Dynamic Intellectual Property Protection for Reconfigurable Devices

Tim Güneysu, Bodo Möller, Christof Paar
Horst Görtz Institute for IT Security, Ruhr-Universitt Bochum, Germany
{gueneysu,bmoeller,cpaar}@crypto.rub.de

Abstract

The distinct advantage of SRAM-based Field Programmable Gate Arrays (FPGA) is their flexibility for configuration changes. However, this opens up the threat of theft of Intellectual Property (IP) since the system configuration is stored in easy-to-access Flash memory. To prevent this, high-end FPGAs have already been extended with symmetric-key decryption engines used to load an encrypted version of the configuration that cannot simply be copied and used without knowledge of the secret key. However, such protection systems based on straightforward use of symmetric cryptography are not well-suited with respect to business and licensing processes, since they are lacking a convenient scheme for key transport and installation.

We propose a new protection scheme for the IP of circuits in configuration bit files that provides a significant improvement to the current unsatisfying situation. It uses both public-key and symmetric cryptography, but does not burden FPGAs with the usual overhead of public-key cryptography: While it needs hard-wired symmetric cryptography, the public-key functionality is moved into a temporary configuration bit stream for a one-time setup procedure. This approach requires only very few modifications to current FPGA technology. Using five basic stages, the new protection scheme allows new accounting models for volume licensing of IP, with automated key installation on FPGAs taking place at the customer's site.

Keywords: IP protection, secure configuration, FPGA, embedded security

1 Introduction

When Field Programmable Gate Arrays (FPGA) were first introduced in the 1980s, this was a revolutionary step from static ASIC and VLSI solutions to flexible and maintainable hardware applications. It has become possible to avoid the static designs of standard VLSI technology, and instead to compile electrical circuits for arbitrary hardware functions into configuration bit files used to program a fabric of reconfigurable logic. A new market has evolved where companies have specialized on the development of abstract hardware functions that can be distributed and licensed to system integrators by using only a logical description file. However, the flexibility of SRAM-based FPGAs also brings up the issue of protecting the Intellectual Property (IP) of such circuit layouts from unauthorized duplication or reverse engineering. Unfortunately, a configuration bit file of an FPGA can easily be retrieved from a product and used to clone a device with only little effort. Furthermore, IP vendors that deliver configuration bit files to licensees do not have any control over how many times the IP is actually used.

Previous Work. To cope with these problems, various approaches have been proposed. A simple "security by obscurity" approach is to split the IP among multiple FPGAs and create a unique timing relationship between the components [4]. This type of mechanism, however, will not protect proprietary IP from more intensive attacks. Moreover, such techniques force IP vendors selling only bit files to deal with the customer's board layout as well.

In a smarter approach, IP vendors can insist on installing their configuration bit file only on encryption-enabled FPGA devices using a previously inserted secret key. Common high-end FPGA types like Virtex 2, Virtex 4 and Virtex 5 from Xilinx [25] as well as Altera's Stratix II GX and Stratix III [2] devices provide decryption cores based on symmetric 3DES and AES blockciphers. With an encrypted configuration file, the IP can only be used on a device that has knowledge of the appropriate secret key. But here the issue of key transfer arises. One approach is to ship FPGAs to the IP owner for key installation: The IP owner installs secret keys in the devices such that these keys are available to decrypt configuration bit streams but remain otherwise unaccessible; after key installation, the devices are returned to the customer. The high logistical effort makes this a very unsatisfying solution to the problem.

Further solutions are based on separate security chips that dongle the IP to a specific FPGA by exchanging cryptographic handshaking tokens between the components [1]. Similarly, this approach requires modification

1-4244-1472-5/07/$25.00 © 2007 IEEE

to the customer's board layout, additional hardware, and a secure domain for installing the secret parameters; so it provides only a partial solution at a high cost.

In the literature, there are only very few suggestions to enhance this situation. In [11, 12], a strategy was proposed based on a static secret key already inserted during the manufacturing process. The issue of key transfer is solved by including cores both for encryption and for decryption in the FPGA: an FPGA specimen containing the appropriate key can be used to encrypt a configuration file, yielding a configuration that will work for itself and for other FPGAs sharing the same fixed key. In [13, 22], more complex protocols have been proposed for a more complete solution. Both approaches require the implementation of additional security features in the FPGA. They also require the participation of the FPGA manufacturer (as a trusted party) whenever a bit stream is to be encrypted for a particular FPGA, meaning that such transactions cannot be kept just between the IP vendor and the customer.

Our Contribution. In this paper, we propose a new protection scheme for configuration bit files that provides IP vendors with means for exact tracking and control of their licensed designs without any need for additional hardware components or major modifications of recent FPGA technology. Instead of demanding a crypto component for key establishment in the static logic as needed by [13, 22], we use the reconfigurable logic for a setup process based on public key cryptography. Besides exact tracking of the number of licensed IP, our solution provides the invaluable advantage of off-site IP installation: The installation of the designs can be performed by the licensees without any shipping of hardware; our approach does not require the continuing participation of a third party (such as the FPGA manufacturer) either. To enable FPGAs for these new features, only marginal modification are required on recently available FPGA models.

Outline. Our new protection scheme is described in the following section. This includes assumptions and requirements as well as the description of the protocol and its components. In Section 3, we describe security aspects of our proposal with respect to practical attacker models and objectives. Section 4 discusses implementation aspects and the impact of necessary changes on current FPGA types. Finally, we conclude with an outlook for further research options.

2 Protection Scheme

In this section, we introduce a new protection scheme for the IP in configuration files for FPGAs.

2.1 Participating Parties

Our idea assumes a use case with three participating business parties. The first contributor is the Hardware Manufacturer (HM), who designs and produces FPGA devices. A second participant is the Intellectual Property Owner (IPO), who has created some novel logic design for a specific problem. This IP is synthesized as a configuration bit file for a specific class of FGPAs manufactured and provided by the HM. The IPO wants to distribute the configuration bit file using a special cost or licensing model, usually on a per-volume basis. The final participant is a System Integrator (SI), who intends to use the IPO's design in products employing the HM's FPGA devices. To support a volume licensing model, we want to allow the IPO to limit the number of FPGAs that can use the design.

2.2 Cryptographic Primitives

For a protection scheme that is not just based on obscurity, we need to use cryptography to achieve security.

Symmetric Cryptography: For design protection and configuration confidentiality, some FPGAs already implement symmetric encryption based on well-known blockciphers like AES and 3DES [2, 25]. Such cryptographic functionality can actually be used for more than just confidentiality: In the CMAC construction [19], a computational process mostly identical to that of CBC encryption with a blockcipher is used to generate a one-block message authentication code (MAC). Thus, to keep the footprint of the crypto component small, an implementation of a single blockcipher can be used both for decryption and for MAC verification, using a CBC scheme in both cases. Using separate blockcipher keys, a combination of such a MAC with encryption can achieve *authenticated encryption* [5]: we simply have to provide a MAC of the CBC-encrypted ciphertext. (This will provide security in the sense of the IND-CCA and INT-CTXT notions as explained in [5]; thus a party that does not know the keys cannot hope to derive a new ciphertext that would be accepted as valid.) We can consider the pair of such keys a single (longer) key k. Throughout this paper, we will write $enc_k(x)$ for authenticated encryption of a value x (the plaintext) yielding a ciphertext including a MAC value, and $dec_k(y)$ for the reverse step of decryption of a ciphertext y while also checking for an authentication error based on the MAC value provided as part of the ciphertext.

The use of CBC for confidentiality with CMAC for authentication is just an example of a convenient scheme. Alternatively, we could use any suitable symmetric cryptographic scheme that provides authenticated encryption in an appropriate sense. As an implemen-

tation note regarding the particular combined scheme using CBC with CMAC, note that an implementation of either blockcipher encryption or blockcipher decryption is sufficient in the FPGA: we can arrange to use the blockcipher "in reverse" for one of the two cryptographic steps, e.g. use a CMAC based on blockcipher decryption rather than on blockcipher encryption.

Asymmetric Cryptography: If we want to use symmetric data encryption, this means we have the issue of establishing a shared key k between the parties (usually over an untrusted communication channel). With asymmetric (or public-key) cryptography, a pair of keys consisting of a public (PK) and a private (SK) component is used to overcome the key transport deficiencies of symmetric methods, at the cost of higher system and computation complexity. The first publicly known example of public-key cryptography was the Diffie-Hellman (DH) scheme [9], which can be used to establish keys for symmetric cryptography. Appropriately used in conjunction with a key derivation function (KDF) based on a cryptographic hash function, the DH scheme remains state of the art. An important variant of this is the DH scheme using elliptic curve cryptography [8], ECDH. See [20] for elaborate recommendations on the proper use of DH and ECDH.

Public-key cryptography can also be used for authentication through digital signatures. These authentication features are beneficial for our purposes, although their exact use in authenticated communication is outside of the scope of the present paper.

2.3 Key Establishment

For the protocol interactions, we define the set of parties as

$$Z = \{\mathrm{HM}, \mathrm{IPO}, \mathrm{SI}, \mathrm{FPGA}\},$$

which corresponds to the participants introduced in Subsection 2.1 plus any specific FPGA device, here considered a party in its own right. A key for symmetric cryptography chosen by party $z \in Z$ is denoted K_z. Keys for asymmetric cryptography are given in pairs (PK_z, SK_z) where PK_z is the public key component (which may be known by anyone), and SK_z is the private, or secret, component (which generally should only be known by z, but may sometimes be shared with specific other parties).

A key establishment scheme based on Diffie-Hellman (including the ECDH variant for elliptic curve cryptography) is described in detail in [20]. Given any two parties' public and private keys $PK_z, SK_z, PK_{z'}, SK_{z'}$ and an additional bit string *OtherInfo*, where the keys are understood to have been generated based on common domain parameters, these parties can determine symmetric key material by evaluating $key(PK_z, SK_{z'}, OtherInfo)$ and $key(PK_{z'}, SK_z, OtherInfo)$ where *key* is a function combining the basic DH primitive (possibly in the ECDH variant) with a key derivation function (KDF). Each party uses the other party's public key and its own secret key as input to the Diffie-Hellman primitive, which then will yield identical intermediate results for both parties. To get the final output, the KDF is applied to the given inputs: by varying the *OtherInfo* value (which directly becomes part of the KDF input), static-key Diffie-Hellman can be used to generate many seemingly independent symmetric keys. (The recommendations in [20] for static-key Diffie-Hellman settings additionally require the use of a nonce in the KDF input for each invocation of the key establishment scheme. This use of non-repeated random values ensures that different results can be obtained based on otherwise identical inputs. However, we do not need this nonce here: for our application, the reproducibility of key establishment results is a feature, not a deficiency.)

2.4 Prerequisites and Assumptions

For realizing our protection scheme, we assume the following prerequisites (P).

P1. **Trusted Party.** We assume the FPGA hardware manufacturer (HM) as a commonly trusted party. All other participating and non-involved parties in the protocol can assume the HM to be trustworthy and unbiased, i.e., the HM will neither share any secrets with other parties nor unfairly act in favor of someone else. All other parties, however, are regarded per se as untrusted, and may try to cheat.

P2. **Secure Cryptography.** Furthermore, we assume that all cryptographic primitives to be computationally secure, i.e., no attacker with a practical amount of computational power will likely be able to compute any secret keys or otherwise break the cryptography in any reasonable amount of time. This includes the symmetric and asymmetric cryptographic primitives as well as the auxiliary cryptographic hash function. Moreover, implementations used for the protocol are assumed to be fault-tolerant, tamper-resistant (cf. [21]) and to remain secure against side-channel attacks over multiple executions, i.e., it must not be possible to learn any further information concerning a secret key by analyzing observations from multiple independent protocol runs.

P3. **FPGA Security Environment.** For the scheme, we assume an FPGA with an integrated symmetric decryption engine that is able to handle encrypted

configuration files. This reference device is extended with the following features:

(i) A unique device identifier ID (an l-bit value) is assigned by the hardware manufacturer (HM), accessible from the fabric.

(ii) A symmetric key K_{HM} (an m-bit value) that is statically integrated by the HM during the manufacturing process, and which can only be read by the internal decryption engine, not from the FPGA fabric.

(iii) A symmetric key store K_{FPGA} (also an m-bit value) that is implemented as non-volatile memory and allows for storing a variable key. The key stored in K_{FPGA} can either be updated using an external interface (e.g., JTAG, SelectMap) or via an internal port from the reconfigurable logic of the FPGA; however, it can only be read from the internal decryption engine (not from the fabric).

(iv) A data exchange register that can be accessed via a standardized configuration interface like JTAG as well as from the reconfigurable fabric using a dual-ported logic. This feature is already available on many common FPGAs based on user-defined instructions in the JTAG protocol.

(v) Tamper-resistant control and security components that can withstand invasive and non-invasive attacks on the device. Particular protection mechanisms should cover the integrated keys K_{HM} and K_{FPGA}, the decryption engine and the FPGA controller distributing the unencrypted configuration bits in the fabric of the FPGA after decryption. Hence, a readback of the plain configuration bits or partial reconfiguration must not be possible on a device with an encrypted configuration loaded. Possible physical threats to security-enabled devices as well as side-channel attacks have been discussed more thoroughly in [16, 7, 24].

We assume that the decryption engine can be invoked such that, at the user's choice, either K_{HM} or K_{FPGA} is employed to decrypt a configuration file for subsequent use.

2.5 Steps for IP-Protection

This section covers the steps performed by each party according to an ideal protection protocol, i.e., assuming that all parties behave as desired without any fraud attempts or protocol abuse. We have five main stages in our scheme (described in detail further below):

A. **SETUP.** On the launch of a new class of FPGA devices, the HM creates a specific bit stream for them, a *Personalization Module* (PM) that later will be used in the personalization stage. An encrypted version of this PM is made available to all participants, together with a public key associated to it.

B. **LICENSING.** When an IPO offers licenses to an SI, it provides a public key of its own. The device identifier ID of each FPGA on which the SI intends to use the licensed IP is transferred to the IPO.

C. **PERSONALIZATION.** A personalization stage is performed for each FPGA device in the domain of the SI. The PM is used to install a device-specific key in each FPGA on which it is executed.

D. **CONFIGURATION.** Using the device information, the IPO sends copies of the configuration file containing its IP to the SI, specifically encrypted for each FPGA.

E. **INSTALLATION.** The SI installs the IP in each FPGA (using the appropriate encrypted copy).

The information exchange between the parties is simple. Figure 1 shows the data flow between the participants. Steps 1 through 3 can be considered one-time operations (these are part of the setup and licensing stage); steps 4 and 5 (part of the licensing and configuration stage) are required to be performed repetitively for each FPGA that should make use of the protected IP.

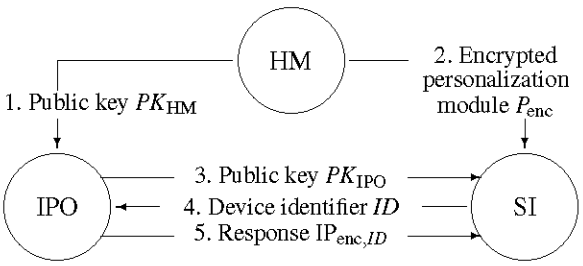

Figure 1. Data flow between parties

2.5.1 SETUP Stage (A)

The setup stage, described in the following, is performed once by the HM for each FPGA class to be enabled for the proposed security features. Note that it is reasonable to artificially keep each FPGA class relatively small by splitting a large series of otherwise identical FPGAs into

multiple separate classes. This limits the potential damage done if an FPGA's permanent secret K_{HM} or the secret SK_{HM} is compromised (cf. Section 2.4). Any such class should be treated as independent from any other class; i.e., the results of the setup stage as described below cannot be shared between classes. For practical reasons, a specific part of the device ID should be used to denote the class that a given FPGA belongs to.

A1. The HM generates a symmetric key K_{HM} and an asymmetric key pair (SK_{HM}, PK_{HM}) for key establishment.

A2. The HM creates a specific bit stream P for the FPGA class such that P implements a key establishment scheme, as described below in 2.5.3. P includes the private key SK_{HM}. All components employed should be fault-tolerant and include countermeasures against external tampering [16]. The bit stream P acts as a personalization module and implements the FPGA behavior that we will present in Subsection 2.5.3.

A3. After the configuration bit file P has been built, it is encrypted using the secret key K_{HM}, yielding the encrypted configuration file $P_{enc} = enc_{K_{HM}}(P)$.

A4. The secret key K_{HM} is statically integrated in each FPGA (cf. Subsection 2.4) during the manufacturing process.

After these setup steps have been completed, the HM distributes the encrypted personalization bit stream P_{enc} and the public key component PK_{HM} to all participating parties.

The other parties (notably IPO) must convince themselves that PK_{HM} actually originates from HM. How this authentication is performed in detail is outside of the scope of the present paper; however, we remark that if HM supports many different FPGA classes, then a public-key infrastructure (PKI) can be very useful.

2.5.2 LICENSING Stage (B)

The licensing stage can be regarded as a first interaction between IPO and SI. To use external IP, the SI signs a contract with the IPO (usually for a volume license). Then, the following steps are required.

B1. The IPO creates a key pair (SK_{IPO}, PK_{IPO}) and sends the public component PK_{IPO} to the SI.

B2. SI provides IPO with a list of the ID values of those FPGAs for which the SI intends to obtain a license.

Again, authentication of the communication between IPO and SI is not explicitly covered by this paper since well-known solutions do exist (e.g., digital signatures). It should be remarked that authentication is required to avoid any abuse of the license model; again, a PKI, can be very useful.

2.5.3 PERSONALIZATION Stage (C)

In contrast to current solutions, our scheme allows for a key installation that is done in the untrusted domain of the SI, thanks to the secret key K_{HM} available in the specific FPGAs from the setup stage. The following steps need to be performed for every FPGA intended to be operated with the IP from the IPO. They can be automated very efficiently. The personalization and the key establishment within an FPGA makes use of the encrypted configuration P_{enc} performing the (one-time) key setup in logic residing in the fabric. Compared with the option of demanding static security components for this step in the static part of an FPGA, this provides huge efficiency advantages since it limits the resources that have to be permanently allocated in the FPGA device.

C1. Using a programming interface, the FPGA is configured with the encrypted personalization module P_{enc} made available by the HM, which is decrypted using the statically integrated key K_{HM}.

C2. Then, the data exchange register of the FPGA is loaded with the public key PK_{IPO} via a common interface (e.g., JTAG), thus initiating the computation process.

C3. The personalization module that has now been loaded determines a symmetric key $key(PK_{IPO}, SK_{HM}, ID)$ using the integrated key agreement scheme, and stores the resulting symmetric key in K_{FPGA}. From now on, the FPGA can decrypt designs that are encrypted using this key.

The security properties of the key establishment scheme imply that knowledge of either SK_{HM} or SK_{IPO} is required to compute the key now stored in K_{FPGA}; thus, SI cannot determine K_{FPGA}. Including ID in the KDF input ensures that K_{FPGA} will differ for different FPGA instances. For further implementation aspects, see Section 4.

2.5.4 CONFIGURATION Stage (D)

For each FPGA ID for which SI has bought a license, the IPO returns a corresponding configuration file to the SI usable only on the specific FPGA. This mechanism allows the IPO to easily track the number of FPGAs that

have been configured to use the licensed IP. In detail, the IPO performs the following steps to generate the FPGA-specific configuration file.

D1. The IPO recovers the specific key K_{FPGA} using its own secret key and the HM's public key:

$$K_{\text{FPGA}} = key(PK_{\text{HM}}, SK_{\text{IPO}}, ID)$$

D2. The IPO encrypts the plain IP configuration bit file using the secret key, thus binding the IP to a specific FPGA device:

$$\text{IP}_{\text{enc},ID} = enc_{K_{\text{FPGA}}}(\text{IP}).$$

IPO then transmits this encrypted file to SI.

By the properties of the key establishment scheme, we have $key(PK_{\text{IPO}}, SK_{\text{HM}}, ID) = key(PK_{\text{HM}}, SK_{\text{IPO}}, ID)$, so they key K_{FPGA} as determined by the IPO is identical to the key K_{FPGA} as determined by the FPGA in the personalization stage.

2.5.5 INSTALLATION Stage (E)

After having received $\text{IP}_{\text{enc},ID}$, the SI configures the FPGA with the personalized IP.

E1. SI configures the flash memory of the specific FPGA denoted by identifier ID with $\text{IP}_{\text{enc},ID}$ to operate the device.

Since K_{FPGA} is available in the FPGA from the personalization stage, this step enables the FPGA to use the IPO's configuration bit file IP by computing $dec_{K_{\text{FPGA}}}(\text{IP}_{\text{enc},ID})$.

3 Security Aspects

We assumed the implementations of cryptographic elements integrated in the personalization module to be fault-tolerant and tamper-resistant. This is mandatory since an attacker might take physical influence on the device while security-relevant processes are performed. Hence, countermeasures against device tampering are required to check if the module is executed under unconventional conditions, e.g, over-voltage, increased temperature or clock frequency. A fairly easy approach to detect operational faults caused by such conditions is to use multiple, identical cryptographic components in the personalization module operated at different clocks or clock shifts. When all computations are finished, all results are compared which will detect any injected faults and operational irregularities. With the option to integrate a multitude of countermeasures [3, 16], we are willing to rely on the FPGA behaving as specified. Then, the protocol is a rather straightforward application of well-known cryptographic mechanisms: symmetric encryption in the sense of authenticated encryption, and a key establishment scheme.

Only for authenticated encryption, we require a hardwired implementation within the FPGA device. As sketched in Section 2.2, this can be done based on a single blockcipher such as AES or 3DES. Authenticated encryption ensures integrity of configuration bit strings, i.e., a user cannot modify the encrypted configuration to obtain another (related) configuration that would be accepted by FPGA device and modify part of the intended behavior, which would be the case for unauthenticated encryption (such as mere CBC-mode encryption using AES or 3DES).

Finally, we want to emphasize that special care should be taken for the realization of the personalization module because of its exposed application in an untrusted domain. Luckily, such a fault-tolerant and tamper-resilient design has to be developed only once and should be easily portable between different FPGA classes.

4 Implementation Aspects

We will give some brief suggestions and implementation details how to realize the participating components, and discuss their expected system costs.

4.1 Implementing the Personalization Module

In the following we will demonstrate the feasibility of the personalization module that incorporates the implementation of a key establishment scheme as specified by the protection scheme in Section 2.

Several public-key algorithms have successfully been implemented in FPGAs. Low footprint RSA/Diffie-Hellman implementations have been proposed in [10, 14, 18] and are already available in ready-to-use IP cores by FPGA vendors like Altera.

Taking the smallest Virtex-4 XC4VFX12 FPGA with an integrated AES-256 bit stream decryption core as a reference device, we have implemented the key establishment scheme for the personalization module. For this proof-of-concept implementation, we realized an ECC core over prime fields, which forms the basis for an Elliptic Curve Diffie-Hellman (ECDH). Parameters for this implementation have been chosen by a trade-off of long-term security and efficiency: We designed an ECDH component over $GF(p)$ and $\log_2 p = 160$ bits, which is sufficient for mid-term security nowadays. For details, Table 1 presents requirements of logical elements and data throughput of our implementations.

For the implementation of the KDF according to [20], we integrated an implementation of the SHA-1 as cryp-

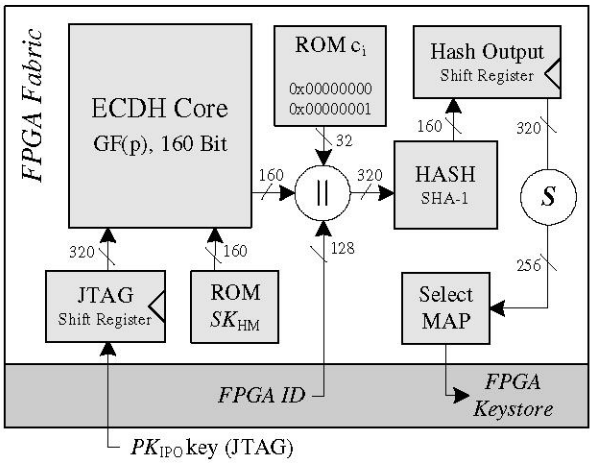

Figure 2. Simplified schematic of a personalization module for Xilinx Virtex-4 FPGAs with a 256-bit decryption key

tographic hash function $h(x)$ with an output bit length of 160 bits. During key generation, the SHA-1 hash function is executed twice using three different inputs: Given two 32-bit counter values (effectively, two constants) c_0, c_1, a 128-bit FPGA ID and the ECDH result E, a 256-bit AES key K_{FPGA} can be derived as follows:

$$\begin{aligned} H_i &= h(c_i \,\|\, E \,\|\, ID) \text{ for } i \in \{0,1\} \\ K_{\text{FPGA}} &= S(H_0 \,\|\, H_1), \end{aligned}$$

where $S(x)$ is a selection function choosing the first 256 out of 320 input bits and where $\|$ denotes concatenation. The data exchange between the personalization module and an external party (SI) was realized using a shift register which is writable from the JTAG interface. Beside a ROM for storing the secret key of the HM and constants c_0, c_1, a SelectMAP controller is required to program the key storage of the FPGA, which was provided by Berkeley's Bee2 project [6]. The schematic of the implemented personalization module is sketched in Figure 2.

For this proof-of-concept work, the SelectMAP core needs to be externally connected with the FPGA's SelectMAP interface since a direct configuration from the fabric is not yet available. It should be remarked that all implementations have been developed for an optimal area-time product so that reductions in logical elements can still be achieved if data throughput is not a primary issue. Concluding, the implementations at hand are small enough to fit even the smallest Virtex-4 XC4VFX12 device (providing a total of 5472 slices of which less than 4000 are required) with some remaining logical resources to add functionality providing tamper resistance and fault tolerance.

Component	Logical Elements	Data Throughput
ECDH Core	2706 slices	186Kbit/s
SHA-1 Core	716 slices	730Mbit/s
ROM	112 slices	—
JTAG Register	160 slices	—
SelectMAP Core	159 slices	—

Table 1. Data throughput and logic requirement of personalization components on a Xilinx Virtex-4 FPGA

4.2 Additional FPGA Features

To use our proposed key scheme on common FPGA devices, an additional key storage with a fixed key needs to be added to the static design. This is no technical and big financial issue since it can be integrated by the HM either directly in the circuit layout, or using antifuses, or by using similar techniques in a post-production step.

Such a strategy based on laser inscriptions or antifuses can also be used to provide each FPGA with a unique identification number. Alternatively, so called Physically Unclonable Functions (PUF) implemented using a coating or delay technique might be an option to create a unique identification of each chip [17].

A further additional requirement for our scheme is access from the fabric to the key store via an internal (write-only) interface. This is obviously no problem since it only requires some internal component repositioning and some dedicated internal I/O pins.

Moreover, for advanced bit stream protection in the FPGA, we require the decryption direction of authenticated encryption as explained in Section 2.2. A single blockcipher engine can be used both for authentication and for decryption, and such reuse means that only little modification are needed for current FPGA architectures already containing a symmetric blockcipher engine.

To achieve tamper resistance against invasive attackers, most HMs have already taken efforts to hide away critical parts from observation and manipulation, i.e., by distracting those components over multiple layers and locations. Moreover, strategies known from smart cards [16] could be applied to strengthen FPGAs against physical and invasive attacks.

5 Conclusions and Outlook

We proposed a new protection scheme for IP of circuits for FPGAs that provides a significant improvement to the recent unsatisfying situation, with only very few modifications to current FPGA devices and corresponding infrastructures. Due to a resource-preserving per-

sonalization module temporarily located in the reconfigurable logic, for a one-time key-establishment, the new protection scheme is suitable for nearly every modern FPGA type with just minor modifications to the function set and architecture.

An open issue is the protection of partial designs, e.g., functional cores that are used as a subcomponent in a configuration of an FPGA. One could make use of a protection scheme with multiple keys K^*_{FPGA} per FPGA, where protected subdesigns are loaded into well-defined fabric areas of the FPGA using a feature of partial reconfiguration.

Instead of integrating a static ID into each FPGA, the HM might use a Random Number Generator (RNG) that is additionally included in the personalization module. This RNG can generate a random value taken as input to the KDF and used to bind an encrypted configuration $\text{IP}_{\text{enc},RN}$ to a specific FPGA. Implementations for cryptographically secure RNGs in FPGAs are already available [23, 15]. The RNG implemented in the reconfigurable logic might be an additional option to reduce the number of modifications with respect to current FPGA architectures to implement our protection scheme.

References

[1] Altera Corp. FPGA design security using MAX II reference design. http://www.altera.com/end-markets/refdesigns/sys-sol/indust˙mil/ref-des-secur.html

[2] Altera Corp. Stratix II GX and Stratix III FPGAs, 2006. www.altera.com/products/devices/

[3] R. Anderson, M. Bond, J. Clulow, and S. Skorobogatov. Cryptographic processors – a survey. *Proceedings of the IEEE*, 94(2):357– 369, Feb 2006.

[4] T. Barraza. How to Protect Intellectual Property in FPGA Devices Part II. *Design and Reuse Online: Industry Articles*, 2005. http://www.us.design-reuse.com/articles/article11240.html

[5] M. Bellare and C. Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm.

[6] Berkeley University. Bee2 project. http://bee2.eecs.berkeley.edu

[7] L. Bossuet, G. Gogniat, and W. Burleson. Dynamically configurable security for SRAM FPGA bitstreams. *International Journal of Embedded Systems*, 2(1-2):73–85, 2006.

[8] H. Cohen, G. Frey, R. Avanzi, C. Doche, T. Lange, K. Nguyen, and F. Vercauteren. *Handbook of Elliptic and Hyperelliptic Curve Cryptography*. CRC Press, 2006.

[9] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[10] J. Fry and M. Langhammer. RSA & Public Key Cryptography in FPGAs. Technical report, Altera Corp., 2005.

[11] T. Kean. Secure configuration of field programmable gate arrays. In *In proceeding of 11th International Conference on Field-Programmable Logic and Applications, FPL2001*, Belfast, United Kingdom, 2001.

[12] T. Kean. Secure configuration of field programmable gate arrays. In *IEEE Symposium on Field Programmable Custom Computing Machines (FCCM)*, Rohnert Park CA, 2001.

[13] T. Kean. Cryptographic Rights Management of FPGA Intellectual Property Cores. In *Proceedings ACM Conference on FPGAs*, Monterey, CA, 2002.

[14] Ç. K. Koç. RSA hardware implementation. Technical report TR801, RSA Data Security, Inc., Aug. 1995.

[15] P. Kohlbrenner and K. Gaj. An embedded true random number generator for FPGAs. In R. Tessier and H. Schmit, ed., *FPGA*, pp. 71–78. ACM, 2004.

[16] O. Kömmerling and M. G. Kuhn. Design principles for tamper-resistant smartcard processors. In *Proceedings of the USENIX Workshop on Smartcard Technology*, 1999.

[17] D. Lim. Extracting secret keys from integrated circuits. Master's thesis, 2004.

[18] A. Mazzeo, L. Romano, G. PaoloSaggese, and N. Mazzocca. FPGA-based implementation of a serial RSA processor. *Design, Automation and Test in Europe Conference and Exposition*, pp. 10582–10589, 2003.

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

[20] National Institute of Standards and Technology (NIST). Recommendation for pair-wise key establishment schemes using discrete logarithm cryptography. NIST Special Publication SP 800-56A, 2006.

[21] S. B. Örs, E. Oswald, and B. Preneel. Power-analysis attacks on an FPGA — first experimental results. In *CHES 2003*, LNCS vol. 2779, pp. 35–50, 2003.

[22] E. Simpson and P. Schaumont. Offline hardware/software authentication for reconfigurable platforms. In *CHES 2006*, LNCS vol. 4249, pp. 311–323, 2006.

[23] B. Sunar, W. J. Martin, and D. R. Stinson. A provably secure true random number generator with built-in tolerance to active attacks. *IEEE Tr. Computers*, 56.1:109–119, January 2007.

[24] T. Wollinger and C. Paar. How secure are FPGAs in cryptographic applications, 2003.

[25] Xilinx Corp. Virtex2, Virtex 4 and Virtex 5 FPGAs, 2006. www.xilinx.com/products/silicon˙solutions/