# A Secure Keyflashing Framework for Access Systems in Highly Mobile Devices

Alexander Klimm, Benjamin Glas, Matthias Wachs, Jürgen Becker, and Klaus D. Müller-Glaser
Institut für Technik der Informationsverarbeitung, Universität Karlsruhe (TH)
email: {klimm,glas,mueller-glaser,becker}@itiv.uni-karlsruhe.de

***Abstract*—Public Key Cryptography enables for 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 to mobile systems, the public key of authorized tokens need 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 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 for 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.**

## I. Introduction

Embedded systems in various safety critical application domains like automotive, avionic and medical care perform more and more complex tasks using distributed systems like networks of electronic control units (ECUs). The introduction of Public-Key Cryptography (PKC) to embedded systems provides essential benefits for the production 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 platforms is minimized. At the same time only the private key of the platform itself needs to be stored secretly inside each entity - in contrast to symmetric crypto systems where a secret key needs to be stored inside several different entities at the same time. In context of PKC, if one entity is compromised, the others remain uneffected.

Computational efforts of cryptographic functionalities 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 provides for high speed up of demanding PKC crypto systems such as *hyperelliptic curve cryptography* (HECC). By adding dedicated hardware modules for certain parts of a crypto algorithm, a substantial reduction of computation time can be achieved [10] [9].

Besides encrypting or signing messages, PKC can be employed to control user access to a device via electronic tokens. Examples for this are Remote Keyless Entry (RKE) systems [17] in the automotive domain, or Hilti's TPS technology [2]. These systems incorporate contactless electronic tokens that substitute classical mechanical keys. The owner or authenticated user identifies himself to the user device (UD) by possession of the token. The UD and token are linked. Only if a linked token is presented to the UD it is enabled or access to the UD is granted. In order to present a token to a UD, information needs to be exchanged between the two over an usually insecure channel. To prevent the usage of a device or its accessibility through an unauthorized person this data exchanged needs to be secured.

Authentication schemes based on Public Key Cryptography such as the Needham-Schroeder protocol [11], Okamoto Protocol [12], and Schnorr-Protocol [16] provide authentication procedures where no confidential data needs to be transmitted. Secret keys need only be stored in the tokens and not in the UD thus omitting the need for costly security measures in the UD. Only public keys need to be introduced into the UD (see section II). This operation certainly does need to be secured against attacks. For real-world operation this operation is done in the field where the UD is not necessarily under the control of the manufacturer (OEM) and a live online connection to the OEM is not possible.

In this paper we propose a system to introduce public keys into FPGA based user devices to pair these with a new token. The proposed key flashing method allows for authorization of the flashing process through an OEM. At the same time it can be carried out with the UD in the field and with no active online connection while flashing the key.

Introduction or flashing of new keys to an embedded device can be seen as a special case of a software update. Here the main focus is usually on standardization, correctness, robustness, and security. Recent approaches for the automotive area have been developed e.g. in the german HIS [8], [7] or the EAST-EEA [3] project. A general approach considering security and multiple software providers is given in [1]. 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 keyflashing scenario.

The remainder of this paper is structured as follows. In section II we present the basic application scenario followed by a short introduction of public key cryptography in section III. The requirements for the targeted scenario are described in IV. In section V the protocol is shown and some implementational results are given in section VI. We conclude in section VII.

## II. Application Scenario

A mobile user device (UD) such as a vehicle, construction machine or special tool with restricted access is fitted with an FPGA based access control system. This allows only the owner or certified users access to the device's functionalities or even the device itself. This is achieved with a transponder (TRK) serving as an electronic version of a mechanical key. The transponder is able to communicate to the UD via a wireless communication channel. The user device accepts a 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.

Authentication is done using Public Key Cryptography (PKC). The Public Key of the transponders are stored securely inside the user device thus establishing a "'guest list"' of legal users to the device. During production two initial Public Keys are introduced into the user device.

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, replacement of the transponder's public key usually needs to be done in the field. This might include very remote areas with minor to none communication infrastructure.

## III. Basic PKC Functionalities

In 1976, Whitfield Diffie and Martin Hellman introduced PKC crypto systems [6]. Two different keys are used, one public and the other secret (SK). SK and VK are a fixed and unique keypair. It is computational infeasible to deduce the private or secret key (SK) from the public key[1] (VK). With VK 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 $M_{Alice,plain}$ to an entity Bob, it encrypts it with Bobs public key $\mathrm{VK}_{Bob}$. Only Bob can retrieve the plain text from the encrypted message, by applying the appropriate decryption algorithm using his own secret key SK.

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 by 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. This key is used to encrypt the hash value of the message $M_{Bob,plain}$. The resulting value $\{HASH(M_{Bob,plain})\}_{sig}$ is transmitted the together with $M_{Bob,plain}$. A receiver can validate the signature by using Bob's public key and retrieving $HASH(M_{Bob,plain})$. From $M_{Bob,plain}$ the receiver can reconstruct the received hash value and compare it with the decrypted value. If both match the signature has been validated.

## IV. Situation and requirements analysis

In our application scenario we have the following main entities:

- A user device UD that can only be accessed or used by an authenticated user
- A human user OWN. He is authorized to access or use UD if he possesses a legit token
- A transponder key token $\mathrm{TRK}_{orig}$ originally linked to UD and a second token $\mathrm{TRK}_{new}$ that shall be flashed to UD additionally.
- The manufacturer OEM that produces UD

UD accepts a number of TRK to identify an authenticated human user OWN of the UD. At least two tokens are linked to a UD, by storing the respective public keys $\mathrm{VK}_{TRK}$ 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 are used for any authorization check of TRKs using any PKC authentication protocol (e.g. [11], [12], [16]). The OEM's public key $\mathrm{VK}_{OEM}$ is stored in the UD as well.

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

### A. Goals

A new $\mathrm{TRK}_{new}$ should be linked to a UD to substitute for an original $\mathrm{TRK}_{orig}$ that has been lost or is defective. From this point on we'll call the process of linking $\mathrm{TRK}_{new}$ to a UD *flashing*. Flashing a TRK should be possible over the complete life cycle of the UD. When flashing the UD it is probably nowhere near the OEM's location while flashing of a TRK needs to be explicitly authorized by the OEM. Any TRK can only be flashed into a single UD. Theft or unauthorized use of the UD resulting from improper flashing of the TRK needs to be prohibited.

In addition we demand that online connection of UD and OEM during flashing a TRK must not be imperative.

### B. Security Requirements

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

- **Correctness:** In absence of an adversary the protocol has to deliver the desired result, i.e. after complete execution of the protocol the flashing should be accomplished.
- **Authentication:** The flashing should only be feasible if both OEM and OWN have been authenticated and have authorized the operation.
- **No online dependency:** The protocol shall not rely on any live online connection to the OEM.
- **Confidentiality:** No confidential data like secret keys should be retrievable by an adversary.

### C. Adversary model

We assume an in processing power and memory polynomially bounded adversary $\mathcal{A}$ that has access to all inter-device communications, meaning he can eavesdrop, delete, delay, alter, replay or insert any messages. We assume further that

[1] In the case of signature schemes the public key is often called verification key.

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 hashing function used as a random oracle.

## V. Key Flashing Concept

Focus of the proposed key flashing protocol is the introduction of a public key $\mathtt{VK}_{TRK}$ into UD. We abstract over the implementation of communication interfaces, PKC systems as well as the immediate implementation of the devices and entities themselves.

Two basic Flashing Scenarios are conceivable. One is that TRKs are flashed directly by the OEM, either during production or via an online connection. We concentrate on the second one, flashing of TRKs through an authorized service point (SP) with no immediate online connection to the OEM.

### A. Entities

In addition to the entities introduced above (UD, OWN, TRK and OEM) we use two additional participants, namely a service point SP and an employee SPE of this service point conducting the flashing procedure.

*1) OEM - Manufacturer:* The OEM manufactures the UD and delivers it to OWN. OWN is issued the corresponding TRKs linked to the UD. All UDs are obviously known to the OEM. The verification keys $\mathtt{VK}_{TRK}$ are stored by the OEM together with their pairing to the UD. Therefore the OEM knows what TRK is linked to what UD. We regard the entity OEM as a trusted central server with a database functionality.

The OEM can store data, sign data with $\mathtt{SK}_{OEM}$ and send data. It possesses all cryptographic abilities for PKC based authentication schemes and can thereby authenticate communication partners.

*2) 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 $\mathtt{VK}_{TRK}$ are stored in the UD. Additionally the public key of the OEM $\mathtt{VK}_{OEM}$ is stored in the UD and can not be erased or altered in any way and UD has a OEM-issued certificate for it's own public key certifying being a genuine part. UD grants read access to all stored public keys. Write access to the memory location of $\mathtt{VK}_{TRK}$ 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.

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

*4) TRK - Transponder:* TRK[2] possesses a keypair $\mathtt{VK}_{TRK}$/$\mathtt{SK}_{TRK}$ for PKC functionality. It is generated inside the TRK to ensure that the secret key $\mathtt{SK}_{TRK}$ is known solely to TRK. Read access to $\mathtt{VK}_{TRK}$ is granted to any entity over a communication interface.

TRK possesses cryptographic primitives for PKC based authentication schemes on prover's side and can thereby be authenticated by communication partners.

*5) SP - Service Point:* SP is a service point in the field such as a wholesaler, certified by the OEM. Typically a SP is a computer terminal. Access to the terminal is secured by means of a password as in standard PC practice. A SP can communicate to the OEM as well as to the UD. At the same time it is able to read the $\mathtt{VK}_{TRK}$ of any TRK.

Furthermore the 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:

- storage of authorized $\mathtt{VK}_{TRK}$
- secure counter
- key management of authorized $\mathtt{VK}_{TRK}$

SP possesses cryptographic primitives for PKC based authentication schemes on prover's and verifier's side and can thereby be authenticated by communication partners as well as authenticate communication partners.

*6) SPE - Employee of Service Point:* A SPE is a physical person that is operating the SP and is regarded as a potential attacker of the flashing operation. Access control of a SPE to the 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, $\mathtt{TRK}_{new}$, and SP are under control of the SPE and the communication links to UD, $\mathtt{TRK}_{orig}$, $\mathtt{TRK}_{new}$, SP, and OEM can be eavesdropped, the trusted module can not be penetrated though.

### B. Steps

The following steps are necessary to introduce an new $\mathtt{VK}_{TRK}$ into a UD avoiding online dependency. All of them are included in figure 1.

1) Delegation of trust to SP
2) Authorization of SPE by SP
3) Authorization of $\mathtt{TRK}_{new}$ by OEM
4) Introducing an authorized $\mathtt{TRK}_{new}$ into a UD

Authorization of SPE can be done e.g. via a password (knowledge) or by biometrical identification (physical property). The delegation of trust and authorization of $\mathtt{TRK}_{new}$s is very closely related and described in section V-C. 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 final introduction of a new $\mathtt{VK}_{TRK}$ is the second phase and is detailed in section V-D. It does no longer depend on interaction with OEM.

### C. Trust Delegation and $\mathtt{TRK}_{new}$ 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

[2] TRKs can be manufactured by a supplier that has been certified by the OEM

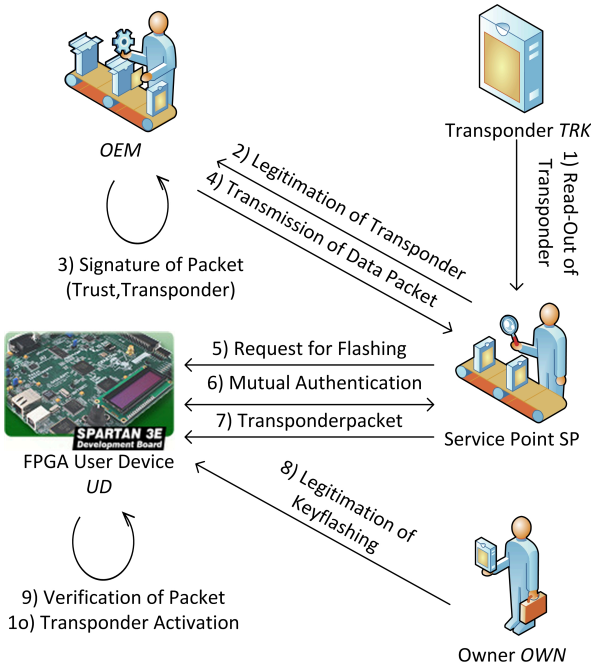


Fig. 1. Flashing Scheme

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*.

In our case SP is the local representative. In order to request the flashing credential from the OEM, SPE has to be authenticated first to prevent SP abusive operations. Afterwards SP can connect to OEM and request a trust credential. This is issued only after mutual authentication and only to known partner service points. It is always valid for only a limited time and limited number of flashing operations to minimize negative impact of compromised SPs. This is controlled and enforced by the trusted component inside SP using the secure unforgeable counter keeping track of the number of flashing cycles.

The public key of a $\mathrm{TRK}_{new}$ needs to be authorized by OEM. SP can read out $\mathrm{VK}_{TRK}$ and send it to OEM. If SP is allowed to flash TRKs into a UD, the OEM sends the authorized $\mathrm{VK}_{TRK}$ back to SP which is stored in SP's 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 for a data connection between SP and OEM. This does not necessarily need to be an online connection since data could be transported via data carriers such as CDs, memory sticks, or the like.

### D. Flashing of TRK

The actual flashing of a $\mathrm{TRK}_{new}$ to a given UD demands for a valid new transponder $\mathrm{TRK}_{new}$, authorization by OEM and OWN, former either directly or delegated to SP using the credential introduced above, latter done by presenting a valid linked $\mathrm{TRK}_{orig}$ 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, SP only relaying communication.

In either case UD and SP authenticate each other mutually using their respective OEM-issued certificates. UD additionally checks authorization by OWN, testing whether a valid linked token is present or not. If all these tests passed, SP presents the authorized and OEM-signed new $\mathrm{TRK}_{new}$ to UD which checks the OEM signature and credential. In the case of successful verification UD accepts the new token $\mathrm{TRK}_{new}$ and adds $\mathrm{VK}_{TRK}$ to it's internal list of linked tokens.

### E. Entity Requirements

Regarding the proposed flashing protocols certain requirements for the entities' functionalities have to be satisfied. An overview is given in table I

| | OEM | SP | UD | TRK |
|---|---|---|---|---|
| Initiate Communication | • | • | | |
| Acknowledge Communication | • | | • | • |
| Generation of Keypairs | • | | | • |
| Signatures Generation | • | • | • | • |
| Signature Verification | • | • | • | |
| Random Number Generation | • | • | • | |
| 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 | | • | • | |

TABLE I
ENTITY REQUIREMENTS

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 entity and must not be altered in any case, otherwise the OEM can not be identified correctly from the protocol's point of view.

## VI. Implementation

The protocol has been implemented as a proof of concept in a prototypical setup based on a network of a standard PC representing OEM and SP. Furthermore Digilent *Spartan3E Starter Boards* with a Xilinx XC3S500 FPGA represent TRKs and UDs.

In figure 2 all implemented instances are depicted. 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 with the SP.

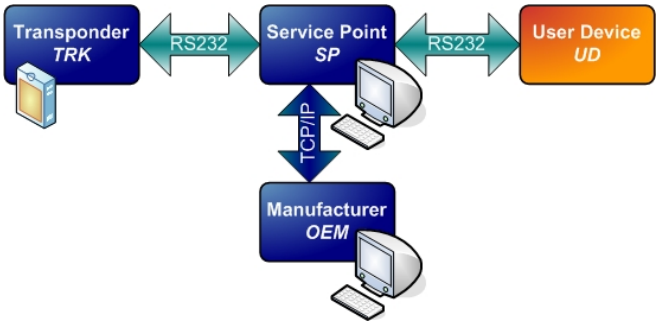


Fig. 2. Component Interaction

| 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 |

TABLE II
PARAMETERS FOR RSA-SYSTEM

| 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 III
PROPERTIES OF OEM COMPONENT

| 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 IV
FPGA RESOURCES

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

### A. Choice of cryptographic Algorithms

The proposed keyflashing concept demands for asymmetric encryption and a cryptographic hashfunction. RSA [15] is chosen for encryption and signing, SHA1 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 if needed. RSA as well as SHA1 implementations are freely available as software and hardware IP for numerous platforms. In table II the RSA parameters chosen are given.

All signatures in our context are SHA1-hash values of data that has been encrypted according to the signing scheme PKCS#1 v1.5 [14]. Such a signature has a length of 128 Byte when using a keylength of 1024 bit and hashvalues of 160 bit bitlength.

### B. OEM/Service Point - Software Platform

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

### C. Transponder/UserDevice - FPGA platform

The targeted user device is an FPGA. To enable for easy 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.

A MicroBlaze softcore processor is incorporated that provides all functionality including cryptographic functions. Hardware peripherals such as a LCD controller have been integrated for debugging purposes. To enable for handling of big numbers, as are used in the cryptographic functions of the protocol, the libraries `libtommath` [5] and `libtomcrypt` [4] are used. Only necessary components have been extracted from those libraries and are integrated into `TRK` and `UD`.

### D. 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 III gives an exemplary overview of the lines of codes 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 IV. 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 non-optimized memory usage of the crypto library used.

Shown in table V are the execution times of the divers protocol instances. The duration of parts of the protocol that are

| Protocol instance | Duration (min:sek.ms) |
|---|---|
| ReadOut of Transponder | 01:32.000 |
| Mutual Authentication of UD und 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 |

TABLE V
PROTOCOL EXECUTION TIMES

based soley 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 microcontroller. Main factor here is the RSA decryption operation. With appropriate hardware support, choice of parameters and cryptosystem, substantial speedups can be achieved as shown in [9].

## VII. CONCLUSIONS AND FUTURE WORK

In this paper we proposed a scheme for flashing public keys into mobile FPGA devices under the constraint that no online connection to the system manufacturer (OEM) is mandatory. It is applicable for a variety of embedded systems that need to implement and enforce access or usage restrictions in the field. The scheme was implemented as a proof-of-concept using a combination of PC-based and FGPA-based protocol participants.

### A. 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. But 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 authorized $\mathrm{TRK}_{orig}$.

If these assumptions are broken e.g. 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.

### B. Future Work

Flashing speed is of utmost importance in real world implementation. To make allowance for a real world integration of the proposed flashing schemes, optimizations regarding usage and speed of the computational units involved are needed. In the current prototype the MicroBlaze processor has been used for simplicity. Speed up can be achieved with a hardware/software codesign as done in [10]. For maximal speed a full FPGA hardware implementation is desirable, as has been done in [13] for cryptographic functionalities of a HECC system.

The user authentication via PKC can be a solution for dedicated function enabling. Different functionalities can be configured onto an FPGA using partial dynamic reconfiguration. By either allowing or prohibiting, the configuration of a certain bitstream depending on the user employing the system, usage policies could be enforded thus opening up new business models for suppliers of FPGA based systems.

One crucial point is the protection of the TRK's public key stored in the UD against physical attackers. The possibility of countermeasuring attacks that might alter stored keys on a physical level needs to be investigated in the future as well.

## REFERENCES


[1] Andr Adelsbach, Ulrich Huber, and Ahmad-Reza Sadeghi. Secure software delivery and installation in embedded systems. In Robert H. Deng, editor, *ISPEC 2005*, volume 3439 of *LNCS*, pages 255–267. Springer, 2005.

[2] Hilti Corporation. Electronic theft protection. Available electronically at www.hilti.com, 2007.

[3] Gerrit de Boer, Peter Engel, and Werner Praefcke. Generic remote software update for vehicle ecus using a telematics device as a gateway. *Advanced Microsystems for Automotive Applications*, pages 371–380, 2005.

[4] Tom St. Denis. Libtomcrypt. http://libtomcrypt.com/.

[5] Tom St. Denis. Libtommath. http://math.libtomcrypt.com/.

[6] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.

[7] Herstellerinitiative Software (HIS). *HIS-Presentation 2004-05*, 2005. available electronically at www.automotive-his.de.

[8] Herstellerinitiative Software (HIS). *HIS Security Module Specification v1.1*, 2006. available electronically at www.automotive-his.de.

[9] Alexander Klimm, Oliver Sander, and Jurgen Becker. A microblaze specific co-processor for real-time hyperelliptic curve cryptography on xilinx fpgas. *Parallel and Distributed Processing Symposium, International*, 0:1–8, 2009.

[10] Alexander Klimm, Oliver Sander, Jürgen Becker, and Sylvain Subileau. A hardware/software codesign of a co-processor for real-time hyperelliptic curve cryptography on a spartan3 fpga. In Uwe Brinkschulte, Theo Ungerer, Christian Hochberger, and Rainer G. Spallek, editors, *ARCS*, volume 4934 of *Lecture Notes in Computer Science*, pages 188–201. Springer, 2008.

[11] Roger M. Needham and Michael D. Schroeder. Using encryption for authentication in large networks of computers. *Commun. ACM*, 21(12):993–999, December 1978.

[12] Tatsuaki Okamoto. Provably secure and practical identification schemes and corresponding signature schemes. In *CRYPTO '92: Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology*, pages 31–53, London, UK, 1993. Springer-Verlag.

[13] J. Pelzl, T. Wollinger, and C. Paar. *Embedded Cryptographic Hardware: Design and Security*, chapter Special Hyperelliptic Curve Cryptosystems of Genus Two: Efficient Arithmetic and Fast Implementation. Nova Science Publishers, NY, USA, 2004. editor Nadia Nedjah.

[14] RSA Laboratories Inc: RSA Cryptograpy Standard PKCS No.1. Elektronisch verfügbar unter http://www.rsasecurity.com/rsalabs.

[15] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[16] Claus P. Schnorr. Efficient identification and signatures for smart cards. In *CRYPTO '89: Proceedings on Advances in cryptology*, pages 239–252, New York, NY, USA, 1989. Springer-Verlag New York, Inc.

[17] Henning Wallentowitz and Konrad Reif, editors. *Handbuch Kraftfahrzeugelektronik: Grundlagen, Komponenten, Systeme, Anwendungen*. Vieweg, Wiesbaden, 2006.