Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2011, Article ID 820454, 19 pages
doi:10.1155/2011/820454

Review Article

A Security Scheme for Dependable Key Insertion in Mobile Embedded Devices

Alexander Klimm, Benjamin Glas, Matthias Wachs, Sebastian Vogel, Klaus D. Müller-Glaser, and Jürgen Becker

Institute for Information Processing Technology, Karlsruhe Institute of Technology (KIT), 76021 Karlsruhe, Germany

Correspondence should be addressed to Alexander Klimm, klimm@kit.edu

Received 27 August 2010; Revised 5 February 2011; Accepted 10 February 2011

Academic Editor: Michael Hübner

Public Key Cryptography enables entity authentication protocols based on a platform's knowledge of other platforms' public key. This is particularly advantageous for embedded systems, such as FPGA platforms, with limited or none read-protected memory resources. For access control systems, an access token is authenticated by the mobile system. Only the public key of authorized tokens needs to be stored inside the mobile platform. At some point during the platform's lifetime, these might need to be updated in the field due to loss or damage of tokens. This paper proposes a holistic approach for an automotive access control system based on Public Key Cryptography. Next to a FPGA-based hardware architecture, we focus on a secure scheme for key flashing of public keys to highly mobile systems. The main goal of the proposed scheme is the minimization of online dependencies to Trusted Third Parties, Certification Authorities, or the like, to enable key flashing in remote locations with only minor technical infrastructure. Introducing trusted mediator devices, new tokens can be authorized and later their public key can be flashed into a mobile system on demand.

1. Introduction

Embedded systems in various safety critical application domains such as automotive, avionic, and medical care perform more and more complex tasks using distributed systems like networks of electronic control units (ECUs). Introducing Public Key Cryptography (PKC) to embedded systems provides essential benefits for the fabrication of electronic units needing to meet security requirements as well as for the logistics involved. Due to the nature of PKC, the number of keys that need to be stored in the individual platform is minimized. Only the private key of the platform itself needs to be stored secretly inside each entity—in contrast to symmetric crypto systems where a single secret key needs to be stored inside several different entities. In context of PKC, if one entity is compromised, the others remain unaffected.

Besides encrypting or signing of messages, PKC can be employed to control user access to a device via electronic tokens. Examples for this are Remote Keyless Entry (RKE) systems [1] in the automotive domain or Hilti's TPS technology [2]. These systems incorporate contactless electronic tokens that substitute classical mechanical keys. The owner or authorized user identifies himself to the user device (`UD`) by possession of the token. `UD` and token are linked. Only if a linked token is presented to `UD`, it is enabled or access to `UD` is granted. In order to present a token to `UD`, information has to be exchanged between the two. The communication channel is usually assumed to be insecure. To prevent the usage of a device or its accessibility by an unauthorized person, the authentication has to be performed in a secure manner.

Authentication schemes based on Public Key Cryptography such as the Needham-Schroeder protocol [3], Okamoto-Protocol [4], and Schnorr-Protocol [5] provide authentication procedures where no confidential data is transmitted. Secret keys are stored in the tokens only and not in `UD`, thus omitting the need for costly security measures in the `UD`. Only public keys have to be introduced into `UD` (see Section 2), which can usually only be done by the manufacturer (`OEM`) of `UD`. In real-world operation, the introduction of public keys is

done in the field where UD is not necessarily under the control of OEM and a live online connection to OEM may not be possible. PKC is computationally very expensive, especially when aiming for high security levels. Dedicated hardware can provide the necessary speed up of cryptographic operations. With the decreasing cost of FPGAs, these devices are introduced more and more into embedded systems and mass market products. Therefore, hardware accelerators can be made available in these cost sensitive systems by adding cryptographic computation blocks on FPGA.

We propose a system to introduce public keys into FPGA based user devices to pair them with a new token. The proposed key flashing method allows authorization of the flashing process by OEM. Additionally it can be carried out with UD in the field and with no active online connection to OEM while flashing a key into UD. Introduction or flashing of new keys to an embedded device can be seen as a special case of a software update. Latter focuses on protection of the intellectual property, interoperability, correctness, robustness, and security. Recent approaches for the automotive area have been developed, for example, in the german HIS [6, 7] or the EAST-EEA [8] project. A general approach considering security and multiple software providers is given in [9]. Nevertheless, general update approaches are focused on the protection of IP and the provider against unauthorized copying and less on the case that the system has to be especially protected against unwanted updates as in our key flashing scenario.

The remainder of this paper is structured as follows. In Section 2, we present the basic application scenario followed by a short introduction to public key cryptography in Section 3. Section 4 describes a high-speed architecture for cryptographic computations. The requirements for the keyflashing scenario are described in Section 5. Based on this, we propose our flashing concept in Section 6, followed by the according requirements (Section 6.3). Section 7 details the flashing protocol with a live online connection available and Section 8 the protocol with no online dependability. Implementation details of the prototypical flashing framework are given in Section 9. We conclude with a security analysis and an outlook to future work in Sections 10 and 11.

2. Application Scenario: Automotive Access Control Systems

The target application focused on in this work is foremost automotive access control system. They comprise an entity that acts as the verifier (an ECU within the car) and an entity that acts as a prover (the traditional car key). Traditionally, a standard car key serves the sole purpose of identifying the current owner of the key as the authenticated user of the car (*authentication by ownership*). This also holds true for electronic car keys. As depicted in Figure 1, access to the car is granted by unlocking the doors only if the correct car key (prover) is presented to the car. The same procedure can be employed to disable or enable the immobilizer of the car, allowing the car's engine to start or not.

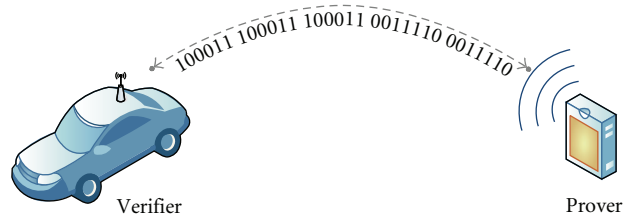

Figure 1: Access control: authentication scheme.

The automotive domain implies a very specific set of requirements. The industry is extremely cost driven, thus creating the need for very small hardware footprints. To comply with limited cost, OEMs tend to resort to cheap off-the-shelf components instead of specialized ASICs or complete systems-on-chip (SoC). Additionally a car's life cycle is about 10–15 years. Within this time span, all systems should work flawlessly.

Access control systems are a natural point of attack. Therefore, they need to offer very good security. To provide this, electronic car keys incorporate some kind of cryptographic algorithm. Raising security levels in this context can be achieved by adaption of the authentication protocol being used, enlarging key lengths, or substituting cryptographic primitives. All these measures tend to increase computation times. But all underlying computations and algorithms incorporated in access systems shall not be noticeable to the user of a car for best usability. Keeping the underlying hardware platform adaptable to varying interfaces and functionalities, it enables for integration of the same hardware components into a wide range of car keys for a multitude of different car models. With FPGAs dropping in cost over the last years, they also have been introduced more and more in cost driven industries such as the automotive domain. These devices are already being used in infotainment and multimedia devices. In addition to that, they can be used to provide dedicated hardware modules to accelerate cryptographic computations within user authentication in these systems. By using FPGA platforms for access control systems, they are adaptable over the lifetime of a car and offer some flexibility regarding changes in protocol and processing units.

In summary, we will regard the following application scenario: an access control system is applied to a mobile user device (UD); in our case, a vehicle is depicted in Figure 2. Through the access control system, the use of the UD can be restricted by allowing only the owner or authorized user access to the device. A transponder (TRK) serves as an electronic version of a mechanical key. TRK communicates to UD over a wireless communication channel. The user device accepts a limited number of transponders. If one of these is presented to the user device, it authenticates the transponder and the device is unlocked, thus granting access. Anyone possessing a valid TRK is considered an authorized user (OWN). This setup forms an authentication chain for usage of UD. An authorized user is authenticated through the possession of a valid TRK paired to the the UD. TRK in turn is authenticated by UD.

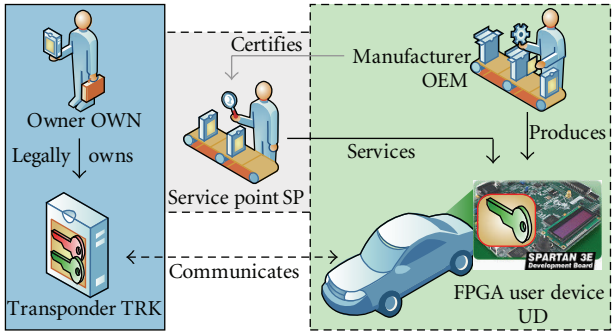

Figure 2: Entities and application scenario.

In automotive systems, authentication of a TRK can be achieved through a number of methods: rolling codes, symmetric codes, one-way-functions, or asymmetric codes. As analyzed in [10], there is a major disadvantage in using rolling codes and symmetric codes since secrets have to be stored within TRK as well as in UD, demanding for highly secure key management. One-way-functions such as cryptographic hash functions can circumvent this to some extent but demand for a substantial amount of secure storage. The most wide spread method for authentication in mobile devices is probably the usage of rolling codes (such as the KeeLoq [11] algorithm) due to easy implementation, followed by one-way-functions.

Asymmetric codes are very computationally expensive, although they provide extremely high security. With the advent of more and more computational power in embedded systems [12, 13], introducing such codes for user authentication is now feasible.

OEM is the manufacturer of UD. Due to the mobility of UD, it may be nowhere near OEM. Therefore, a service infrastructure has been established by OEM to repair, service, or replace a UD in the field. This infrastructure consists of a number of service points SP that are OEM certified. In the depicted example from the automotive domain, this would be a dealer or a car repair shop. SP is enabled by the OEM to carry out certain work on UD and acts in a way as a substitute for the OEM in the field.

In case of loss of a transponder, it is desirable to replace it, particularly if the user device itself is very costly or actually irreplaceable. Since the user device is mobile, linking a new transponder to a UD usually needs to be done in the field. This might include very secluded areas with minor to none communication infrastructure.

3. Basic PKC Functionalities

In 1976, Diffie and Hellman introduced the first PKC crypto system [14] for data encryption and confidential data transfer. Two different keys are used, one public (PK) and the other secret (SK). SK and corresponding PK are a fixed and unique keypair. It must be computational infeasible to deduce the secret key (SK) from the public key. With PK, a message M_p can be encrypted into M_c but not decrypted with the same key. This can only be done with knowledge of SK. If an entity Alice wants to transmit a message M_{Alice} to an entity Bob, it encrypts it with Bobs public key PK_{Bob}. Only Bob can retrieve the plain text from the encrypted message, by applying the appropriate decryption algorithm using his own secret key SK_{Bob}.

PKC can also be used to digitally sign a message. For this, a signature scheme is used that is usually different from the encryption scheme. When signing a message, the secret key is used and the signature can be verified using the according public key. In other words, if Bob wants to sign a message, he uses his own private key that is unique to him and solely known to himself. This key is used to sign a cryptographic hash value of the message M_{Bob}. The resulting value $\{\text{HASH}(M_{\text{Bob}})\}_{\text{sig}}$ is transmitted together with M_{Bob}. A receiver can validate the signature by using Bob's public key to retrieve $\text{HASH}(M_{\text{Bob}})$. From M_{Bob}, the receiver can compute the according hash value and compare it with the retrieved value. If both match, the signature has been validated. Since in the case of signature schemes the public key is often called verification key and the secret key is called signing key, we denote them accordingly VK and SK in the following.

4. Cryptographic Processing Entity

Computational efforts of cryptographic functionalities for PKC are very high and time consuming if carried out on today's standard platforms (i.e., microcontrollers) for embedded applications. Integrating security algorithms into FPGA platforms can provide high speed up of demanding PKC crypto systems such as *hyper elliptic curve cryptography* (HECC). By adding dedicated hardware modules for certain parts of a crypto algorithm, a substantial reduction of computation time can be achieved [15, 16].

In [16], an FPGA platform has been introduced which allows extremely fast authentication as proven by an experimental setup with two of these platforms. For this demonstrator, both platforms have been implemented on a Xilinx Spartan-3 XC3S2000 FPGA at 33 MHz. The communication channel in the setup is a wireless automotive transmitter [17] as is currently used in keyless go systems and is clocked with 412,5 kHz. The transceiver is connected to the FPGA system over SPI. Authentication of TRK via the Schnorr-protocol [5] in this setup lasts 120 ms including communication times over the wireless channel. To enable for even faster computation, we have developed a new, lean cryptographic core for Xilinx FPGA. It enables to carry out aforementioned mutual authentication within 82 ms.

Both platforms carry out calculations for public key cryptography based on hyper elliptic curves (HECC). They offer a higher security level than RSA while relying on relatively small key sizes of around 160 bit [18]. A detailed view on HECC and its underlying mathematics can be found in [19].

As shown in Figure 3, the automotive electronic control unit (ECU) comprises a MicroBlaze processor that handles arbitrary tasks necessary for running the car and is equipped with an appropriate interface such as CAN

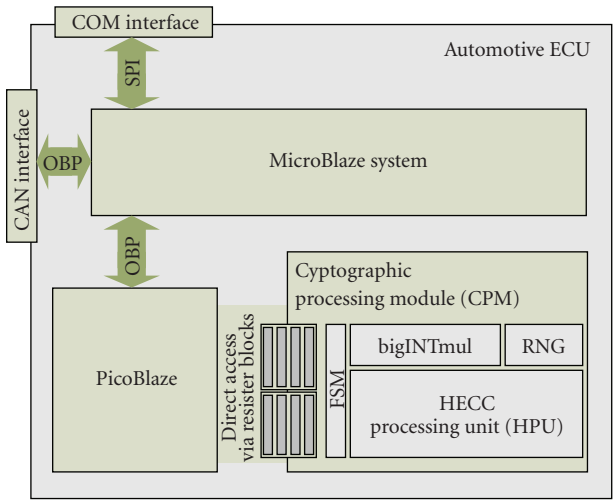

Figure 3: Overview of the system design.

to communicate with other ECUs residing in the vehicle. Additionally, a coprocessing unit composes of a PicoBlaze processor [20] and a *Cryptographic Processing Module* (CPM) is included. All cryptographic computations are done within this coprocessor. When no extensive tasks need to be run and only cryptographic functionality is needed, the coprocessor can also be run without the MicroBlaze. In this case, the PicoBlaze Controller is interfaced directly to the communication interface. This setup is very suitable for implementing a car key (`TRK`).

For HECC, three types of operations are essential (P_i denoting a point on a hyperelliptic curve and k, y, a, e, r are denoting integer values):

(i) calculation on a hyperelliptic curve ($k \cdot P$ and $P_1 + P_2$),

(ii) integer calculation with large operands ($y = a \cdot e + r$),

(iii) data exchange to/from the cryptographic unit.

Each arithmetic operation is assigned to a specialized hardware module to enable fast computation. At the same time, all cryptographic functionality is bound strictly to CPM, thus keeping all sensitive data on chip.

4.1. Cryptographic Processing Module. The *Cryptographic Processing Module* (CPM) is designed to efficiently compute $k \cdot P$ on a hyperelliptic curve, as well as integer multiplication with large operands. As depicted in Figure 4, the proposed architecture encompasses dedicated modules (HPU, bigINTmul) for these two operations and an additional module for generating random numbers (RNG).

A small finite state machine (FSM) is implemented for control flow of the calculations and to provide data exchange over the PicoBlaze processor. It controls all modules within the CPM directly. All arithmetic modules can work fully in parallel, allowing for concurrent operations within the protocol if necessary. A set of registers is provided for data exchange that can be accessed directly by PicoBlaze. Register e acts as input whereas y_i and the address space X of the CPM's internal memory *DataMem* are doubling as output registers. Some additional internal registers store cryptographic key material (Reg a_i) or random numbers (Reg r_i) and cannot be accessed from outside the CPM.

4.1.1. bigINTmul. A dedicated integer unit performing $y = a \cdot b + c$ on large operands is included in the CPM. Input to the multiplier are two operands a and b. The result $p = a \cdot b$ and operand c are then input to an adder stage calculating $y = p + c$. A sequential multiplier as depicted in Figure 5 is provided to execute a naive *shift&add* algorithm. p is accumulated by bitwise shifting of the bigger operand b, evaluating the least significant bit and adding a to the intermediate result if $b_0 = 1$, and then shifting p. If $b_0 = 0$, p is shifted without adding a.

In our use case, the two operands do not have the same bitlength since one input is the platforms secret key a_i and the other input is the challenge e.

4.1.2. HECC Processing Unit. The *HECC Processing Unit* (HPU) acts as a stand alone module for scalar multiplication $X_i = r_i \cdot P_i$ on a hyper elliptic curve. It comprises a dedicated arithmetic logical unit *dALU* for finite field arithmetic (*GF-operations*), internal memory *DataMem* for storage of intermediate values such as curve parameters and points P_i, and a control entity *HECC CTRL* connected to a program memory *pMEM*.

HECC CTRL in conjunction with *pMEM* implements the control flow of a dedicated algorithm for a scalar multiplication as a fixed sequence of GF-operations. The control flow is strongly optimized to execute a scalar multiplication in wMOF [21, 22]. To execute it, a highly specialized instruction set is implemented. An example of such an instruction is *shiftl_2* which shifts the content of the accumulator 2 bits to the left, an operation essential in wMOF [22]. The full instruction sequence of wMOF is stored in *pMEM*.

HPU is laid out as accumulator machine with harvard architecture. This enables to implement different data widths for *DataMem* and *pMEM* individually. This is particularly advantageous as we operate on galois fields GF(2^n), n being a big prime number, resulting in data words of n-bit length being stored in *DataMem*, while *pMEM* only stores minimal instruction codes.

Input to the HECC processing unit is a scalar r_i (bitlength of $r_i \leq l$) that is written into a dedicated register *rREG* of length l. Point P_i is a predetermined common point on a hyperelliptic curve and acts as a constant input decided upon during design time. Therefore, it is permanently stored in *DataMem* together with other curve parameters.

After storing r_1, HPU is triggered via the signal `control` to start a scalar multiplication. As soon as the result X representing the *commitment* is available, this is signaled by HPU over `trigger`. All r_i are random numbers generated by RNG. Therefore r_1 is loaded from the RNG module into HPU and $X = r_1 \cdot P_1$ is computed. Simultaneously, RNG generates a new random number r_2. After HPU signals the end of the current operation, r_2 can be loaded into *rREG* and HPU can calculate a new result $X = r_2 P_2$.

The dedicated arithmetic logical unit (*dALU*) can perform $u + v$, $u \cdot v \bmod p$ and u^2 with $u, v \in$ GF(2^n).

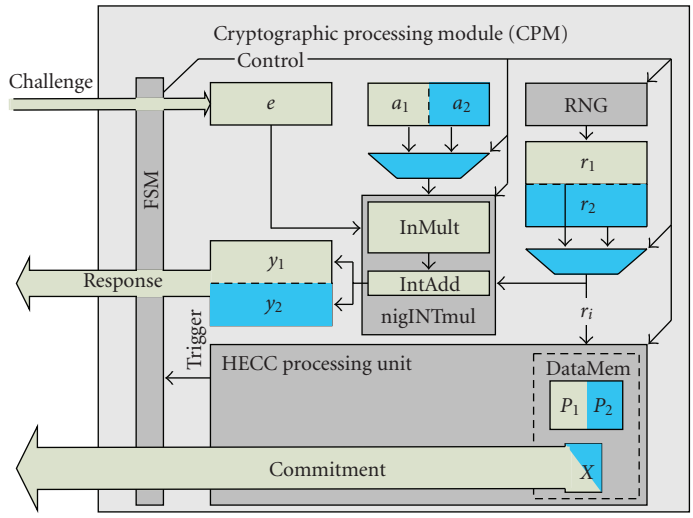

Figure 4: Cryptographic processing module (CPM).

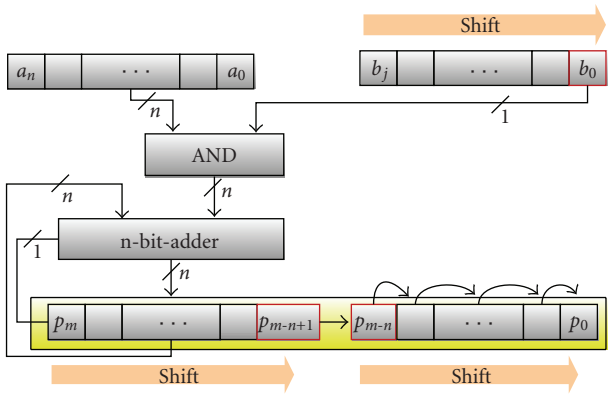

Figure 5: Sequential multiplier.

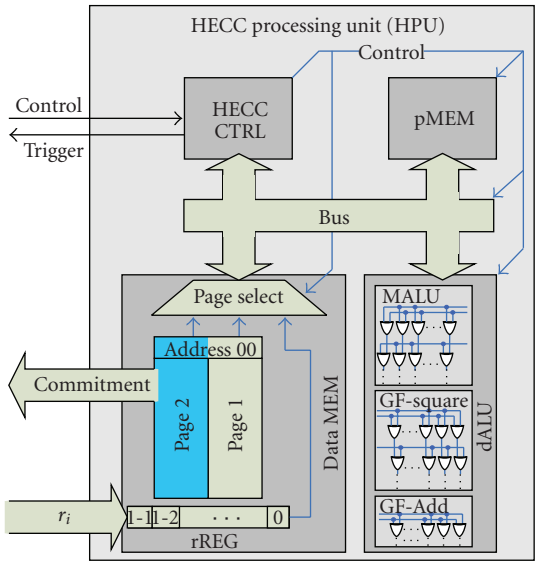

Figure 6: HECC processing unit.

On galois fields of genus 2, addition and subtraction can be implemented very efficiently by a bitwise *XOR* of the two operands u and v. Modular multiplication can be implemented as shown in Algorithm 1. It is performed as a *Shift&Add* algorithm with modular reduction in every step by adding the irreducible polynomial $P(x)$ if the intermediate result $T(x)$ exceeds the field size of GF(2^n). A single GF-multiplication $U \cdot V$ mod P will take n clock cycles since the size of the input word U is n-bit and U is examined bitwise (line 2–7 in Algorithm 1).

To speed up the multiplication a scalable GF-Multiplier (MALU) is integrated into the HPU. Instead of examining U a single bit at a time, a number d of neighboring bits in U are evaluated blockwise and in parallel. Algorithm 2 shows how d adder stages with modular reduction are fed with the operands U and V (line 3–6). This gives d individual intermediate results $T_d(x)$ which are then added and reduced to form the result $T(x)$ (line 6). This addition step is done over d cascading adders, so no additional clock cycle is necessary. The final result of a GF-multiplication is available after $\lfloor (n + d - 1)/d \rfloor$ clock cycles.

Speed up of the multiplication is achieved through the parallelization of additions, but hardware area increases as well. No additional instructions in *pMEM* are necessary, thus keeping this change in algorithm transparent to *HECC CTRL*.

Although u^2 can be calculated by feeding u into both inputs of *MALU*, a dedicated HW squarer requires 25% less computation time [23]. Because *HECADD* and *HECDBL* operations depend heavily on squaring [24], we included a GF-Square module in *dALU*.

4.2. Resource Usage and Timing Results. Table 1 gives a detailed view of the size of our system, and Table 2 shows the processing speed. To the best of our knowledge, no measurements of the execution time of Schnorr- and Okamoto-Authentication protocols based on HECC in the domain of

Requires: $U(x) = u_{n-1}x^{n-1} + \cdots + u_1x + u_0 \in \mathrm{GF}(2^n)$,
$V(x) = v_{n-1}x^{n-1} + \cdots + v_1x + u_0 \in \mathrm{GF}(2^n)$,
$P(x) = x^n + p_{n-1}x^{n-1} + \cdots + p_1x + p_0 \in \mathrm{GF}(2^{n+1})$
Ensure: $U(x)V(x) \bmod P(x)$
(1) $T(x) = t_nx^n + \cdots + t_1x + 0 \leftarrow 0$
(2) **for** $i = n - 1$ **to** 0 **do**
(3) $\quad m \leftarrow u_i \cdot V(x)$
(4) $\quad r \leftarrow t_n \cdot P(x)$
(5) $\quad T(x) \leftarrow (T(x) + m - r)x$
(6) $\quad i \leftarrow i - 1$
(7) **end for**
(8) **return** $T(x)/x$

Algorithm 1: GF multiplication of $U \cdot V \bmod P$ by adding n times.

Requires: $U(x) = u_{n-1}x^{n-1} + \cdots + u_1x + u_0 \in \mathrm{GF}(2^n)$,
$V(x) = v_{n-1}x^{n-1} + \cdots + v_1x + u_0 \in \mathrm{GF}(2^n)$,
$P(x) = x^n + p_{n-1}x^{n-1} + \cdots + p_1x + p_0 \in \mathrm{GF}(2^{n+1})$
Ensure: $U(x)V(x) \bmod P(x)$
(1) $T(x) = t_nx^n + \cdots + t_1x + 0 \leftarrow 0$
(2) **for** $i = n - 1$ **to** ≤ 0 **do**
(3) $\quad$ **for** $j = i$ **to** $i - d + 1$ **do**
(4) $\quad\quad m_j \leftarrow u_j \cdot V(x)$
(5) $\quad\quad r_j \leftarrow t_n \cdot P(x)$
(6) $\quad\quad T(x) \leftarrow (T(x) + m_j - r_j)x$
(7) $\quad$ **end for**
(8) $\quad i \leftarrow i - d$
(9) **end for**
(10) **returen** $T(x)/(x^{d-\lfloor (n+d-1) \bmod d \rfloor})$

Algorithm 2: GF multiplication of $U \cdot V \bmod P$ by adding $\lfloor (n + d - 1)/d \rfloor$ times.

embedded systems has been published. Because of this, we use the execution time of one scalar multiplication $k \cdot P$ as a benchmark in Table 3 to give a fair comparison of our architecture to other implementations.

As shown in Table 1, the deviation in size of the platform synthesized for executing a single scalar multiplication and the platform synthesized for execution two of this operation is less than 10%. The increase lies in the *Cryptographic Processing Module* (CPM) due to the increase in memory needed. An additional secret key a_2, as well as an additional random number r_2, needs to be stored, thus adding two registers. Also a 2nd memory page is needed, reflecting directly in the doubled number of BRAMs. These in turn require some additional slices for glue logic and enlarged multiplexers.

In a prototypical setup both the Schnorr- and Okamoto authentication protocols have been implemented. Table 2 shows that for both variants our architecture easily beats the real-time constraint of 180 ms, which is the average human reaction time [25].

When comparing our architecture with others (see Table 3), we compute a single $k \cdot P$ in roughly 8 ms. This is more than twice as fast as the platform no. 3 and no. 4. It also outperforms platform no. 1 which is one of the first full hardware implementations of a HECC scalar multiplication.

5. Pairing of Verifier/Prover Devices

With introduction of public-key-cryptography to automotive access control systems is advantageous for logistics. Less secret key material has to be handled. When integrating an authentication protocol such as Schnorr [5], no secret key resides in a vehicle (UD)—it only needs to be stored in the vehicle's transponder key token (TRK). Pairing of the two is done by storing a transponders public key in UD. In this paper, we mainly focus on this key flashing procedure for automotive entities and especially introduction of key material into UD during it's lifetime in the field.

Every UD has a number of public keys of transponders securely stored, thus establishing a "guest list" of legal TRKs. During production, at least two initial public keys of TRKs are written to the user device. This ensures that upon loss of one of the transponders, the remaining can be used to authenticate the owner. This initial operation certainly has to be secured against attacks to ensure that the "guest list" is not altered maliciously, otherwise illegal access to a UD might be granted.

As mentioned above, it is necessary to pair TRKs with a UD. This is achieved by flashing the public key of TRK into the user device, where the key is stored securely and is protected against unauthorized alteration. A number of initial TRKs are

Table 1: Resource usage.

		Okamoto				Schnorr			
		Slices	Slice FlipFlops	LUTs	BRAMs	Slices	Slice FlipFlops	LUTs	BRAMs
	Complete system	2651	2263	4323	8	2439	2026	3799	5
I	*PicoBlaze*	389	286	714	1	389	286	714	1
II	*CPM*	2275	1977	3585	0	2039	1740	3055	0
II-a	BigINTmul	265	392	201	0	265	392	201	0
II-b	HECC processing	1684	1082	2855	7	1682	1081	2759	4
II-b-i	*HECCCTRL*	61	32	108	0	61	32	108	0
II-b-ii	*dALU*	953	658	1611	0	953	658	1611	0
II-b-iii	*ProgramMem*	50	70	93	1	50	70	93	1
II-b-iv	*dbus*	98	0	162	0	98	0	162	0
II-b-v	*DataMem(singlepage)*	453	324	838	6	—	—	—	—
II-b-vi	*DataMem(doublepage)*	—	—	—	—	452	324	657	3

Table 2: Performance speed.

Operation	[μs]@50 MHz	
	Schnorr	Okamoto
Scalar multiplication $k \cdot P$		8 069
Complete protocol	81 621	132 858

paired during production in a secured environment by the OEM. Today pairing TRK to UD is done by introducing some kind of key material into UD, thus authorizing this token to use UD. Today's procedures either demand a live online connection to OEM or accept new TRKs if a "master" token (MTRK) is presented to the device [28, 29]. This means that such UD specific MTRK have to be stored very securely by SP and OEM has to fully entrust SP to do so. At the same time, there is no way to prevent TRKs to be flashed into a UD. Therefore, they have to be kept in secure, physical storage as well.

When employing asymmetric codes, these drawbacks are inexistent. Pairing procedures in this case depend mostly on a *trusted third party* (TTP) that generates key pairs and distributes them to the different entities. Because not only public key material is transferred but also secret key material this demands for fully encrypted end-to-end communication channels. Traditionally, this is done by establishing a mutual secret key between the two communication partners (i.e., via Diffie-Hellman key exchange [14]) and using symmetric ciphers to encrypt all data over the communication channel.

In our application scenario, we have the following main participants:

(i) a user device UD that may only be accessed or used by an authenticated user,

(ii) a human user OWN and he is authorized to access or use UD if he possesses a legitimate token,

(iii) a transponder key token TRK_{orig} originally linked to UD and a second token TRK_{new} that shall be flashed to UD additionally,

(iv) the manufacturer OEM that produces UD.

UD accepts a number of TRK to identify an authenticated user OWN of the UD. At least, two tokens are linked to a UD by storing the respective public keys VK_{TRK} inside the UD. The OEM is initially the only entity allowed to write public keys into any UD.

Solely, the public keys stored inside the UD shall be used for any authorization check of TRKs. The OEM's public key VK_{OEM} is stored in the UD as well.

OEM, TRK, and UD can communicate over any insecure medium, through defined communication interfaces.

5.1. Goals and Security Requirements. A new transponder TRK_{new} should be linked to UD to substitute an original token TRK_{orig} that has been lost or is defective. In the following, we will call the process of linking TRK_{new} to an UD *flashing*. Introduction of a TRK should be possible anytime in the complete life cycle of the UD. When flashing the UD it is probably nowhere near the OEM's location while introducing a TRK needs to be explicitly authorized by the OEM. Also should any TRK only be flashable into a single UD. Theft or unauthorized use of the UD resulting from improper pairing of a TRK needs to be prohibited. In addition, we demand that online connection of UD and OEM during the pairing procedure must not be imperative.

In summary, the protocol shall allow dependable authorized flashing under minimal assumptions while preventing unauthorized flashing reliably. Therefore, it has to guarantee the following properties, while assuming communication over an unsecured open channel.

(i) Correctness. In absence of an adversary, the protocol has to deliver the desired result, that is, after complete execution of the protocol, the flashing should be accomplished.

(ii) Authentication. The flashing should only be feasible if both OEM and OWN have been authenticated and have authorized the operation.

(iii) No online dependency. The protocol shall not rely on any live online connection to the OEM.

(iv) Secrecy. No confidential data like secret keys should be retrievable by an adversary.

Table 3: Duration of 1 scalar multiplication $k \cdot P$.

No.		Platform	Slices	f [MHz]	t [ms]	$t_{\text{normalized}}(f)$
1	Full Custom HW	[26]	$\gg$ 16000	45	20,2	1,03
2	8051 μC & CoPro	[18]	3781	12	2488	33,89
3	Microblaze & Coproc	[16]	1984	33	26,7	1
4	CrμP	[27]	1854 (+2379 for memory)	31,25	30,3	1,07
4	HECC processor	This work	2439	50	8,0693	0,46

5.2. Adversary Model. We assume an adversary $\mathcal{A}$ that is polynomially bounded in processing power and memory. $\mathcal{A}$ has access to all inter device communications, meaning he can eavesdrop, delete, delay, alter, replay, or insert any messages. We assume further that the adversary is attacking on software level without tampering with the participating devices. Without choosing particular instances of the cryptographic primitives, we assume that the signature scheme used is secure against existential forgery of signatures and regard the cryptographic hash function used as a random oracle.

6. Key Flashing Definitions and Requirements

The objective of the proposed key flashing protocol is the introduction of a public key $\mathtt{VK}_{\mathtt{TRK}}$ into UD. Main focus of it is the security aspect of the protocol itself, while it shall be usable under real world constraints as well. The protocol shall ensure the legitimacy of all entities involved as well as the security of the protocol itself to prevent misuse by a malicious attacker. Only after a correct, complete and successful flashing procedure, a new public key may be accepted and stored inside UD. If any error occurs during the flashing procedure, all previous steps in the protocol have to be revoked. All data resulting from these steps shall carry no information that can be exploited by an attacker.

Since TRKs gain their relevance only after successfully linking them to UD, they shall have no utility value before a successful key flashing procedure. This enables holding numerous TRKs in stock without an inherent need to restrict access to unflashed TRKs.

Two basic flashing scenarios are conceivable. One is that TRKs are flashed directly by the OEM, either during production or via an online connection as is addressed in Section 7. The second scenario is the flashing of TRKs through an authorized service point (SP) with no immediate online connection to the OEM (see Section 8).

6.1. Notations. For presentation of the protocols, we abstract from the specific algorithms and use abstract cryptographic primitives instead. Therefore, we introduce some assumptions, definitions, and notations.

Let H be a collision resistant cryptographic hash function of length k that maps any input of arbitrary bit length to an output of fixed bit length k. Application of H can be seen as taking a fingerprint of the input and is often used by signature systems. Nevertheless, our notion of a signature system given in Definition 1, abstracts from any implicitly used hash function.

Definition 1 (signature system). Let Σ_1, Σ_2 be finite alphabets. A *signature system SigSys* is a 7-tuple

$$SigSys = \left(\mathcal{M}, \mathcal{S}, \mathcal{SK}, \mathcal{VK}, f, \widetilde{S}, \widetilde{V}\right) \tag{1}$$

with

(1) $\mathcal{M}$ a nonempty set of messages $\emptyset \neq \mathcal{M} \subseteq \Sigma_1^*$ of arbitrary length over alphabet Σ_1,

(2) $\mathcal{S}$ a nonempty set of signatures $\mathcal{S} \subseteq \Sigma_2^*$,

(3) $\mathcal{SK}$ a nonempty set of signature keys,

(4) $\mathcal{VK}$ a nonempty set of verification keys,

(5) f a bijective function $f : \mathcal{SK} \to \mathcal{VK}$, mapping each signature key $\mathrm{SK} \in \mathcal{SK}$ on the respective verification key $f(\mathrm{SK}) = \mathrm{VK} \in \mathcal{VK}$, and we define a set $\mathcal{K} \subset \mathcal{SK} \times \mathcal{VK}$ of key pairs by $\mathcal{K} = \{(\mathrm{SK}, \mathrm{VK}) \in \mathcal{SK} \times \mathcal{VK} \mid f(\mathrm{SK}) = \mathrm{VK}\}$,

(6) $\widetilde{S} : \mathcal{M} \times \mathcal{SK} \to \mathcal{S}$ a signature function,

(7) $\widetilde{V} : \mathcal{M} \times \mathcal{S} \times \mathcal{VK} \to \{0, 1\}$ a verification function with the property, that for $\mathrm{SK} \in \mathcal{SK}, \mathrm{VK} \in \mathcal{VK}$, $M, M' \in \mathcal{M}$ holds:

$$\widetilde{V}\left(M, \widetilde{S}(M', \mathrm{SK}), \mathrm{VK}\right) = 1 \iff f(\mathrm{SK}) = \mathrm{VK},\ M' = M. \tag{2}$$

To ease readability and presentation, we introduce a shortened notation. We define the signed message

$$\mathtt{Sig}_{\mathrm{SK}}(M) = \left(M, \widetilde{S}(M, \mathrm{SK})\right) \tag{3}$$

as the message M together with the respective signature. Let $\mathcal{SM}$ be the set of all signed messages. In the following, we use an extended signature function

$$\begin{aligned} S :& \mathcal{M} \times \mathcal{SK} \longrightarrow \mathcal{SM}, \\ & (M, \mathrm{SK}) \longmapsto \mathtt{Sig}_{\mathrm{SK}}(M), \end{aligned} \tag{4}$$

and the respective extended verification function

$$\begin{aligned} V :& \mathcal{SM} \times \mathcal{VK} \longrightarrow \{0, 1\}, \\ & (\mathtt{Sig}_{\mathrm{SK}}(M), \mathrm{VK}) \longmapsto \widetilde{V}\left(M, \widetilde{S}(M, \mathrm{SK}), \mathrm{VK}\right). \end{aligned} \tag{5}$$

For a tuple (M, S'), where either the signature or the message is altered, we define

$$V((M, S'), \mathrm{VK}) = \widetilde{V}(M, S', \mathrm{VK}). \tag{6}$$

Furthermore, the tuple $(\mathrm{SK}_X, \mathrm{VK}_X) \in \mathcal{K}$ denotes the key pair of entity X.

6.2. Entities. In addition to the entities introduced in Section 5 (UD, OWN, TRK, and OEM), we use three additional participants, namely the transponder manufacturer TRKM, a service point SP and an employee SPE of this service point conducting the flashing procedure. In the following, an overview of the entities involved as well as their required properties and abilities is given.

6.2.1. OEM: Manufacturer. The OEM manufactures the UD and delivers it to OWN. OWN issued the corresponding TRKs linked to the UD. All UDs are obviously known to the OEM. Furthermore, TRKM and all SP and the respective public verification keys are known to the OEM. We regard the entity OEM as a trusted central server with database functionality. OEM can store data, sign data with $\mathtt{SK}_{\mathtt{OEM}}$, and send data. It possesses all cryptographic abilities for PKC based authentication schemes and can thereby authenticate communication partners.

6.2.2. TRK: Transponder. TRK possesses a keypair ($\mathtt{VK}_{\mathtt{TRK}}$, $\mathtt{SK}_{\mathtt{TRK}}$) for PKC functionality. It is generated inside TRK to ensure that the secret key $\mathtt{SK}_{\mathtt{TRK}}$ is known solely to TRK. Read access to $\mathtt{VK}_{\mathtt{TRK}}$ is granted to any entity over a communication interface. As TRKs can be manufactured by a supplier TRKM that has been certified by OEM, the $\mathtt{VK}_{\mathtt{TRK}}$ is signed by TRKM after generation and stored in TRK as a certificate. TRK possesses cryptographic primitives for PKC-based authentication schemes on prover's side and can thereby be authenticated by communication partners.

6.2.3. TRKM: Transponder Manufacturer. TRKM is a supplier for TRKs that has been certified by OEM and manufactures TRKs that fulfill OEM's requirements. Certification of TRKM is bound to the fulfillment of the conditions of manufacturing, defined by OEM and is enforced through appropriate legal contracting. TRKM possesses cryptographic abilities to sign the public key $\mathtt{VK}_{\mathtt{TRK}}$ of TRK. These signed keys act as certificates guaranteeing the origin of the respective TRK as well as the compliance of TRKM to all of OEM's manufacturing policies.

6.2.4. UD: User Device. UD is enabled only when a linked TRK is presented by authenticating the TRK via a PKC authentication scheme. All linked TRKs' public keys $\mathtt{VK}_{\mathtt{TRK}}$ are stored in UD. Additionally, the public key of the OEM $\mathtt{VK}_{\mathtt{OEM}}$ is stored in UD and cannot be erased or altered in any way. UD grants read access to all stored public keys. Write access to the memory location of $\mathtt{VK}_{\mathtt{TRK}}$ is only granted in the context of the proposed key flashing scheme. UD possesses all cryptographic abilities for PKC-based authentication schemes and can thereby authenticate communication partners.

6.2.5. OWN: Legal User. OWN is the legal user of UD and can prove this by possession of a linked $\mathtt{TRK}_{\mathrm{orig}}$.

6.2.6. SP: Service Point. SP is a service point in the field such as a wholesaler or workshop, certified by the OEM. For the protocol, SP is considered to be a computer terminal at the respective institution. The terminal and access to it is secured by appropriate means as in standard PC practice. SP can communicate to the OEM as well as to UD. In addition, it is able to read the $\mathtt{VK}_{\mathtt{TRK}}$ of any TRK.

Furthermore, SP constitutes a trusted platform meaning that it always behaves in the expected manner for the flashing procedure and accommodates a trusted module responsible for:

(i) storage of OEM-authorized keymaterial of TRKs,

(ii) key management of TRK keys,

(iii) secure counter.

SP possesses cryptographic primitives for PKC-based authentication schemes on prover's and verifier's side and can thereby be authenticated by communication partners, while it can also actively authenticate communication partners.

6.2.7. SPE: Employee of Service Point. SPE is a physical person that is operating SP and has to be authenticated prior to a flashing procedure to prevent misuse of the system. At the same time, SPE is regarded as a potential attacker of the flashing operation so that the protocol has to have a certain robustness against a compromised SPE. Access control of SPE to SP is enforced via password or similar. SPE is responsible for the system setup for the flashing application consisting of establishing the communication links of UD, SP, TRK, and OEM if needed.

UD, $\mathtt{TRK}_{\mathrm{new}}$, and SP are under control of SPE, and the communication links to UD, $\mathtt{TRK}_{\mathrm{orig}}$, $\mathtt{TRK}_{\mathrm{new}}$, SP, and OEM can be eavesdropped, but the trusted module cannot be penetrated.

6.3. Flashing Policies. In order to meet the goals defined in Section 5, some additional requirements must be met as follows:

(1) legitimation of the flashing procedure by OWN,

(2) legitimation of $\mathtt{TRK}_{\mathrm{new}}$ by a certified TRKM,

(3) only an OEM-authorized TRK may be flashed,

(4) any single TRK may only be flashed into a single UD.

Legitimation of a flashing procedure is achieved if the legal owner of OWN has commissioned and approved a flashing procedure. Legal ownership of a UD is proven by OWN through possession of a valid TRK that is already activated in UD. If no such TRK is available, legal documents proving the ownership have to be presented to OEM. Such a document could be a deed of ownership, for example. If OWN cannot prove his legal ownership, it is mandatory to prohibit flashing in order to prevent usage of an illegally acquired UD with an unauthorized TRK.

Legitimation of TRK is first achieved by adhering to all manufacturing policies posed by OEM. This is guaranteed by TRKM which in turn certifies the manufactured TRKs by signing their respective public keys $\mathtt{VK}_{\mathtt{TRK}}$. A main requirement for a legitimate TRK is its uniqueness and

accordingly the uniqueness of its cryptographic key material. Otherwise two identical TRKs, linked to two separate UDs would automatically be able to access both UDs. Additionally it has to be ensured and guaranteed that no secret key material of TRK is available to any entity other than to TRK itself.

Authorization of a legitimate TRK to be flashed is handled through OEM. OEM verifies the identity of each individual TRK to be flashed. Identification of a TRK can be achieved directly via checking VK_{TRK}. The identities are then stored by OEM. Only TRKs that have their identities checked and stored this way are considered to be authorized by OEM. Prove of authorization can be given by OEM through a signature $Sig_{OEM}(VK_{TRK})$.

7. Key Flashing Protocol without Mediator

The most direct flashing scenario is depicted in Figure 7, where the key flashing into a UD is done directly through the OEM. This scenario is valid during production of the UD or if the OWN is not able to legitimize the procedure through the possession of a second TRK as is needed in the standard flashing scenario as described later in Section 8. Therefore the legitimation of the flashing procedure is done implicitly through the OEM by checking the legal credentials of OWN (i.e., billing receipts, legal records, etc.). Only if sufficient proof of ownership is presented to the OEM, the flashing procedure is carried out.

The following entities are involved in the key flashing protocol:

(i) manufacturer OEM,

(ii) user device UD,

(iii) transponder TRK_{new}.

As shown in Algorithm 3, the direct flashing has two requirements. It is mandatory that UD has stored an immutable VK_{OEM}. This enables UD to verify the correctness of the OEM's signature later in the flashing protocol. A mandatory requirement for carrying out the flashing procedure is that the OEM has verified that the commissioning of the flashing procedure has been done by the legal owner of UD.

In a first step, OEM reads out the public key of the TRK to be flashed (TRK_{Bob}). The manufacturer TRKM of TRK_{Bob} has certified the key by signing it ($Sig_{SK_{TRKM}}(VK_{TRK_{new}})$). The OEM then checks if TRKM has fulfilled all legal obligations to be considered as a trusted manufacturer. If this is the case, the OEM checks if $VK_{TRK_{new}}$ is already stored in its internal database and if it has been already flashed to a UD or not. Only if $VK_{TRK_{new}}$ is a fresh key, it is stored in the OEM's database and the protocol is continued.

The second step consists of the OEM triggering the start of the flashing procedure. UD authenticates the OEM by means of an appropriate public key authentication protocol, referring to the internally stored VK_{OEM}. The protocol can only be passed successfully by the OEM since knowledge of the signing key SK_{OEM} is mandatory for the entity being authenticated. Subsequently, UD sends its self-signed verification key $Sig_{SK_{UD}}(VK_{UD})$ to the OEM.

The signature is then verified by the OEM in order to ensure that the transmitted verification key VK_{UD} has not been tampered with. Subsequently, the OEM binds $VK_{TRK_{new}}$ to VK_{UD} by composing an adequate data packet and signing it as a whole. This is then transmitted back to UD (Step 4 in Algorithm 3).

UD verifies the correctness of the packet by checking the OEM's signature in Step 5. After that, UD verifies the correct binding of data packet to its own identity by inspecting the VK'_{UD} including the data packet received. Only if the received VK'_{UD} is identical to his own verification key VK_{UD}, UD will accept $VK_{TRK_{new}}$ included in the received data packet as a valid, legitimate, and authorized transponder key to be flashed. UD stores $VK_{TRK_{new}}$ into internal protected memory and sends an acknowledge message back to the OEM. Storing $VK_{TRK_{new}}$ in UD turns TRK_{new} into an activated transponder $VK_{TRK_{orig}}$ linked to UD. OEM logs the successful conclusion of the protocol and annotates the $VK_{TRK_{new}}$ (now $VK_{TRK_{orig}}$) accordingly to exempt it from future flashing attempts.

8. Key Flashing Protocol with Mediator

The procedure as outlined in Section 7 demands a live online connection of UD to OEM. To fully comply with the flashing requirements introduced in Section 6, this online connection shall not be mandatory for the complete protocol. Therefore, an additional entity is introduced, a trusted Service Point SP that substitutes for the OEM in the field for introducing a new VK_{TRK} into a UD when no direct online connection to OEM is possible. The mediator and its properties are detailed in Section 8.1. The flashing procedure including a mediator comprises the following steps (see also Figure 8):

(i) delegation of trust to SP,

(ii) authorization of TRK_{new} by OEM,

(iii) introducing an authorized TRK_{new} into an UD.

The first two steps form an initialization phase to enable an SP to substitute for the OEM while flashing a new TRK into a UD. This two-step initialization will be detailed in Section 8.3. These steps form the first phase of the flashing process and can be done in advance without UD and OWN but need a communication link to OEM.

The last of the three steps, the actual flashing process of a new VK_{TRK} into an UD, do no longer depend on any direct interaction with OEM. Details will be given in Section 8.4.

8.1. Mediator. Because a mediator has to partly replace the OEM during the flashing protocol and UD only allows TRKs to be flashed through a trustworthy source—namely the OEM—the mediator has to be enabled to act as a trustworthy entity [30]. For this, the OEM has to delegate trust to a SP, in order to enable UD to entrust SP enough to accept a flashing request from it. It has to be ensured that the security of the overall flashing protocol is not weakened. Every mediator (SP) is evaluated by the OEM for its trustworthiness. Assessment factors can also include nontechnical aspects such as political and cultural environment, legal issues, or business models.

Requirements
(i) UD has knowledge of VK_{OEM}
(ii) OEM has ensured, that the legal owner OWN of UD has commissioned the flashing procedure
Protocol
Step 1: For a new TRK_{new} to be flashed OEM reads out the respective certified verification key $Sig_{SK_{TRKM}}(VK_{TRK_{new}})$ and verifies that VK_{TRKM} is in the internal database of trusted transponder manufacturers.
Step 2: OEM contacts UD and is authenticated using a PKC authentication protocol. UD sends $Sig_{SK_{UD}}(VK_{UD})$ to OEM.
Step 3: OEM verifies that $V(Sig_{SK_{UD}}(VK_{UD}), VK_{UD}) \stackrel{!}{=} 1$
Step 4: OEM sends $Sig_{SK_{OEM}}(VK_{TRK_{new}}, VK_{UD})$ to UD.
Step 5: UD verifies that $V(Sig_{SK_{OEM}}(VK_{TRK_{new}}, VK'_{UD}), VK_{OEM}) \stackrel{!}{=} 1$.
Step 6: UD verifies that $VK'_{UD} \stackrel{!}{=} VK_{UD}$. Then the new transponder TRK_{new} can be activated. The protocol is completed by sending a DONE-message to OEM.

Algorithm 3: Direct flashing protocol (with online connection to OEM).

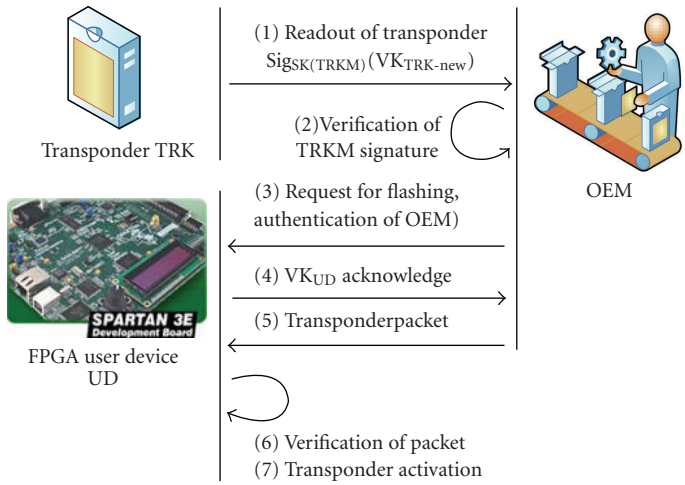

Figure 7: Key flashing without mediator.

Based on this evaluation, trust credentials (see Section 8.2) are issued to each individual SP.

To restrict access to the flashing capability of SP, as might be necessary in order to comply with the flashing policies of OEM a separate authentication of employees (SPE) working at SP is suggested. Such authorization of SPE can be done, for example, via a password (knowledge) or by biometrical identification (physical property).

8.2. Service Point as Trusted Platform. An SP constitutes a trusted platform as defined in [30] meaning that SP always acts as specified at any point in the protocol. At the same time, it needs to act reliably in order to enforce trust policies. Typically, an SP might reside in a hostile environment and can be accessible to malicious attackers. Therefore, some minimal functionalities of SP must be inherently secure and are encapsulated in a *Trusted Zone* (see Figure 9) as follows:

(i) generation of trust key pairs,

(ii) storage of private keys (SK_{SP}^{TD} and SK_{SP}),

(iii) signature generation,

(iv) enforcement of Trust Policy.

For all key pairs that are generated to be used as temporary trust keys, it has to be ensured that a SK_{SP}^{TD} is never communicated to another entity. Also, it has to be ensured that SK_{SP}^{TD} cannot be deducted from VK_{SP}^{TD}. This can be ensured with a proper key generation algorithm. Signing of messages has to be secure in order to prevent manipulation of signed data packets. This means that any signing operation is always done with the proper signing key residing in SP.

The main point of the trusted platform is the enforcement of trust policies. SP is issued a temporal trust key pair $(SK_{SP}^{TD}, VK_{SP}^{TD})$ as will be described in Section 8.3. This key pair expires after a timepoint T or after a certain amount of flashing procedures N. Expiration is enforced from within the *Trusted Zone* with a secure unforgeable counter that is keeping track of the number of flashing cycles. As soon as the counter value reaches N, the trust key pair is fully deleted and the counter is reset. SP also compares its system

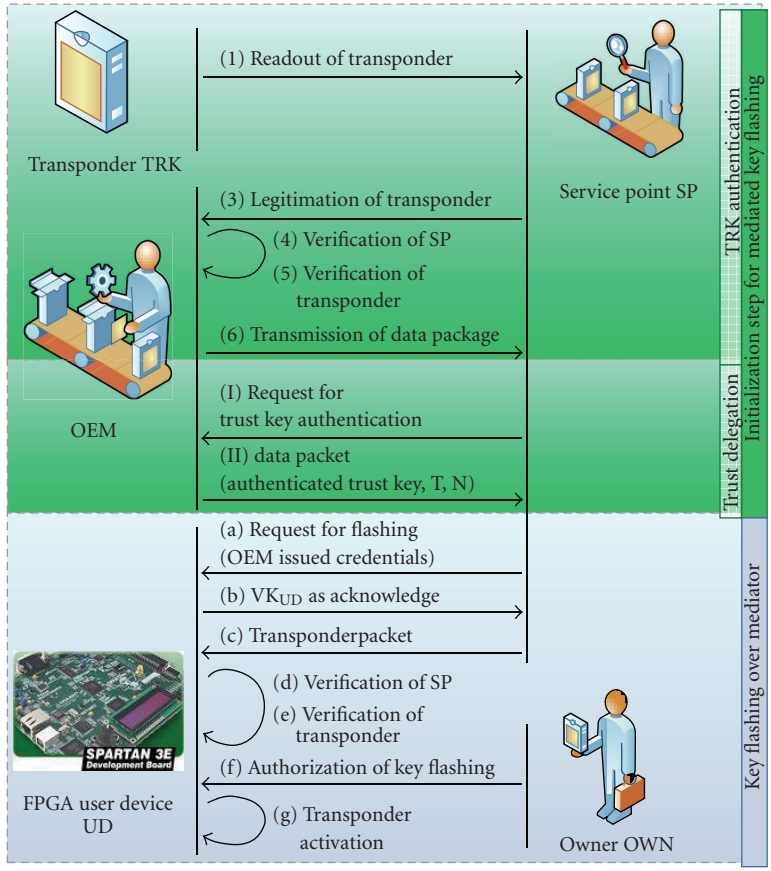

Figure 8: Flashing scheme.

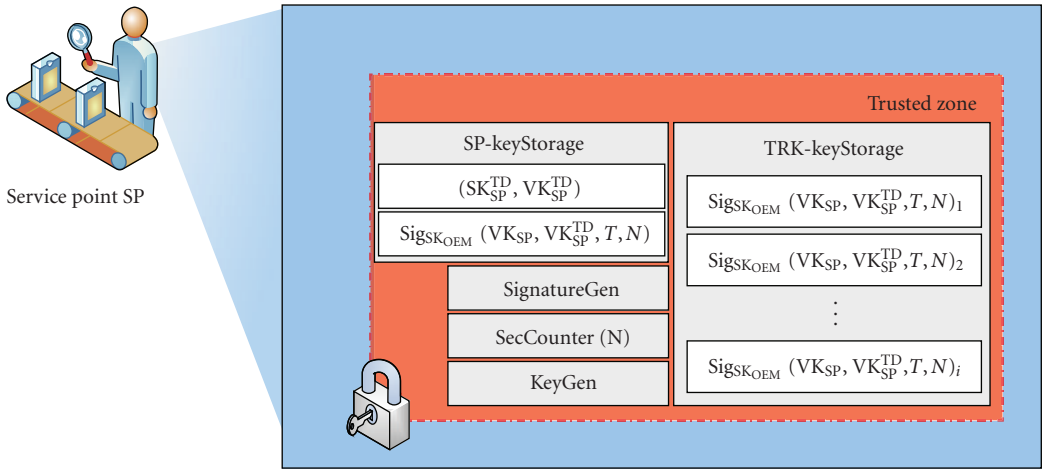

Figure 9: Service point as trusted platform.

time T_{SP} to T. If $T_{SP} \geq T$ the trust key pair is also fully deleted.

8.3. Trust Delegation and TRK_{new} Authorization. To be able to perform a key flashing procedure without an active link to OEM, a local representative has to be empowered by the OEM to perform the flashing, assuming that UD trusts only the OEM to flash legit keys. This is done by presenting a credential to UD accounting that flashing is authorized by OEM. The exchange of this credential is denoted in the following as *trust delegation*.

Algorithm 4 shows the protocol for instantiating a mediator that can flash a TRK into a UD. Steps I.1 to I.4 detail the trust delegation to SP. First of all, SPE is authenticated by SP

Requirements

(i) OEM has knowledge of VK_{SP} and VK_{TRKM}

Protocol

Step I.1: SPE presents his credential $CRED_{SPE}$ and SP authenticates SPE. After that SP is activated and communication to OEM is enabled.

Step I.2: SP creates a new key pair $(SK_{SP}^{TD}, VK_{SP}^{TD})$ and sends its ID together with the created verification key VK_{SP}^{TD} as a signed request $[Sig_{SK_{SP}}(VK_{SP}^{TD}), VK_{SP}]$ for a trust credential to OEM.

Step I.3: OEM verifies that SP and the respective verification key VK_{SP} is listed in the internal database of trusted mediators and that $V(Sig_{SK_{SP}}(VK_{SP}^{TD}), VK_{SP}) \stackrel{!}{=} 1$. In this case OEM creates a trust delegation credential $Sig_{SK_{OEM}}(VK_{SP}^{TD}, VK_{SP}, T, N)$ bound to SP with timestamp T and number of granted transactions N and sends it to SP.

Step I.4: SP receives $Sig_{SK_{OEM}}(VK_{SP}, VK_{SP}^{TD}, T, N)$ and stores it in the trusted storage. This step completes the trust delegation for flashing.

Step II.1: For a number of TRK_{new} to be flashed, SP reads out the respective certified verification keys $Sig_{SK_{TRKM}}(VK_{TRK_{new}})$ and sends $Sig_{SK_{SP}}(Sig_{SK_{TRKM}}(VK_{TRK_{new}}))$ to OEM.

Step II.2: OEM verifies that VK_{SP} and VK_{TRKM} are in the internal database of trusted peers and that $V(Sig_{SK_{SP}}(Sig_{SK_{TRKM}}(VK_{TRK_{new}})), VK_{SP}) \stackrel{!}{=} 1$ and $V(Sig_{SK_{TRKM}}(VK_{TRK_{new}}), VK_{TRKM}) \stackrel{!}{=} 1$. Afterwards OEM creates $Sig_{SK_{OEM}}(VK_{TRK_{new}}, VK_{SP})$ and sends it to SP.

Step II.3: SP receives $Sig_{SK_{OEM}}(VK_{TRK_{new}}, VK_{SP})$ and stores it in the trusted storage. This step completes the activation of the transponder TRK_{new} for flashing over SP.

Algorithm 4: Initialization step for mediated key flashing.

to prevent SP abusive operations. Afterwards, SP can connect to OEM and request a trust credential (Step I.1).

A trust credential consists of a cryptographic temporal trust key pair $(SK_{SP}^{TD}, VK_{SP}^{TD})$ with the public key VK_{SP}^{TD} being signed by OEM. Therefore, SP creates a fresh trust key pair. From this pair, SP sends the fresh verification key VK_{SP}^{TD} as well as its ID as a signed data packet to the OEM, while the secret key SK_{SP}^{TD} never leaves SP. The ID is the standard verification key VK_{SP} of the service point (see Algorithm 4 Step I.2). For ease of implementation, this can be replaced with a unique identifier that can then be matched by the OEM to the appropriate VK_{SP} stored in an internal database.

Using VK_{SP} OEM verifies the correctness of the data packet received. If SP is considered a trustworthy entity, that is, if it adheres to all of OEMs policies, OEM issues a trust credential. As shown in Step I.3 (Algorithm 4) it comprises the verification key of the trust key pair, the standard verification key (VK_{SP}) of SP, as well as a timestamp T, and a maximum transaction number N in an OEM signed packet. Through inclusion of VK_{SP}, OEM binds the trust key VK_{SP}^{TD} to SP. Later, this binding will be verified by UD (see Section 8.4). A trust key pair is only valid for a limited time and limited number of flashing operations after which it is deleted by SP. Step I.4 concludes the trust delegation phase with SP storing the OEM-signed packet in trusted storage for later use in the actual flashing procedure (Section 8.4).

In order to flash a TRK_{new}, the transponder needs to be authorized by OEM. The second part (Steps II.1–Step II.3) of the protocol shown in Algorithm 4 accomplishes this for a single TRK_{new}. If more than one TRK shall be set up for flashing, this part of the protocol is rerun for each additional TRK_{new}. SP reads out the verification key of TRK that previously has been certified and signed by TRKM. SP send this key as a self-signed message as shown in Step II.1 to OEM for authorization. In Step II.2, OEM verifies both signatures and ensures that TRKM as well as SP are trusted peers. If this is the case, OEM forms a data packet comprising the public key of TRK_{new} as well as the standard public key of SP and sends it as a signed message to SP. This message effectively binds TRK_{new} to SP, thus enabling solely SP to flash TRK_{new} into UD later on. The second part of the protocol in Algorithm 4 is finalized in Step II.3 by SP storing the received, signed message in its trusted module.

Only a limited number of authorized TRKs can be stored at any given point in time. As soon as a TRK has been authorized by the OEM, physical access to the TRK needs to be controlled. The authorization process of TRKs is the only step that demands a data connection between SP and OEM. This does not necessarily need to be an online connection since data could also be transported via data carriers such as CDs, memory sticks, or the like.

8.4. Flashing of TRK. The actual flashing of a TRK_{new} to a given UD is shown in Algorithm 5. It demands a valid new transponder TRK_{new} and authorization by OEM and OWN. Former either directly or delegated to SP using the credential introduced above, latter done by presenting a valid and linked TRK_{orig} assumed to be solely accessible by OWN. If an online connection to OEM is available, the protocol can be performed by UD and OEM directly as described in Section 7, with SP only relaying communication.

If SP has to act as an offline mediator, the initialization protocol (Algorithm 4) has had to be successfully completed. From there on, the flashing protocol commences as shown in Algorithm 5 with SP contacting UD and sending the trust credentials that SP has received from OEM (Step 1). In Step 2,

Requirements

(i) The initialization protocol has been completed successfully.

(ii) UD has knowledge of $\mathrm{VK}_{\mathtt{OEM}}$.

(iii) SP has a valid trust key pair and has not reached the maximum quota N of allowed flashing procedures.

Protocol

Step 1: SP contacts UD and sends $\mathtt{Sig}_{\mathrm{SK}_{\mathtt{OEM}}}(\mathrm{VK}_{\mathtt{SP}}, \mathrm{VK}_{\mathtt{SP}}^{\mathrm{TD}}, T, N)$ to UD.

Step 2: UD verifies $V(\mathtt{Sig}_{\mathrm{SK}_{\mathtt{OEM}}}(\mathrm{VK}_{\mathtt{SP}}, \mathrm{VK}_{\mathtt{SP}}^{\mathrm{TD}}, T, N), \mathrm{VK}_{\mathtt{OEM}}) \stackrel{!}{=} 1$ and sends back $\mathtt{Sig}_{\mathrm{SK}_{\mathtt{UD}}}(\mathrm{VK}_{\mathtt{UD}}, \mathrm{VK}_{\mathtt{SP}})$ as an acknowledge.

Step 3: OWN authorizes the start of a key flashing procedure by presenting a valid $\mathtt{TRK}_{\mathrm{orig}}$. UD authenticates $\mathtt{TRK}_{\mathrm{orig}}$ using the internally stored $\mathrm{VK}_{\mathtt{TRK}_{\mathrm{orig}}}$ and a PKC authentication protocol.

Step 4: SP sends the certified new key package $\mathtt{Sig}_{\mathrm{SK}_{\mathtt{SP}}^{\mathrm{TD}}}(\mathtt{Sig}_{\mathrm{SK}_{\mathtt{OEM}}}(\mathrm{VK}_{\mathtt{TRK}_{\mathrm{new}}}, \mathrm{VK}_{\mathtt{SP}}), \mathrm{VK}_{\mathtt{UD}})$ to UD.

Step 5: UD verifies that $V(\mathtt{Sig}_{\mathrm{SK}_{\mathtt{SP}}^{\mathrm{TD}}}(\mathrm{S}_{\mathrm{SK}_{\mathtt{OEM}}}(\mathrm{VK}_{\mathtt{TRK}_{\mathrm{new}}}, \mathrm{VK}_{\mathtt{SP}}), \mathrm{VK}_{\mathtt{UD}}), \mathrm{VK}_{\mathtt{SP}}^{\mathrm{TD}}) \stackrel{!}{=} 1$ and $V(\mathtt{Sig}_{\mathrm{SK}_{\mathtt{OEM}}}(\mathrm{VK}_{\mathtt{TRK}_{\mathrm{new}}}, \mathrm{VK}_{\mathtt{SP}}), \mathrm{VK}_{\mathtt{UD}}, \mathrm{VK}_{\mathtt{OEM}}) \stackrel{!}{=} 1$. Then the new transponder $\mathtt{TRK}_{\mathrm{new}}$ can be activated: $\mathrm{TRK}_{\mathrm{new}} \mapsto \mathrm{TRK}_{\mathrm{orig}}$. The protocol is completed by sending a DONE-message to SP.

Algorithm 5: Indirect flashing protocol over a trusted mediator (no online connection to OEM).

UD verifies these credentials using the OEM's verification key that is already embedded in UD (Section 6.2.4). The verification key of the trusted key pair $\mathrm{VK}_{\mathtt{SP}}^{\mathrm{TD}}$ is stored temporarily for use in Step 6. As an acknowledge message, UD sends a self-signed packet to SP that includes its own public key as well as the standard public key of SP.

Since OWN has to authorize the flashing procedure he presents his credential in form of an original $\mathtt{TRK}_{\mathrm{orig}}$ already linked to UD. UD authenticates $\mathtt{TRK}_{\mathrm{orig}}$ by means of a public key authentication protocol (Step 3). Only if $\mathtt{TRK}_{\mathrm{orig}}$ has been successfully authenticated UD will accept a $\mathtt{TRK}_{\mathrm{new}}$ being flashed into UD. In Step 4, SP sends the authorized data packet for the TRK to be flashed that it has received from the OEM during the second phase of the initialization procedure (Section 8.3). It is annotated with UD's public key $\mathrm{VK}_{\mathtt{UD}}$ and additionally signed by SP using the trust key pair. That way, the $\mathtt{TRK}_{\mathrm{new}}$ is bound to UD. To finalize the protocol, UD enforces the flashing policies in Step 5. First it verifies if the signature of SP is correct and has used the trust key pair. If that is the case, it verifies the correctness of the OEM's signature on the data packet for $\mathtt{TRK}_{\mathrm{new}}$. Since this data packet has been bound to UD, UD verifies if its own public key has been used to annotate $\mathrm{VK}_{\mathtt{TRK}_{\mathrm{new}}}$. Only a correctly annotated $\mathrm{VK}_{\mathtt{TRK}_{\mathrm{new}}}$ will be accepted, all others are dismissed and will not be stored into UD.

In the case of successful verification, UD accepts the new token $\mathtt{TRK}_{\mathrm{new}}$ and adds $\mathrm{VK}_{\mathtt{TRK}_{\mathrm{new}}}$ to its internal list of linked tokens thus transforming $\mathtt{TRK}_{\mathrm{new}}$ into a $\mathtt{TRK}_{\mathrm{orig}}$. The correct and full flashing is then reported back to SP. Subsequently, SP will log this as a successful flashing procedure and decrement its internal counter for allowed flashing processes.

8.5. Entity Requirements. Regarding the proposed flashing protocols certain requirements for the entities' functionalities have to be satisfied. An overview is given in Table 4. Data management is one of the key requirements in the protocol in the sense that public key data needs to be stored. Secure storage for delegated trust has some additional requirements such as intrusion detection to protect data from being altered in any way. At the same time, it is mandatory that this data is always changed correctly as demanded by the protocol. Also, the OEM's public key needs to be firmly embedded into the entities and must not be altered in any way. Otherwise, the OEM cannot be identified correctly within the proposed protocols.

Table 4: Entity requirements.

	OEM	SP	UD	TRK
Initiate communication	•	•		
Acknowledge communication	•		•	
Generation of keypairs	•	•		•
Signature generation	•	•	•	
Signature verification	•	•	•	
Datamanagement for suppliers	•			
Datamanagement for user devices	•			
Datamanagement for service points	•			
Datamanagement for TRKs		•	•	
Secure storage for delegated trust		•		
Knowledge of OEM's public key		•	•	

9. Implementation

The protocol has been implemented as a proof of concept in a prototypical setup based on a network of standard PCs representing OEM and SP (see Figure 10). Furthermore, Digilent *Spartan3E Starter Boards* with a Xilinx XC3S500 FPGA represent TRKs and UDs. TRK, SP, and UD have to be connected when flashing the key. The OEM connection needs to be established anytime prior to the flashing according to the proposed protocol and is connected via TCP/IP to the SP.

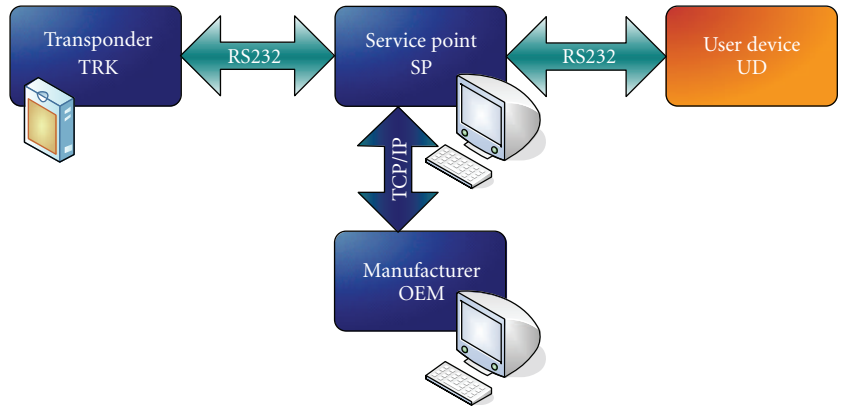

Figure 10: Component interaction.

Table 5: Parameters for RSA-System.

Key length	1024 Bit
Exponent	$2^{16} + 1$ (65537)
Padding scheme	PKCS#1 v1.5
Signature scheme	PKCS#1 v1.5
Hashing scheme used for signing	SHA1

All other communication is done over RS232 interfaces that are available both on PC and the FPGA boards. These can be substituted for other communication structures if needed, that is, wireless transmitters.

9.1. Choice of Cryptographic Primitives. The proposed key flashing concept demands asymmetric encryption and a cryptographic hash function. RSA [31] is chosen for encryption and signing, SHA1 [32] for hash functionality. Both schemes are today's standard and have not been broken yet, but can be substituted in our implementation for more secure schemes such as HECC if needed. RSA as well as SHA1 implementations are freely available as software and hardware modules for numerous platforms. RSA parameters used in the prototype are given in Table 5.

All signatures in our context are SHA1-hash values of data that has been encrypted according to the signing scheme PKCS#1 v1.5 [33]. Such a signature has a length of 128 Byte when using a key length of 1024 bit and hash values of 160 bit length.

9.2. OEM/Service Point-Software Platform. Both components `OEM` and `SP` have been implemented on a standard PC in software under the .NET framework version 2.0 [34] using C#. The .NET framework provides the Berkeley Socket-interface for communication over the PC's serial interface. It also includes the `Cryptography`-namespace providing all needed cryptographic primitives including hash functions and a random number generator that are based on the FIPS-140-1 [35] certified Windows CryptoAPI. The software is modularized to enable easy exchange of functional blocks and seamless replacement of algorithms. Software modules communicate only over defined interfaces to enable full functional encapsulation. For ease of usage, a graphical user interface (GUI) is included as well in both entities.

9.3. Transponder/UserDevice—FPGA platform. The targeted user device is an FPGA. To ease reuse of functionalities the exemplary `TRK` has been implemented on FPGA as well, but can also be integrated into a smart card or RFID chip as long as the appropriate cryptographic primitives are provided.

In the prototypical setup, we used a MicroBlaze-based ECU (see Figure 3) for both `UD` and `TRK`. We omitted the coprocessor and implemented all functionality on the MicroBlaze including cryptographic functions. Hardware peripherals such as an LCD controller have been integrated for debugging purposes. To enable handling of big numbers, as are used in the cryptographic functions of the protocol, the libraries `libtommath` [36] and `libtomcrypt` [37] are used. Only necessary components have been extracted from those libraries and are integrated into `TRK` and `UD`.

9.4. Resource Usage. The resource usage of the components `OEM` and `SP` are very similar, since almost identical functional software blocks are used in both. Table 6 gives an exemplary overview of the lines of code of the `OEM` implementation. The memory footprint of the compiled `OEM` implementation is 129 KB (139 KB for the `SP` implementation). At start up, 15400 KB of main memory is used. The execution times for RSA- and SHA1-operations were measured on a PC (2 GHz, 1024 MB RAM) and are all in the range of milliseconds.

Resource usage of the FPGA-based components `UD` and `TRK` are given in Table 7. By implementing all functionality on a MicroBlaze softcore, the hardware usage is quite moderate. On the other hand, the software footprint is 295 KB for the `UD` implementation, due to the nonoptimized memory usage of the crypto library.

Shown in Table 8 are the execution times of the diverse protocol instances. The duration of parts of the protocol that are based solely on `OEM` and `SP` is in the area of few milliseconds. As soon as mobile devices (`UD`, `TRK`) process parts of the protocol, speed is declining since all crypto operations are currently carried out on an embedded

Table 6: Properties of OEM component.

Module	Lines of code	Percentage
Main application	1234	41.77
GUI	264	8.94
Cryptography	385	13.03
Interaction	383	12.97
Communication	545	18.45
Data management	143	4.84
Total	2954	100

Table 7: FPGA resources.

Slices	1.791 of 4.656 (38%)
Slices: FlipFlops uses	1.590 of 9.312 (17%)
Slices: LUTs used	1.941 of 9.312 (20%)
BlockRAMs used	16 of 20 (80%)
Equivalent logic cells	1.135.468
Minimal clock period	18,777 ns
Maximum clock frequency	53,257 MHz

Table 8: Protocol execution times.

Protocol instance	Duration (min:sec.ms)
ReadOut of transponder	01:32.000
Mutual authentication of UD and TRK	03:14.000
Direct keyflashing	
Keyflashing to transponder by OEM	23:50.000
Keyflashing by servicePoint	
Delegation of trust OEM to SP	00:00.350
Transponderdelegation	00:00.250
Keyflashing to transponder by SP	12:43.000

microcontroller. Main factor here is the RSA decryption operation. With appropriate hardware support, choice of parameters, cryptosystem, and substantial speedups can be achieved as shown in [16].

10. Security Analysis

Looking at the security of the proposed concept some points can be identified where security relies on policies and implementing rules while other issues are covered by design.

Using PKC primitives and trusted computing approaches, the protocol ensures confidentiality of secret keys and mutual authentication of SP and OEM, OWN and UD, SP and UD, SP, and SPE. Due to the necessity of online independence, there are some assumptions that have to be made to guarantee security. This is mainly the trustworthiness of the SP in combination with the physical protection of any authorized TRK_{new} and all TRK_{orig}.

If these assumptions are broken, for example, by theft of authorized TRK, the corresponding SP and the SPE password, unauthorized flashing may be possible. As countermeasures, the usage of the protocol can be adapted to dilute effects of such events. So, the number of allowed authorized TRK should be as low as possible and the SP should be implemented using trusted components and based on a trusted platform. Secrets should be especially protected against misuse by a physical attacker.

There certainly is a tradeoff between security and usability of the flashing scheme, since the protocol has been designed for real-world implementation.

10.1. Security of Direct Flashing Scenario. In the flashing scenario with no mediator (Section 7), an illegal flashing of TRK is not possible. The flashing procedure is authorized through the OEM directly. Only the OEM is considered trustworthy enough to accept flashing commands from. By verifying the signature on a TRK's key, it can be checked if a certain TRK has been manufactured by a certified supplier TRKM or not. Certification policy ensures that such a TRK has a unique ID, unique cryptographic keys, and secret key material is solely known to the TRK itself and is nowhere else available or reproducible.

Verifying the signature of UD and the mutual authentication phase of OEM and UD, OEM can be sure that a UD is targeted in the flashing procedure that has been manufactured by OEM. In turn, UD can be sure that its communication partner is the OEM.

Binding the key material to be flashed to a dedicated UD by incorporating a mutual signature of the key material enforces that a certain transponder is flashed only into a single UD. Also, it can be enforced that the packet containing the VK_{TRK} is only used for a single flashing procedure, thus countermeasuring replay attacks. Neither unrecognized mutation of dedicated parts of this communication packet is possible, nor forging the signatures on data, due to the security assumptions of the cryptographic primitives. No confidential information is included in any communication packet. By activating the VK_{TRK} inside the UD only after a successful transmission without errors, it is ensured that the TRK has no previous utility value. Readingout the TRK's VK_{TRK} has no benefit to an attacker. Therefore, a TRK has not to be stored away safely before linking it to a UD. Loss of an unlinked TRK does also not lead to any security issues.

If all entities involved in the flashing procedure adhere to the protocol, abolish all data resulting from a disrupted flashing procedure, and implement all cryptographic primitives securely, an attacker is not able to carry out an illegal flashing procedure.

10.2. Security of Mediator Flashing Scenario. In the flashing scenario involving a mediator as described in Section 8, the flashing procedure is legitimated through OWN directly by presenting a second TRK that is already linked to UD during the last phase of the flashing process. A UD receiving the final communication packet carrying the VK_{TRK} to be flashed can verify if the sender of the packet is legit and trustworthy by checking the signatures of the OEM as well as checking the trust credentials of the SP. Therefore, the UD directly enforces the policy that only certified parties may flash a VK_{TRK} by dismissing any received VK_{TRK} as soon as a invalid signature is detected.

Usage of SP through a SPE is restricted and protected by appropriate access control, so no malicious outsider can flash a VK_{TRK}. Since trust has to be redelegated after a certain time or amount of flashing procedures, it is not possible to haphazardly flash TRKs.

No sensitive data is transmitted during the flashing protocol that might compromise the system's security. It seems that the data packet sent by OEM to SP might be highly sensitive, but even if an attacker were able to access the keypair forming the delegation of trust, it would not be possible to authenticate new TRK's with the OEM since this demands knowledge of SK_{SP} known only to SP itself. TRK's already authenticated by the OEM can also not be flashed into a UD, since it will be impossible for an attacker to correctly sign the data packet containing the TRK's public key, because the signing key SK_{SP}^{TD} is also known solely by SP.

In this second flashing scenario, it is again ensured that the TRK has no previous utility value before finally linking it to a UD in the final step of the protocol. The loss of of an unlinked TRK does not lead to any security issues as long as it has not been authorized by the OEM. Therefore all unauthorized TRK do not pose a security risk. Authorized TRKs do need to be stored away securely since they can be flashed into a UD, but only with the SP that is linked to the TRK. As before, no attacker may be able to carry out an illegal flashing procedure as long as all entities involved in the flashing procedure adhere to the protocol, abolish all data resulting from a disrupted flashing procedure, and implement all cryptographic primitives securely.

10.3. Potential Risks. Although the technical aspects of the flashing protocols can be secured against manipulation and tampering, there are still some risks involved resulting from nontechnical aspects. A malicious insider such as a SPE might be able to gain access to the SP, an authenticated TRK_{new} as well as a UD and the corresponding TRK_{orig}. Only if SPE has access to all aforementioned entities, then it is possible for him to flash TRK_{new} into UD, unknown to the legal owner OWN. Although such malicious misbehavior of SPE cannot be prohibited, it can at least be traced by logging all activity inside the SP.

A similar risk is faulty implementations of security primitives that are used in the protocol leading to a leak of secret cryptographic keys, thus enabling an attacker to impersonate an entity. The two main concerns regarding security leaks lie in the nontechnical aspects of the flashing protocol through mediators.

10.3.1. Social Engineering. Since the flashing of keys involves human interaction, this can offer an entry point for an attacker using social engineering [38–40]. A conceivable entry point is the SPE. If an attacker is able to extract the credentials from SPE, he can gain access to SP and therefore flash TRKs into any user device of OEM. This is a widely known issue in security systems in general that can only be countermeasured by proper training of SPE to enhance security awareness. Any misuse of the system through an SPE can be tracked if a secure log of all activities is provided within the trusted part of SP. As soon as misuse is detected, the trust delegation to SP can be revoked by OEM through not reissuing a trust keypair. Therefore, damage can be limited to flashing of the TRK_{new} already prepared for introduction into a UD.

Since the flashing scheme demands for authentication of the procedure through OWN, it is necessary to ensure the security of the second TRK_{orig} to be presented at the final stage of the process. If the second TRK_{orig} is in possession of an attacker and additionally the attacker has access to SP, he is able to flash an additional TRK into one UD. This attack is limited to a single UD, thus representing a fairly small risk, which on the other hand can easily be countermeasured.

10.3.2. SP Theft Scenario. If an SP carrying a valid trust key falls into the hands of a malicious attacker, the credential of SPE must also be known to the attacker in order to use SP to flash TRKs. Additionally, it is mandatory that the attacker also has a TRK_{new} in his possession that has already been certified by OEM. Additionally, TRK_{new} has to be bound to the stolen SP. If no such TRK_{new} is available, the system is not compromised. Even in case of such an aggressive attack the risk to the overall system is minimized, since only a very limited number of TRKs is flashable and a trust revocation can be carried out.

The risk level for such a scenario can be adapted through appropriate policies based on risk assessment through OEM. The most aggressive policy is not to allow flashing through a mediator. A minimal risk policy is to only have a single TRK on location that can be flashed into a UD. Only after it has been flashed successfully, does the OEM authenticate another TRK. While providing higher security, such restrictive policies will naturally inhibit usability.

11. Conclusions and Future Work

Access control systems are an important part in many systems such as vehicles or expensive machinery. Authentication protocols based on Public Key Cryptography offer advantages in logistics and key handling. These protocols are computationally very expensive but can be accelerated in hardware. In this paper, we presented a high-speed crypto architecture based on FPGA for fast authentication protocols based on HECC. Exploiting the properties of PKC, we introduced a scheme for pairing user devices (UD) and transponder tokens (TRK) by flashing public keys into UD. Compared to current key flashing procedures, our proposed protocol eliminates some logistic issues. TRKs do not have to be physically stored securely in SP any more. Also, shipment of TRKs does not have to be secured physically. Today OEM has to fully trust an SP to flash TRKs in the field. With our protocol the amount of trust in an SP can now be reduced to allow for risk management.

Security of the system is guaranteed by appropriate policy enforcement and usage of secure cryptographic primitives. No online connection is mandatory for linking a new transponder (TRK) for user authentication to a user device (UD). This makes it very practical for scenarios if TRK's have

to be replaced in the field with no intact communication infrastructure. It is applicable for a variety of embedded systems that need to implement and enforce access or usage restrictions in the field. We have shown the portability of the concept to non-FPGA platforms by implementing the protocol as a proof-of-concept using a combination of PC-based and FGPA-based protocol participants.

Flashing speed is of utmost importance in real-world implementation. To make allowance for a real world integration of the proposed flashing schemes, optimization is needed regarding usage and speed of the computational units involved. In the current prototype, the MicroBlaze processor has been used for simplicity and to show that the protocol can already be easily deployed in microprocessor driven embedded system such as the automotive domain. With the coprocessing unit in Section 7, very short computation times are achieved, and on a public key cryptosystem with a high security level than RSA. Adapting this system to the complete flashing scheme is target of future work and promises dramatic acceleration of the entire key flashing procedure.

One crucial point is the protection of the TRK's public key stored in the UD against physical attackers. Means to countermeasure attacks that might alter stored keys on a physical level need to be investigated in the future.

References

[1] H. Wallentowitz and K. Reif, *Handbuch Kraftfahrzeugelektronik: Grundlagen, Komponenten, Systeme, Anwendungen*, Vieweg, Wiesbaden, Germany, 2006.

[2] Hilti Corporation, "Electronic theft protection".

[3] R. M. Needham and M. D. Schroeder, "Using encryption for authentication in large networks of computer," *Communications of the ACM*, vol. 21, no. 12, pp. 993–999, 1978.

[4] T. Okamoto, "Provably secure and practical identification schemes and corresponding signature schemes," in *Proceedings of the 13th Annual International Cryptology Conference on Advances in Cryptology (CRYPTO '93)*, pp. 31–53, Springer, Santa Barbara, Calif, USA, 1993.

[5] C. P. Schnorr, "Efficient identification and signatures for smart cards," in *Proceedings of the 9th Annual International Cryptology Conference on Advances in Cryptology (CRYPTO '89)*, pp. 239–252, Springer, Santa Barbara, Calif, USA, August 1989.

[6] HIS Security Module Specification v1.1, Herstellerinitiative Software (HIS), 2006, http://www.automotive-his.de/.

[7] HIS-Presentation 2004-05, Herstellerinitiative Software (HIS), 2005, http://www.automotive-his.de/.

[8] G. de Boer, P. Engel, and W. Praefcke, "Generic remote software update for vehicle ecus using a telematics device as a gateway," *Advanced Microsystems for Automotive Applications*, pp. 371–380, 2005.

[9] A. Adelsbach, U. Huber, and A.-R. Sadeghi, "Secure software delivery and installation in embedded systems," in *Proceedings of the 1st International Conference on Information Security, Practice and Experience (ISPEC '05)*, R. H. Deng, Ed., vol. 3439 of *Lecture Notes in Computer Science*, pp. 255–267, April 2005.

[10] G. Dr-Ing and H. Brinkmeyer, "Authentikationsverfahren für fahrzeuganwendungen," *VDI-Berichte*, no. 1287, pp. 819–833, 1996.

[11] Microchip, "Keeloq authentication products," http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=2074.

[12] B. Drisch and T. Zeggel, "Unterstützende Hardware-sicherheitsmodule für Automotive-anwendungen: voraussetzungen für die sichere umsetzung kryptographischer verfahren in fahrzeug-steuergeräten," *VDI Berichte*, no. 2016, pp. 147–156, 2007.

[13] L. Batina, N. Mentens, K. Sakiyama, B. Preneel, and I. Verbauwhede, "Public-key cryptography on the top of a needle," in *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS '07)*, pp. 1831–1834, May 2007.

[14] W. Diffie and M. E. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, 1976.

[15] A. Klimm, O. Sander, J. Becker, and S. Subileau, "A hardware/software codesign of a co-processor for real-time hyperelliptic curve cryptography on a spartan3 fpga," in *Proceedings of the 21st International Conference on Architecture of Computing Systems (ARCS '08)*, U. Brinkschulte, T. Ungerer, C. Hochberger, and R. G. Spallek, Eds., vol. 4934 of *Lecture Notes in Computer Science*, pp. 188–201, Springer, 2008.

[16] A. Klimm, O. Sander, and J. Becker, "A microblaze specific coprocessor for real-time hyperelliptic curve cryptography on Xilinx FPGAs," in *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS '09)*, pp. 1–8, IEEE Computer Society, Rome, Italy, 2009.

[17] Atmel Corporation, "Ata5811/5812, uhf ask/fsk transceiver," 2006.

[18] L. Batina, D. Hwang, A. Hodjat, B. Preneel, and I. Verbauwhede, "Hardware/software co-design for Hyperelliptic Curve Cryptography (HECC) on the 8051 μP," in *Proceedings of the 7th International Workshop on Cryptographic Hardware and Embedded Systems (CHES '05)*, Lecture Notes in Computer Science, pp. 106–118, September 2005.

[19] H. Cohen, G. Frey, and R. Avanzi, *Handbook of Elliptic and Hyperelliptic Curve Cryptography*, Chapman & Hall/CRC, Boca Raton, Fla, USA, 2006.

[20] Xilinx, "Picoblaze 8-bit embedded microcontroller user guide," 2005.

[21] K. Okeya, K. Schmidt-Samoa, C. Spahn, and T. Takagi, "Signed binary representations revisited," in *Proceedings of the 24th Annual International Cryptology Conference on Advances in Cryptology (CRYPTO '04)*, pp. 123–139, Santa Barbara, Calif, USA, 2004.

[22] R. Fan, *On the efficiency analysis of wNAF and wMOF*, Diploma thesis, September 2005, Supervised by Professor Dr. Tsuyoshi Takagi.

[23] S. K. Jain, L. Song, and K. K. Parhi, "Efficient semisystolic architectures for finite-field arithmetic," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 6, no. 1, pp. 101–113, 1998.

[24] M. W. Zuccherato, "An elementary introduction to hyperelliptic curves," Tech. Rep. CORR 96, University of Waterloo, Ontario, Canada, 1996.

[25] P. Engel and G. Hildebrandt, "Die rhythmischen schwankungen der reaktionszeit beim Menschen," *Psychological Research*, vol. 32, no. 4, pp. 324–336, 1969.

[26] N. Boston, T. Clancy, Y. Liow, and J. Webster, "Genus two hyperelliptic curve coprocessor," in *Proceedings of the 4th International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 400–414, Springer, 2002.

[27] A. Klimm, M. Haas, O. Sander, and J. Becker, "A flexible integrated cryptoprocessor for authentication protocols based on hyperelliptic curve cryptography," in *Proceedings of the International Symposium on System-on-Chip (SoC '10)*, Tampere, Finland, September 2010.

[28] A. Weigl, K.-E. Weiss, C. Schroff et al., "Vehicle security device," Patent EP0 925 209, 2001, http://www.freepatentsonline.com/EP0925209B1.html.

[29] M. Hirozawa, A. Okamitsu, K. Adachi, and H. Tagawa, "Antivehicle-thief apparatus and code setting method of the apparatus," Patent EP0 695 675, 1999, http://www.freepatentsonline.com/EP0695675B1.html.

[30] Trusted Computing Group, TPM main specification v1.2.

[31] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[32] "National Institute of Standards and Technology (NIST): FIPS-180-2, Secure Hash Standard (SHS)," 2002, http://www.itl.nist.gov/fipspubs.

[33] "RSA Laboratories Inc: RSA Cryptograpy Standard PKCS No.1," http://www.rsa.com/ .

[34] MSDN, ".net framework class library—rsacryptoser-viceprovider class," http://msdn.microsoft.com/en-us/library/system.security.cryptography.

[35] D. L. Evans, K. H. Brown, A. Director, and W. M. Director, "Fips 140-1: security requirements for cryptographic modules," Category Computer Security, Gaithersburg, Md, USA, 1994.

[36] T. S. Denis, "Libtommath," http://math.libtomcrypt.com/.

[37] T. S. Denis, "Libtomcrypt," http://libtomcrypt.com/.

[38] S. Schumacher, "Adminsalbtraum," vol. 7, pp. 11–13, 2009, http://grundschutz.info/fileadmin/kundenbereich/Dokumente/Grundschutz 7-2009 11 13.pdf.

[39] S. Schumacher, "Admins albtraum," vol. 8, pp. 8-9, 2009, http://grundschutz.info/fileadmin/kundenbereich/Dokumente/Grundschutz 8-2009 8 9.pdf.

[40] S. Schumacher, "Admins albtraum," vol. 10/11, pp. 21-22, 2009.