality except AES-based bit stream decryption and authentication. Zambreno et al. [26] provide a further proposal for a software protection system using reconfigurable hardware but the protection of IP contained in bit streams is still an active field of research (see, e.g., [18]).

However, to our knowledge, a holistic approach to transfer the extensive capabilities of Trusted Computing systems to reconfigurable hardware has not been published yet.

3. TCG BASED TRUSTED COMPUTING

This section gives a brief review of the main aspects of Trusted Computing technology as proposed by the TCG [24].

The main TCG specifications are: a component providing cryptographic functions called *Trusted Platform Module* (TPM), a (immutable) part of BIOS (Basic I/O System) called the *Core Root of Trust for Measurement* (CRTM), and the *Trusted Software Stack* (TSS) which is the software interface to provide TC functionalities to the operating system. Many vendors already ship their computing devices with a TPM on the main board and TPM support is also already integrated into commercial operating systems, e.g., to enable hard disk encryption [19], or to measure platform configuration [21]. The TCG issues only functional specifications while implementations are left to vendors.

3.1 Trusted Platform Module (TPM)

Currently, the TPM is a dedicated hardware chip[3] similar to a smart-card that is assumed to be securely bound to the computing device. According to [24], a TPM version 1.2 provides the following features: A hardware-based random number generator (RNG), a cryptographic engine for encryption and signing (RSA) as well as a cryptographic hash function (SHA-1, HMAC), read-only memory (ROM) for firmware and certificates, volatile memory (RAM), non-volatile memory (EEPROM) for internal keys, monotonic counter values and authorization secrets, and optionally, sensors for tampering detection.

3.2 Weaknesses of TPM Implementations

The main hardware-based components CRTM and TPM are assumed to be trusted by all involved parties. According to the TCG specification protection for these components against software attacks are required. However, computing

[2]The choice is due to the more advanced technology deployed for SRAM-based FPGA.

[3]TPM chips are already available, e.g., from Atmel, Broadcom, Infineon, Sinosun, STMicroelectronics, and Winbond.

devices are deployed in a potentially hostile environment where an adversary has full access to the underlying hardware. Thus, certain hardware attacks may undermine the security of the TCG approach. As mentioned before, some TPM manufacturers have already started a third party certification of their implementation with respect to security standards (Common Criteria [8]) to assure a certain level of tamper-resistance (as TPM technology stems from the smart-card technology).

Unprotected Communication Link. Currently, in most TCG-enabled platforms communication channels (buses) between TPM, RAM and microprocessor are unprotected. Hence, even if internal information (cryptographic keys, certificates, etc.) within shielded storage of the TPM cannot be compromised, the communication link can be subject to attacks [16]. Moreover, it is unlikely that future processors itself will include required features (key sharing or exchange) for a establishing a secure channel to the TPM.

Potential Problems with TPM integration. Although an integration of the TPM functionality into chipsets makes the manipulation of the Low-Pin Count (LPC) bus between TPM and microprocessor significantly more difficult and costly, the integration also introduces new challenges: on the one hand an external validation and certification of the TPM functionalities (e.g., required by some governments) will be much more difficult, and on the other hand, users may require computing platforms without TPM functionalities which becomes impossible in case of a static integration.

Moreover, standard microprocessors use an external boot ROM to store their initial boot code as well as the CRTM. Thus, an attacker could switch ROM modules to inject malicious boot code and compromise the chain of trust. Although sophisticated mechanisms can be used to build highly secure TPMs and CRTM and the TCG approach explicitly allows an application to distinguish between different TPM implementations, the vast majority of TPMs will only provide a limited protection against hardware based attacks, due to the trade-off between costs and tamper-resistance. Nevertheless, at least rudimentary tamper precautions and other countermeasures for memory protection have not been considered in the design and manufacturing process. Hence, system designers and developers should be aware of the adversary model and the assumptions underlying the trusted computing architecture and its instantiation. This also is valid for FPGA designs.

4. TRUSTED RECONFIGURABLE HARDWARE ARCHITECTURE

In this section we outline solutions for realizing trusted computing functionalities in reconfigurable hardware, and discuss possible implementations where we follow the TCG approach. However, our architecture can also be used for other possible developments in TC technology.

4.1 Underlying Model

The main parties involved are FPGA *manufacturers*, *hardware IP developers* (e.g., developing the application logic synthesized to a bit stream), *software IP developers* who implement software that runs on the loaded bit stream on the FPGA, *system developers* who integrate hardware and software IP onto an FPGA platform and the *user* who employs the device. All parties trust the FPGA hardware manufacturer, since there is no publicly known efficient mechanism to verify an ASIC implementation for correctness or potential trapdoors. However, IP developers have only limited trust in systems developers, and users have only limited trust in IP and system developers. It is obvious that the entity issuing the update (usually the TPM designer) needs to be trustworthy, or the TPM implementation is subject to certification by some trusted organization.

We assume an *adversary* who can eavesdrop and modify all FPGA-external communication lines, eavesdrop and modify all FPGA-external memories, arbitrarily reconfigure the FPGA, but *cannot* eavesdrop or modify FPGA-internal states. Particularly, we exclude invasive attacks such as glitch attacks, microprobing attacks or attacks using laser or Focused Ion Beam (FIB) to gain or modify FPGA internals. Precautions against other physical attacks such as side channel attack or non-invasive tampering must be taken when implementing the TPM. Furthermore, we do not consider any destructive adversaries which are focusing on denial-of-service attacks, destroying components or the entire system.

4.2 Basic Idea and Design

The basic idea is to include the hardware configuration bit stream(s) of the FPGA in the chain of trust. The main security issue, besides protection of the application logic, is to protect the TPM against manipulations, replays and cloning. Hence, appropriate measures are required to securely store and access the sensitive TPM state $\mathcal{T}$.

In the following we denote a hardware configuration bit stream as B_X with $X \in \{TPM, App\}$ such that B_{TPM} denotes a TPM bit stream and B_{App} an application bit stream. We further define E_X as the encryption of B_X using a symmetric encryption algorithm and a symmetric encryption key $k_{\mathsf{Enc},X}$ such that $E_X \leftarrow \mathsf{Enc}_{k_{\mathsf{Enc},X}}(B_X)$. We define A_X as an authenticator of a bit stream B_X with $A_X \leftarrow \mathsf{Auth}_{k_{\mathsf{Auth},X}}(B_X)$ where $\mathsf{Auth}_{k_{\mathsf{Auth},X}}$ could be for instance a Message Authentication Code (MAC) based on the key $k_{\mathsf{Auth},X}$. We denote the corresponding verification algorithm of an authenticator A_X with $\mathsf{Verify}k_{\mathsf{Auth},X}(B_X, A_X)$. In case that a bit stream has been encrypted to preserve design confidentiality, B_X is replaced by E_X. Thus, the corresponding authenticator A_X becomes $A_X \leftarrow \mathsf{Auth}_{k_{\mathsf{Auth},X}}(E_X)$. According to [4], such an *Encrypt-then-MAC* authenticated encryption scheme provides the strongest security (with respect to the two other possible schemes *MAC-then-Encrypt* and *Encrypt-and-MAC*). We finally define C_X as an unique representative of B_X's configuration, e.g., a cryptographic hash value[4] which can be based on a block cipher [17].

Figure 1 shows our high-level reconfigurable architecture. The bit streams B_{App}, B_{TPM} of the application and the TPM core without any state $\mathcal{T}$ are stored authenticated (and encrypted) in the *external (untrusted) memory EM*.

The *FPGA Control* logic allows partial hardware configuration[5] of the FPGA fabric to load the TPM and the application independently using the LOAD and CONFIGURE interfaces.

[4]If C_{TPM} is a hash value, it represents the measurement conform to the TCG approach. However, alternative approaches may use for C_{TPM}, e.g., a property certificate about B_{TPM} signed by a trusted third party that is included within the corresponding authenticator.

[5]This is a feature already available for some recent FPGA devices, e.g., available from Xilinx [25].

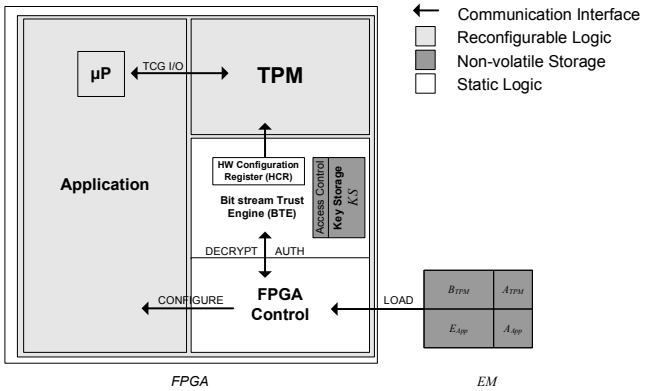

Figure 1: Architecture of a Trusted FPGA.

The *Bit stream Trust Engine* (BTE) provides means to decrypt and verify the authenticity and integrity of bit streams using the DECRYPT and AUTH interfaces.[6] Furthermore, the BTE includes a protected and non-volatile key storage (KS) to store the keys for bit stream decryption and authentication. Finally, the BTE provides a volatile memory location called Hardware Configuration Registers (HCR) to store the configuration information of loaded bit streams. These registers are used later on by the TPM to set up its internal Platform Configuration Registers (PCR).

In the following we define two stages in our protocol, the *setup* and the *operational* phase.

4.3 Setup Phase

To enable an FPGA with TC functionality, a TPM issuer designs a TPM and synthesizes it to a (partial) bit stream B_{TPM} for use on an FPGA. Furthermore, we assume an application designer to provide a TC-enabled FPGA application delivered as partial bit stream B_{App} which can interact with the TPM architecture using a well-defined interface. Of course, particularly when using an *open* TPM implementation, it is possible that both components are developed by a single party, e.g., by the system developer itself.

1. The system developer verifies the authenticity of B_{TPM} and B_{App}, encrypts B_{App} to E_{App} and then creates bit stream authenticators A_{TPM} and A_{App} using the keys $k_{\mathsf{Auth},TPM}$ and $k_{\mathsf{Auth},App}$, respectively.[7]

2. The TPM bit stream B_{TPM}, its authenticator A_{TPM}, the encrypted application bit stream E_{App}, and its authenticator A_{App} are stored in the external memory EM.

3. The system developer writes the appropriate authentication keys $k_{\mathsf{Auth},TPM}$ and $k_{\mathsf{Auth},App}$ (and the encryption key $k_{\mathsf{Enc},App}$) to the key store KS of the BTE.

4.4 Operational Phase

Remember that on each power-up the FPGA needs to reload its hardware configuration from the external memory EM. Hence, for loading a TC-enabled application, the following steps need to be accomplished:

1. On device startup the FPGA controller reads the TPM bit stream B_{TPM} and the corresponding authentication information A_{TPM} from the external memory EM. BTE verifies the authenticity and integrity of B_{TPM} based on the authenticator A_{TPM} by using $\mathsf{Verify}_{k_{\mathsf{Auth},TPM}}(B_{TPM}, A_{TPM})$.

   After successful verification, BTE computes the configuration value C_{TPM} of the TPM bit stream and writes C_{TPM} into the first Hardware Configuration Register (HCR) before the FPGA's fabric is finally configured with B_{TPM}.

2. The TPM requires exclusive access to a non-volatile memory location to store its sensitive state $\mathcal{T} = (EK, SRK, TD)$ where EK denotes an asymmetric key that uniquely identifies each TPM (Endorsement Key), SRK an asymmetric key used to encrypt all other keys created by the TPM (Storage Root Key) and TPM data TD includes further security-critical non-volatile data of the TPM. This requires an extension of recent SRAM-FPGA devices with on-chip non-volatile storage which is discussed in more detail in Section 5. Furthermore, the access to this storage location is protected by an Access Control Function (ACF) in the static logic which provides access to sensitive data only when a specific bit stream (i.e., the TPM) is loaded. For full flexibility, the ACF implements an interface with which a currently configured bit stream can request a reset (and implicitly, a clear) of the non-volatile memory to reassign the access to the storage for its own exclusive use. The access authorization to the memory for a loaded bit stream X can easily be performed by BTE by checking its C_X stored in the first HCR.

3. After the TPM has been loaded into the fabric, the application bit stream E_{App} and its authenticator A_{App} are read from EM, verified and decrypted in the same way. The BTE stores the configuration value C_{App} of the verified application in the second HCR register. After the application bit stream has been configured in the fabric, the first call of the application to use the TC functionality will initialize the TPM as follows: Based on the content of the HCR (C_{TPM}, C_{App}), the TPM initializes its own PCRs: $PCR_1 \leftarrow \mathsf{Hash}(PCR_0|C_{TPM})$ and $PCR_2 \leftarrow \mathsf{Hash}(PCR_1|C_{App})$ where $\mathsf{Hash}(x)$ denotes the internal hash function of the TPM and PCR_0 is some constant (root) value. In this way the (unique) configurations of all bit streams can be included in the chain of trust.[8]

After loading the hardware configuration of TPM and application into the FPGA, the chain of trust can be extended by the measurements of other specific system components like the operating system and high-level application software. This allows to bind any higher level application (of the IP provider) to the underlying FPGA by binding the application (or its data) using the subset of the PCR registers that contain the corresponding measurements of the underlying FPGA.

[6]Except for authentication, recent FPGAs already provide LOAD and DECRYPT interfaces (cf. Section 5) [2, 25].

[7]If TPM bit stream B_{TPM} is also provided by the system integrator itself, he can choose $k_{\mathsf{Auth},TPM} = k_{\mathsf{Auth},App}$.

[8]This is similar to the initialization of a desktop TPM via CRTM. However, now the PCR includes the hardware measurement results of the TPM itself.

4.5 TPM Updates

The update of the current TPM_1 to another TPM_2 on an FPGA is quite easy when the sensitive state $\mathcal{T}$ does not need to be migrated. The TPM_2 needs to be loaded and will obviously not be able to access the ACF controlled memory containing $\mathcal{T}$ of TPM_1 (since TPM_2 cannot provide C_{TPM_1}). Hence, TPM_2 reassigns the ACF to be able to create and store its own $\mathcal{T}$. With the reset of the ACF, the previous $\mathcal{T}$ in the non-volatile memory is cleared[9] so that no confidential information of TPM_1 will be accessible for TPM_2.

However, for migrating $\mathcal{T}$ from TPM_1 to TPM_2 without loss of $\mathcal{T}$, we propose to extend existing TPM implementations by an migration function[10] $\mathsf{Migrate}(\mathsf{UA}, C_{TPM_2})$ where UA is an *Update Authenticator* and C_{TPM_2} a unique reference to the corresponding TPM_2. For a TPM update, a system developer (who has set $k_{\mathsf{Auth},TPM}$ for the corresponding FPGA) generates an update authenticator $\mathsf{UA} \leftarrow \mathsf{Sign}_{SK_{UPD}}(C_{TPM_2}, P_{TPM})$ with the following parameters: SK_{UPD} denotes an update signing key where TPM_1 trusts the corresponding update verification key PK_{UPD}, e.g., pre-installed in TPM_1. Thus, TPM_1 knows a set of trusted update authorities (the system developer, etc.) who are allowed to perform the migration of $\mathcal{T}$ for use with TPM_2. P_{TPM} denotes a reference to the class of TPMs that provides a certain (minimum) set of security properties. Note, P_{TPM} can also be replaced by individual update signing keys each representing a single security property.

When the user requests a TPM update (e.g., as a feature of an application), he invokes the migration function of TPM_1 using the parameters UA and C_{TPM_2} received from the corresponding system developer (over an untrusted channel). Then, the migration function $\mathsf{Migrate}(\mathsf{UA}, C_{TPM_2})$ performs the following steps:

1. The migration function of TPM_1 verifies UA using the update verification key PK_{UPD} and checks whether P_{TPM_2} provides the same (minimum) set of security properties as P_{TPM_1}.

2. After successful verification, the migration function of TPM_1 reassigns the ACF (containing $\mathcal{T}$) for use with TPM_2. The ACF needs to grant access to TPM_2 without erasing the non-volatile memory. More precisely, the BTE provides a further interface so that only TPM_1 with access to the ACF memory can associate the memory with C_{TPM_2}. After reassignment of the ACF memory, only the new TPM_2 is able to access $\mathcal{T}$.

After the migration function has terminated, the application (or manually, the user) overwrites TPM_1 stored in the external memory EM with B_{TPM_2} and the corresponding authenticator A_{TPM_2}. Now, the user restarts the FPGA to reload the updated TPM and application (cf. Section 4.4).

[9]To prevent denial-of-service attacks against $\mathcal{T}$, BTE can additionally implement a mechanism such that TPM_1 has to clear its $\mathcal{T}$ before TPM_2 is able to reassigns the ACF for its own $\mathcal{T}$.

[10]Note, our migration functionality does *not* replace TCG mechanisms for migrating internals called `TPM_Migration` respectively `TPM_Maintenance`.

4.6 Discussion and Advantages

Enhancing an FPGA with TC mechanisms in reconfigurable logic can provide the following benefits.

Enhancing Chain of Trust. As mentioned in Section 3.1, recent TPM enabled systems establish the chain of trust by starting from the CRTM, which is currently part of the BIOS. For FPGA hosted TPMs, the BTE can begin with the hardware configuration of the application and even with the TPM itself. Therefore, the chain of trust can include the underlying hardware as well as the TPM hardware configuration, i.e., the chain of trust paradigm can be moved to the hardware level.

Flexible Usage of TPM Functionality. The developer may also utilize the basic functionality of the TPM in his application which can make the development of additional cryptographic units obsolete. This includes the generation of true random numbers, the asymmetric cryptographic engine as well as protected non-volatile memory. Furthermore, a flexible FPGA design allows to use only that TPM functionality which is required for the application.

Flexible Update of TPM Functionality. A TPM implemented in reconfigurable logic of an FPGA can easily be adapted to new requirements or versions. For example, if the default hash function turns out to be not secure enough [7], an FPGA hosted TPM could include a self-modification feature which updates the corresponding hash function component, in particular no new hardware design is needed. Moreover, patches fixing potential implementation errors or changes/updates enhancing interoperability could be applied quickly and very easily. The current TCG specification defines the binding/sealing functionality based on binary hashes and hence any changes to the chain of trust can render sealed data inaccessible, even when keeping the same level of security. This is a general limitation of the TCG solution and holds for our chain of trust model as well. However, in [15] the authors propose the concept of *property-based sealing* that provide a mapping between security properties provided by a platform configuration and its binary measurements (hash values) making updates very efficient, since as long as properties are preserved, changes during update to binary measurements have no impact on sealed data. In this context, the authors also propose [15] to use a new TPM command called `TPM_UpdateSeal` that allows a TPM to verify a certificate issued (by a trusted third party) on a new configuration, and hence reseal the data under the new configuration. Note that extending the TPM command set by this command can be done very efficiently in particular in our reconfigurable solution.

Improved Communication Security. The integration of CPU, ROM, RAM and TPM into a single chip enhances protection of communication links between these security-critical components from being intercepted or manipulated. Having the boot ROM and RAM integrated on the FPGA chip, makes the injection of malicious boot code or RAM manipulations more difficult.

Vendor Independence. Platform owners can select which TPM implementation is operated on their platforms. This allows even the usage of fully vendor independent *open TPM implementations* providing more assurance regarding trapdoors and Trojans. Moreover, since we can easily implement a TPM soft core into hardware, a multitude of vendors can offer a variety of TPM implementations. Thus, users are not

only restricted to a few TPM ASIC manufacturers as today, they even can implement their own TPM instances[11] and thus do not have to trust any external manufacturer.

5. IMPLEMENTATION ASPECTS

Incorporating TC functionality into FPGAs for enabling trusted embedded computing platforms requires some modifications to current FPGA architectures. Hence, we will only present implementation proposals to demonstrate the feasibility of TPMs on reconfigurable devices. As a starting point, we take a recent FPGA device with protection mechanisms as a reference, i.e., we assume an SRAM-based FPGA that provides symmetric bit stream decryption, partial hardware configuration and a small amount of non-volatile (key) storage. Such FPGA architecture implement the non-volatile key store in different ways, e.g., Altera's Stratix II devices use an (single-write) anti-fuse technique [2]. On Virtex II, 4 and 5 FPGAs, Xilinx stores the key during power-down using rewritable memory buffered by a battery [25]. The advantage of being able to replace the key comes at the price of an additional external battery element, implying a limited lifetime of the system. Other approaches might use logic with built-in flash memory but this is only available with a less advanced manufacturing technology resulting in smaller devices (cf. Actel IGLOO or ProAsic3 [1]). Instead of using battery-buffered memory locations for preserving the TPM state $\mathcal{T}$, it is possible to integrate a non-volatile memory directly on the FPGA chip [9]. Alternatively, newer FPGA devices like the Xilinx Spartan 3AN [25] already offer SRAM-based circuits combined with a non-volatile Flash memory layer in the same package. The Flash memory in those devices allows for storing up to 11 MBit of user-defined data, perfectly suited for storing $\mathcal{T}$ (with an additional ACF implementation).

To realize the BTE and bit stream authentication of B_{App} and B_{TPM} we require minor modifications to the existing (AES) decryption cores. For integrity verification and authentication of bit streams in the BTE, one option is to use a Message Authentication Code (MAC) which ideally uses the same cryptographic engine used for bit stream decryption (cf. block ciphers usage for MAC [20]).

Table 1: Estimated number of Logical Elements (LE) and RAM bits for the TPM functionality

TPM Function	Reference Design	LE	RAM
RSA-2048 Core	ARSA-128 [11]	900	13k
SHA-1 Core	Helion SHA-1 [13]	1000	0
TRNG	Fabric TRNG [14]	< 50	256
TPM Firmware	Internal RAM	-	16k
Volatile Memory	Internal RAM	-	16k
Controller	Picoblaze [25]	192	18k
Total (+overhead)	-	3000	75k

For all other (reconfigurable) cryptographic components of the TPM itself, a multitude of proposals for efficient implementation are available in literature. Table 1 shows estimates on a TPM realized in the fabric of an FPGA. Please note that the implementations have been selected to be area-optimal and have been converted to a metric based on Logical Elements[12] (LE) to estimate the resource consumption of a TPM implemented in the reconfigurable logic of an FPGA. Translated to a low-cost Xilinx Spartan-3AN XC3S1400AN with a total system complexity of 25,344 LEs, the TC enhancements will take about 3,000 LEs and require about 12% of the device capacity. Hence, we can conclude that a TPM implementation will obviously be efficient with recent devices.

[11] These own implementations can be certified for TCG compliance by a trustworthy authority.

[12] An LE consists of a single 4-input LUT connected to a single-bit flip-flop

6. REFERENCES

[1] Actel Corporation. IGLOO and ProASIC Flash-based FPGAs. www.actel.com/products/.

[2] Altera. Stratix II and Stratix II GX FPGAs. www.altera.com/products/devices/.

[3] Alves, T., and Felton, D. TrustZone: Integrated Hardware and Software Security. In *ARM Inc. White Paper* (2004).

[4] Bellare, M., and Namprempre, C. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In *ASIACRYPT* (2000).

[5] Bo, Z., Huanguo, Z., and Rui, H. A New Approach of TPM Construction Based on J2810. In *WUJNS* (2007).

[6] Brizek, J., Khan, M., Seifert, J.-P., and Wheeler, D. A Platform-level Trust-Architecture for Hand-held Devices. In *CRASH* (2005).

[7] Cannière, C. D., and Rechberger, C. Finding SHA-1 characteristics. In *ASIACRYPT* (2006).

[8] Common Criteria Project. Common Criteria and Common Evaluation Methodology Version 3.1. www.commoncriteriaportal.org.

[9] De Vries, A., and Ma, Y. A Logical Approach to NVM Integration in SOC Design. *EDN 2* (2007).

[10] Drimer, S. Authentication of FPGA bitstreams: Why and how. In *International Workshop on Applied Reconfigurable Computing* (2007).

[11] Fry, J., and Langhammer, M. RSA & Public Key Cryptography in FPGAs. Tech. rep., Altera Corporation, 2005.

[12] Guajardo, J., Kumar, S., Schrijen, G.-J., and Tuyls, P. FPGA intrinsic PUFs and their use for IP protection. In *CHES* (2007).

[13] Helion. SHA-1 Cores. www.heliontech.com.

[14] Kohlbrenner, P., and Gaj, K. An embedded true random number generator for FPGAs. In *FPGA* (2004).

[15] Kühn, U., Kursawe, K., Lucks, S., Sadeghi, A.-R., and Stüble, C. Secure data management in trusted computing. In *CHES* (2005).

[16] Kursawe, K., Schellekens, D., and Preneel, B. Analyzing trusted platform communication. In *CRASH* (2005).

[17] Lai, X., and Massey, J. L. Hash function based on block ciphers. In *EUROCRYPT* (1992).

[18] Lilian Bossuet, Guy Gogniat, W. B. Dynamically configurable Security for SRAM FPGA Bitstreams. *International Journal of Embedded Systems 2*, (2006).

[19] Microsoft Corporation. Bitlocker drive encryption: Technical overview, April 2006. www.microsoft.com/technet/windowsvista/security/bittech.mspx.

[20] National Institute of Standards and Technology (NIST). Recommendation for block cipher modes of operation – the CMAC mode for authentication. NIST Special Publication SP 800-38B, 2005.

[21] Sailer, R., Zhang, X., Jaeger, T., and van Doorn, L. Design and Implementation of a TCG-based Integrity Measurement Architecture. In *USENIX Security Symposium* (2004).

[22] Samyde, D., Skorobogatov, S., Anderson, R., and Quisquater, J.-J. On a new way to read data from memory. *Proceedings of IEEE Security in Storage Workshop* (2002).

[23] Simpson, E., and Schaumont, P. Offline hardware/software authentication for reconfigurable platforms. In *CHES* (2006).

[24] The Trusted Computing Group (TCG). www.trustedcomputinggroup.org.

[25] Xilinx. Spartan-3 and Virtex FPGAs. www.xilinx.com/products/silicon_solutions/.

[26] Zambreno, J., Honbo, D., Choudhary, A., Simha, R., and Narahar, B. High-performance software protection using reconfigurable architectures. *IEEE 94* (2006).