PHYSICAL UNCLONABLE FUNCTIONS AND PUBLIC-KEY CRYPTO FOR FPGA IP PROTECTION

Jorge Guajardo, Sandeep S. Kumar, Geert-Jan Schrijen, Pim Tuyls

Philips Research Laboratories, Eindhoven, The Netherlands
{Jorge.Guajardo,Sandeep.Kumar,Geert.Jan.Schrijen,Pim.Tuyls}@philips.com

ABSTRACT

In recent years, IP protection of FPGA hardware designs has become a requirement for many IP vendors. To this end solutions have been proposed based on the idea of bitstream encryption, symmetric-key primitives, and the use of Physical Unclonable Functions (PUFs). In this paper, we propose new protocols for the IP protection problem on FPGAs based on public-key (PK) cryptography, analyze the advantages and costs of such an approach, and describe a PUF intrinsic to current FPGAs based on SRAM properties. A major advantage of using PK-based protocols is that they do not require the private key stored in the FPGA to leave the device, thus increasing security. This added security comes at the cost of additional hardware resources but it does not cause significant performance degradation.

1. INTRODUCTION

In today's globalized economy, it has become standard business practice to include third party Intellectual Property (IP) into products. This trend has led to the realization that internally developed IP is of strategic importance, for two reasons: (i) it decreases the design cycle by implementing reuse strategies and (ii) it is a source of additional licensing income from external parties. However, IP creators must face the counterfeiting challenge. For example, it is estimated that as much as 10% of all high tech products sold globally are counterfeit. This translates into a conservative estimate of US$100 billion of global IT industry revenue lost due to counterfeiting [1]. The same paper advises to employ anti-counterfeiting technologies to mitigate the effects of counterfeiters. This paper deals explicitly with one such technology and its implementation on FPGAs.

SRAM-based FPGAs, which are the majority of the market, offer a very flexible solution for implementation of valuable designs since they can be reprogrammed (configured) in the field. This allows for instance to update current designs with new and improved ones and stands in sharp contrast with ASIC implementations. Often FPGA designs are represented as bitstreams and stored in external memory *e.g.* PROM or flash. When the FPGA is powered up, the bitstream is loaded onto the FPGA and the FPGA is configured. During loading, an attacker can easily tap the bitstream and make a copy of it, which he can then use to (illegally) program other FPGAs without paying the IP owner's licensing fees. This attack is called a *cloning* attack and it is a serious concern to IP developers nowadays.

From a security perspective, the counterfeiting threat can be best explained as an authentication problem. In particular, we are interested in providing (S1) *hardware IP authentication* (a hardware design runs only on a specific device), (S2) *hardware platform authentication* (the FPGA allows only authentic designs to execute), and (S3) *complete design confidentiality* (only the intended design recipient and no one else has access to the design). S1 and S2 were originally introduced in [2] as key services in the IP authentication problem. The problem of design traceability was considered in [3]. Most widely available solutions to the IP authentication problem are based on bitstream encryption. The key used to encrypt the bitstream is stored in either non-volatile memory added to the FPGA or the FPGA stores a long-term key in a few hundred bits of dedicated RAM backed-up by an externally connected battery. Both solutions come with a price penalty and are therefore not very attractive. The second solution has the additional disadvantage of the battery's limited life time, which further shortens the design's operating life. Both effects can result in the key and the design being lost after some time, rendering the overall IP block non-functional.

CONTRIBUTIONS. In this paper, we will focus on providing services S1, S2 and S3. We depart from previous works and study the advantages that asymmetric cryptography provides in this setting. In particular, this allows that secret-key information never has to leave the FPGA in contrast to the proposals in [2, 4]. As usual this comes at the cost of having to implement public-key cryptography on the FPGA, which requires more hardware resources than symmetric-key based constructions. However, we also show that it is possible to obtain a *performance* comparable to that of symmetric-key cryptography by encrypting and scrambling in the sense of [5]. In addition, we describe two possible implementations

1-4244-1060-6/07/$25.00 ©2007 IEEE.

of Physical Unclonable Functions (PUFs) one of which does not require modification of the hardware and thus, it is intrinsic to the FPGA. Finally, we show some of the trade-offs that can be made when implementing a fuzzy extractor.

NOTATION. We denote an IP block by SW and use this terminology interchangeably. We write $\mathtt{Enc}_K(\cdot)$ to mean the symmetric encryption of the argument under key K. We assume that $\mathtt{Enc}(\cdot)$ provides semantic security against chosen plaintext attacks as the standard CBC mode of operation for symmetric ciphers provides. Similarly, we write $\mathtt{Enc}_{K_{pub_I}}(\cdot)$ to indicate the PK encryption using I's public key and $\mathtt{Sig}_{K_{priv_I}}(\cdot)$ to indicate a signature on the argument using the private key K_{priv_I} of I. Decryption and verification operations are then written $\mathtt{Dec}_{K_{priv_I}}(\cdot)$ and $\mathtt{Ver}_{K_{pub_I}}(\cdot)$, respectively. We will denote an error correcting code[1] by $\mathcal{C}$ and a set of universal hash functions [6] by $\mathcal{H}$.

ORGANIZATION. Section 2 provides an overview of PUFs, their properties, security assumptions, and fuzzy extractors. In Sect. 3, we introduce a PK-crypto based protocol that allows for a solution in which the private key associated with a device does not need to leave it, even during enrollment. Section 4 describes several PUF constructions including intrinsic PUFs which are based on the properties of SRAM memories present in FPGAs. We end in Sect. 5 analyzing the cost of a fuzzy extractor implementation.

2. PHYSICAL UNCLONABLE FUNCTIONS

Physical Unclonable Functions, introduced by Pappu et al. [7], consist of inherently unclonable physical systems. They inherit their unclonability from the fact that they consist of random components present in the manufacturing process that cannot be controlled. When a stimulus is applied to the system, it reacts with a response. Such a pair of a stimulus C and a response R is called a *challenge-response* pair (CRP). In particular, a PUF is considered as a function that maps challenges to responses. The following assumptions are made on the PUF: (1) it is assumed that a response R_i (to a challenge C_i) gives only a negligible amount of information on another response R_j (to a different challenge C_j) with $i \neq j$; (2) without having the corresponding PUF at hand, it is impossible to come up with the response R_i corresponding to a challenge C_i, except with negligible probability; and (3) it is assumed that PUFs are tamper evident. This implies that when an attacker tries to investigate the PUF to obtain detailed information of its structure, the PUF is destroyed. In other words, the PUF's challenge-response behavior is changed substantially. Due to noise, PUFs are observed over a noisy measurement channel *i.e.* when a PUF is challenged with C_i a response R'_i which is a noisy version of R_i is obtained.

FUZZY EXTRACTOR/HELPER DATA ALGORITHM. In [8] it was shown how PUFs can be used to store keys in a secure way. Since PUF responses are noisy and the responses are not fully random, a Fuzzy Extractor or Helper Data Algorithm [9, 10] is needed to extract one (or more) secure keys from the PUF responses. In what follows, we will make use of an error correcting code $\mathcal{C}$ and a set $\mathcal{H}$ of universal hash functions. Due to the noisy nature of the PUF data, helper data W are generated during the *enrollment phase*. The enrollment phase (carried out in a trusted environment) runs a probabilistic procedure called Gen, which involves: (1) obtaining a response R and choosing a random code word C_S from $\mathcal{C}$; (2) then, a first helper data vector equal to $W_1 = C_S \oplus R$ is generated, (3) a hash function h_i is chosen at random from $\mathcal{H}$ and (4) the key K is defined as $K \leftarrow h_i(R)$ and the helper data $W_2 = i$. Summarizing the procedure Gen is defined as follows, $(K, W) = (K, \{W_1, W_2\}) \leftarrow \mathsf{Gen}(R)$. Later during the *key reconstruction* phase, the key is reconstructed based on a noisy measurement R'_i and the helper data W. In particular, a procedure $K \leftarrow \mathsf{Rep}(R', W)$ is run. During the procedure Rep the following steps are carried out: (1) *Information Reconciliation*, using the helper data W_1, $W_1 \oplus R'$ is computed, then the decoding algorithm of $\mathcal{C}$ is used to obtain C_S, and from C_S, R is reconstructed as $R = W_1 \oplus C_S$; and (2) *Privacy amplification*, the helper data W_2 is used to choose the correct hash function $h_i \in \mathcal{H}$ and to reconstruct the key as $K = h_i(R)$.

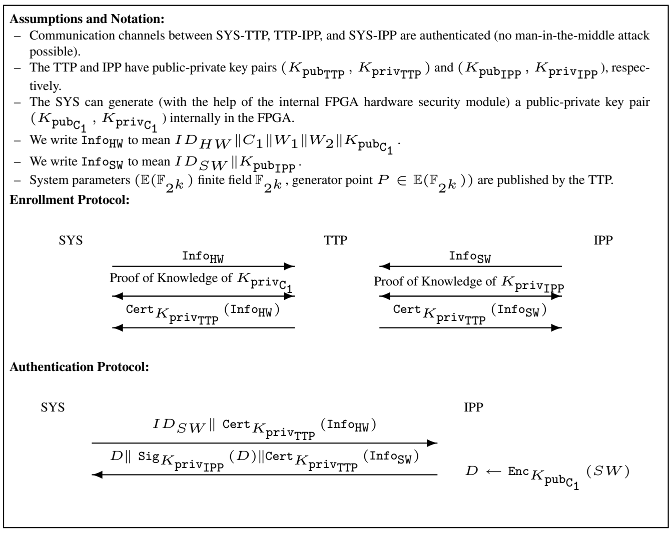
3. PUF-BASED IP AUTHENTICATION FOR FPGAS

In this section, we introduce protocols based on PK encryption for protection of IP blocks. We also compare our protocol to previous protocols in the literature and analyze the advantages that PK cryptography provides in this setting. In the discussed protocols, we only deal with the following parties: the system integrator or designer (SYS), the hardware IP-Provider or core vendor (IPP), the hardware/FPGA manufacturer (HWM) or vendor, and a Trusted Third Party (TTP) [11]. We refer to [11] for a detailed description of all parties in the IP protection chain. Notice that in the protocol proposed in this section as well as in the work presented in [2] and [4], it is assumed that there exists an internal security module with access to the PUF circuit and either an AES module [2, 4] or elliptic curve (EC) and hash modules. In contrast to [2, 4], in this paper the private key never has to leave the device, even during enrollment.

PUBLIC-KEY (PK) BASED APPROACHES. Previous pro-

[1] Given a $[n, k, d]$-code $\mathcal{C}$ over $\mathbb{F}_q$ its words are n-tuples of $\mathbb{F}_q$ elements. The code has minimum distance d, it can correct up to $\lfloor (d-1)/2 \rfloor$ errors, and it has cardinality q^k; i.e. it can encode up to q^k possible messages.

tocols for IP protection [2, 4] are based on the use of symmetric-key cryptography and PUFs. In this section, we investigate the advantages that PK cryptography provides in this setting. Figure 1 shows the proposed protocol. Notice that in contrast to Fig. 2, the PK-based protocol in Fig. 1 requires the TTP to interact only during the enrollment phase. In addition, we do not require an additional nonce to guarantee privacy from the TTP, as PK crypto gives us this for "free".



Assumptions and Notation:

- Communication channels between SYS-TTP, TTP-IPP, and SYS-IPP are authenticated (no man-in-the-middle attack possible).
- The TTP and IPP have public-private key pairs $(K_{\mathtt{pub}_{\mathtt{TTP}}}, K_{\mathtt{priv}_{\mathtt{TTP}}})$ and $(K_{\mathtt{pub}_{\mathtt{IPP}}}, K_{\mathtt{priv}_{\mathtt{IPP}}})$, respectively.
- The SYS can generate (with the help of the internal FPGA hardware security module) a public-private key pair $(K_{\mathtt{pub}_{\mathtt{C}_1}}, K_{\mathtt{priv}_{\mathtt{C}_1}})$ internally in the FPGA.
- We write $\mathtt{Info}_{\mathtt{HW}}$ to mean $ID_{HW} \| C_1 \| W_1 \| W_2 \| K_{\mathtt{pub}_{\mathtt{C}_1}}$.
- We write $\mathtt{Info}_{\mathtt{SW}}$ to mean $ID_{SW} \| K_{\mathtt{pub}_{\mathtt{IPP}}}$.
- System parameters ($\mathbb{E}(\mathbb{F}_{2^k})$ finite field $\mathbb{F}_{2^k}$, generator point $P \in \mathbb{E}(\mathbb{F}_{2^k})$) are published by the TTP.

Enrollment Protocol:

SYS
TTP
IPP
$\mathtt{Info}_{\mathtt{HW}}$
Proof of Knowledge of $K_{\mathtt{priv}_{\mathtt{C}_1}}$
$\mathtt{Cert}_{K_{\mathtt{priv}_{\mathtt{TTP}}}}(\mathtt{Info}_{\mathtt{HW}})$
$\mathtt{Info}_{\mathtt{SW}}$
Proof of Knowledge of $K_{\mathtt{priv}_{\mathtt{IPP}}}$
$\mathtt{Cert}_{K_{\mathtt{priv}_{\mathtt{TTP}}}}(\mathtt{Info}_{\mathtt{SW}})$

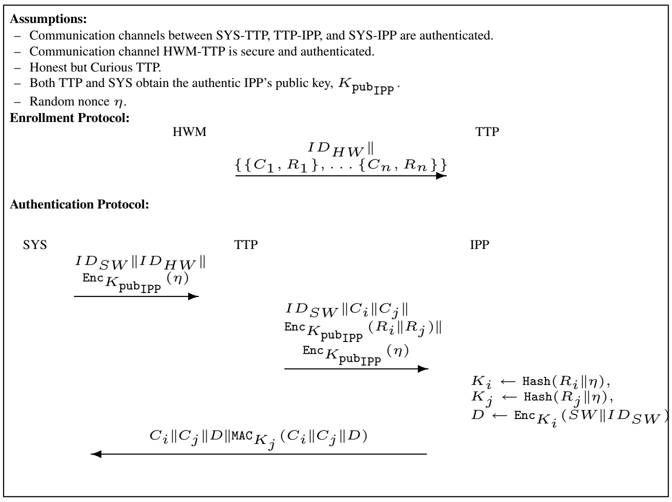
Authentication Protocol:

SYS
IPP
$ID_{SW} \| \mathtt{Cert}_{K_{\mathtt{priv}_{\mathtt{TTP}}}}(\mathtt{Info}_{\mathtt{HW}})$
$D \| \mathtt{Sig}_{K_{\mathtt{priv}_{\mathtt{IPP}}}}(D) \| \mathtt{Cert}_{K_{\mathtt{priv}_{\mathtt{TTP}}}}(\mathtt{Info}_{\mathtt{SW}})$
$D \leftarrow \mathtt{Enc}_{K_{\mathtt{pub}_{\mathtt{C}_1}}}(SW)$

Fig. 1. New Public-key Based Authentication Protocol.

During enrollment the SYS obtains the system parameters published by the TTP, which include: finite field $\mathbb{F}_{2^k}$, an elliptic curve $\mathbb{E}(\mathbb{F}_{2^k})$, and an EC point $P \in \mathbb{E}(\mathbb{F}_{2^k})$ of prime order. Then, SYS instructs the internal FPGA security module to generate a new private-public key pair, by choosing a challenge C_1 and helper data W_1, W_2, deriving a secret key $K_{\mathtt{priv}_{\mathtt{C}_1}}$ (which corresponds to $h_i(R)$, where R is the PUF response as in Sect. 2) and computing $K_{\mathtt{pub}_{\mathtt{C}_1}} = K_{\mathtt{priv}_{\mathtt{C}_1}} \cdot P$. Notice that $K_{\mathtt{priv}_{\mathtt{C}_1}}$ is an integer and $K_{\mathtt{pub}_{\mathtt{C}_1}} \in \mathbb{E}$. The SYS requests a new certificate from the TTP by sending $C_1, W_1, W_2, K_{\mathtt{pub}_{\mathtt{C}_1}}$ to the TTP and executing a zero knowledge proof (and proving) that he is in possession of a device that knows the private key $K_{\mathtt{priv}_{\mathtt{C}_1}}$ corresponding to the public key $K_{\mathtt{pub}_{\mathtt{C}_1}}$ (without disclosing the private key). Neither SYS nor HWM have direct access to $K_{\mathtt{priv}_{\mathtt{C}_1}}$. Upon successful completion of this proof, the TTP sends the SYS a certificate certifying the public key and the helper data information. The IPP goes through a similar certification procedure for his public key. Then, the authentication protocol between the SYS and IPP consists in exchanging: (i) certificates, which proof authenticity of public keys[2], (ii) the signature on the encrypted bitstream D, which provides authenticity and integrity of D, and (iii) the encrypted bitstream itself, which guarantees confidentiality (even from the TTP).

PREVIOUS SYMMETRIC-KEY (SK) APPROACHES. In [2], a protocol is described that provides hardware IP authentication (S1) and hardware platform authentication (S2). This protocol has been recently simplified in [4]. In addition, the authors in [4] introduce a new protocol that allows the parties involved in the IP-block exchange to communicate without the TTP having access to the IP-block itself. This protocol is shown in Fig. 2. In Fig. 2, we write C_i to denote the PUF challenge *and* the corresponding helper data required to reconstruct the PUF response R_i from a noisy version R'_i. It is implicitly assumed that the circuit used to obtain CRPs during the enrollment protocol is destroyed after enrollment and that subsequently, given a challenge C_i the corresponding response R'_i is *only* available internally to the decryption circuit in the FPGA. Without this assumption, anyone could access R_i, and the protocols proposed in [2, 4] would be completely broken. Finally, notice the last message of the protocol is also sent in an off-line fashion, when the bitstream is loaded onto the FPGA from insecure non-volatile storage.



Assumptions:

- Communication channels between SYS-TTP, TTP-IPP, and SYS-IPP are authenticated.
- Communication channel HWM-TTP is secure and authenticated.
- Honest but Curious TTP.
- Both TTP and SYS obtain the authentic IPP's public key, $K_{\mathtt{pub}_{\mathtt{IPP}}}$.
- Random nonce η.

Enrollment Protocol:

HWM
TTP
$ID_{HW} \| \{\{C_1, R_1\}, \ldots \{C_n, R_n\}\}$

Authentication Protocol:

SYS
TTP
IPP
$ID_{SW} \| ID_{HW} \| \mathtt{Enc}_{K_{\mathtt{pub}_{\mathtt{IPP}}}}(\eta)$
$ID_{SW} \| C_i \| C_j \| \mathtt{Enc}_{K_{\mathtt{pub}_{\mathtt{IPP}}}}(R_i \| R_j) \| \mathtt{Enc}_{K_{\mathtt{pub}_{\mathtt{IPP}}}}(\eta)$
$K_i \leftarrow \mathtt{Hash}(R_i \| \eta)$,
$K_j \leftarrow \mathtt{Hash}(R_j \| \eta)$,
$D \leftarrow \mathtt{Enc}_{K_i}(SW \| ID_{SW})$
$C_i \| C_j \| D \| \mathtt{MAC}_{K_j}(C_i \| C_j \| D)$

Fig. 2. SK-based Authentication Protocol [4]

DISCUSSION. The memory requirements are to a large extent the same for both SK and PK approaches. In particular, the same number of bits of helper data (see Sect. 5) are required to generate the private key in the PK based solution and the secret-key in the SK based solution. Similarly, the error correcting codes necessary in both cases have the same complexity. The PK-based solution requires certificates, which are not present in the SK-based solution. This implies if using straight forward certificates an additional memory requirement of $2 \times 492 = 984$ bits[3], assuming an

[2] It is assumed that the TTP's public key is stored in tamper resistance storage.

[3] This includes only the length of the public key and associated signa-

EC defined over a 163-bit field. However, this can be reduced to 328 bits by using implicit certificates [12]. Regarding performance, we have to perform a signature verification and a decryption. First, notice that efficient implementations of ECC for FPGAs are well known [13] as well as hash functions [14]. Signature performance should not constitute a bottleneck since it requires the computation of a hash (comparable in speed to a MAC in the SK case) plus a PK operation on the hash (less than 1 msec [14]). However, the PK decryption operation could constitute a heavy burden on the application as it has to be performed on $L/163$ blocks, where L is the length of the bitstream. To minimize the impact of PK decryption, there are two possibilities. First, we could modify the protocol in Fig. 1 to simply exchange public keys between IPP and SYS, and then fall back to a protocol similar to [4] for encryption and authentication. This would require the implementation of a symmetric encryption algorithm in addition to ECC and hash function modules. The second possibility is to use the construction introduced in [5]. The idea here is to encrypt a single 163-bit block and use hashes (or a stream cipher seeded with the hash of the unencrypted bitstream) to generate a pseudorandom sequence to encrypt the bitstream with a one-time pad. Such a scheme would require a single PK encryption and two hash computations. Finally, we notice that our PK based scheme allows us to reduce the number of rounds in the protocol compared to [2, 4] and as previously mentioned, the private key does not need to leave the FPGA.

HOW EVERYTHING WORKS TOGETHER. For completeness we describe how the combination of bitstream PK encryption and a key extracted from a PUF works for FPGA IP protection applications. The process is as follows: (i) load the encrypted bitstream D, signature $\mathtt{Sig}_{K_{\mathtt{priv}_{\mathtt{IPP}}}}(D)$, and certificate $\mathtt{Cert}_{K_{\mathtt{priv}_{\mathtt{TTP}}}}(\mathtt{Info}_{\mathtt{SW}})$ onto the FPGA, the FPGA then (ii) verifies the certificate information and signature using the TTP's public key $K_{\mathtt{pub}_{\mathtt{TTP}}}$, and (iii) verifies the signature on the encrypted bitstream D using the IPP's public key $K_{\mathtt{pub}_{\mathtt{IPP}}}$ (if any of these checks fails configuration is aborted), finally internally the FPGA (iv) challenges the PUF with challenge C_i (v) measures the PUF response R_i', (vi) retrieves helper data W_1, W_2 from memory, (vii) uses a fuzzy extractor to extract the key $K_{\mathtt{priv}_{\mathtt{C}_1}} \leftarrow \mathtt{Rep}(R_i', W_1, W_2)$, (viii) decrypts the bitstream, and finally (ix) configures the FPGA with the plain bitstream.

4. PUF CONSTRUCTIONS

This section describes known PUF constructions including: Optical and Silicon PUFs, Coating PUFs, and SRAM PUFs.

SILICON PUFS AND OPTICAL PUFS. Pappu et al. [7] introduce the idea of a Physical One-Way Function. They use a bubble-filled transparent epoxy wafer and shine a laser beam through it leading to a response interference pattern. This kind of optical PUF is hard to use in the field because of the difficulty to have a tamper resistant measuring device. Gassend et al. introduce Silicon Physical Random Functions (SPUF) [15] which use manufacturing process variations in ICs with identical masks to uniquely characterize each chip. The statistical delay variations of transistors and wires in the IC were used to create a parameterized self oscillating circuit to measure frequency which characterizes each IC. Silicon PUFs are very sensitive to environmental variations like temperature and voltage. Lim et al. [16] introduce *arbiter based* PUFs which use a differential structure and an arbiter to distinguish the difference in the delay between the paths. Gassend et al. [17] also define a Controlled Physical Random Function (CPUF) which can only be accessed via an algorithm that is physically bound to the randomness source in an inseparable way. This control algorithm can be used to measure the PUF but also to protect a "weak" PUF from external attacks. Recently, Su et al. [18] present a custom built circuit array of cross-coupled NOR gate latches to uniquely identify an IC. Here, small transistor threshold voltage V_t differences that are caused due to process variations lead to a mismatch in the latch to store a 1 or a 0.

COATING PUFS. In [8], Tuyls et al. present coating PUFs in which an IC is covered with a protective matrix coating, doped with random dielectric particles at random locations. The IC also has a top metal layer with an array of sensors to measure the local capacitance of the coating matrix that is used to characterize the IC. The measurement circuit is integrated in the IC, making it a controlled PUF. It is shown in [8] that it is possible to extract up to three key bits from each sensor in the IC leading to approximately 600 $\mathrm{bits/mm}^2$. A key observation in [8] is that the coating can be used to store keys (rather than as a CRP repository as in previous works) and that these keys are not stored in memory. Rather, whenever an application requires the key, the key is generated on the fly. This makes it much more difficult for an attacker to compromise key material in security applications. Finally, Tuyls et al. [8] show that active attacks on the coating can be easily detected, thus, making it a good countermeasure against probing attacks.

FPGA INTRINSIC PUFS AND SRAM MEMORIES. The disadvantage of most of the previous approaches is the use of custom built circuits or the modification of the IC manufacturing process to generate a reliable PUF. In [4], the authors approach the problem by identifying an *Intrinsic* PUF which is defined as a PUF already present in the device and that requires no modification to satisfy the security goals.

ture. The certificate information has to be included in both SK and PK solutions, as previously argued.

We describe next how SRAM memories, which are widely available in almost every computing device including modern FPGAs, can be used as an Intrinsic PUF.

A CMOS SRAM cell is a six transistor device formed of two cross-coupled inverters and two access transistors connecting to the data bit-lines based on the word-line signal. The transistors forming the cross-coupled inverters are constructed particularly weak to allow driving them easily to 0 or 1 during a write process. Hence, these transistors are extremely vulnerable to atomic level intrinsic fluctuations which are outside the control of the manufacturing process and independent of the transistor location on the chip (see [19]). In practice, SRAM cells are constructed with proper width/length ratios between the different transistors such that these fluctuations do not affect the reading and writing process under normal operation. However, during power-up, the cross-coupled inverters of a SRAM cell are not subject to any externally exerted signal. Therefore, any minor voltage difference that shows up on the transistors due to intrinsic parameter variations will tend toward a 0 or a 1 caused by the amplifying effect of each inverter acting on the output of the other inverter. Hence with high probability an SRAM cell will start in the same state upon power-up. On the other hand, different SRAM cells will behave randomly and independently from each other. In [4], the authors consider as a challenge a range of memory locations within a SRAM memory block. The response are the start-up values at these locations. Notice also that SRAM-based PUFs produce a binary string as result of a measurement, in contrast to other PUFs, which have to go through a quantization process before obtaining a bit string from the measurement. This results in a reduction in the complexity of the measurement circuit. For our proof of concept, we use FPGAs which include dedicated RAM blocks.

In order to be useful as a PUF, SRAM startup values should have good statistical properties and be robust over time, to temperature variations, and have good identification performance. These properties were studied in [4]. Here we summarize their findings. Regarding robustness, the Hamming distance between bit strings from repeated measurements of the same SRAM block (intra-class measurements) should be small enough, such that errors between enrollment and authentication measurements can be corrected by an error correcting code admitting efficient decoding. In [4], the authors compared the Hamming distance between a first measurement and repeated measurements of the same SRAM block carried over approximately two weeks. The experiment was done with four different RAM blocks, located in two different FPGAs. The measurements showed that less than 4% of the startup bit values change over time. Similarly, preliminary data indicates that measurements at temperatures ranging from $-20°$C to $80°$C result in bit strings with maximum fractional Hamming distances of 12% when compared to a reference measurement performed at $20°$C. Finally, we notice that intra-class Hamming distances of the SRAM startup values should remain small, even when other data has been written into the memory before the FPGA was restarted. In particular, it is important that the startup values are unaffected by aging and the use of the SRAM blocks to store data. The tests in [4] indicate that storing zeros or ones into the memory has very little influence in the SRAM start-up values. The fractional Hamming distance between bit strings from an enrollment (reference) measurement and any of the other measurements does not exceed 4.5% in this test. The fractional Hamming distance between bit strings of different SRAM blocks and different FPGAs should be close to 50%, such that each FGPA can be uniquely identified. Reference [4] investigated the distribution of Hamming distances between 8190-byte long strings derived from different SRAM blocks (inter-class distribution). The analysis shows that the inter-class fractional Hamming distance distribution closely matches a normal distribution with mean 49.97% and a standard deviation of 0.3%. The intra-class fractional Hamming distance distribution of startup bit strings has an average of 3.57% and a standard deviation of 0.13%.

5. ON THE COST OF EXTRACTING A 163-BIT KEY

It is well known that due to the noisy nature of PUFs a fuzzy extractor is required. A fuzzy extractor, as explained in Sect. 2, provides error correction capabilities to take care of the noisy measurements and privacy amplification to guarantee the uniform distribution of the final secret. In general, we will need to choose an error correcting code, implement its decoding algorithm on the FPGA, and implement a universal hash function, chosen at random from a set $\mathcal{H}$ during enrollment. In the following, we describe the choices that can be made to derive a 163-bit key, which can be used in combination with elliptic curve cryptography and the protocols proposed in Sect. 3.

SECRECY RATE AND ERROR CORRECTION. The fuzzy extractor derives a key K from the SRAM startup bits R by compressing these bits with a hash function h_i. The minimal amount of compression that needs to be applied by the hash function is expressed in the secrecy rate S_R [20]. The maximum achievable secrecy rate S_R is given by the mutual information between bit strings derived during enrollment and reconstruction, $\mathbf{I}(R, R')$. In [20], a method was presented for estimating this secrecy rate using a universal source coding algorithm called the Context-Tree Weighting Method. We have applied this method to the SRAM startup values. By estimating $\mathbf{I}(R, R')$ between repeated measurements of the same memory block, we find an average S_R of 0.76 bits per SRAM memory bit. That means that to derive a secret of size N, we need at least $\lceil 1.32N \rceil$ source bits. In order to choose $\mathcal{C}$, we first consider the number of bits of information, which have to be at least $\lceil 1.32N \rceil_{N=163} = 216$ bits. Assuming that all bits are independent, the probability that a string of S bits will have more than t errors, denoted by P_{total}, is given by $\sum_{i=t+1}^{S} \binom{S}{i} p_b^i (1 - p_b)^{S-i} = 1 - \sum_{i=0}^{t} \binom{S}{i} p_b^i (1 - p_b)^{S-i}$, where p_b denotes the bit error probability. Notice that the maximum number of errors that we have experimentally seen is about 12%. Thus, assume conservatively that we have a bit error probability $p_b = 0.15$ and that we are willing to accept a failure rate of $P_{total} = 10^{-6}$. Since, we are assuming that the errors are independent, a binary BCH code is a good candidate with N-bit code words and a minimum distance at least $d = 2t + 1$. Since we need to generate in the end at least 216 information bits, it becomes an optimization problem to choose the best code in terms of hardware resources, number of SRAM bits required, performance, etc. For example, using $[511, 19, t = 119]$-BCH, we would need $12 \times 511 = 6132$ bits to generate 228 information bits. On the other hand, if we assume $p_b = 0.06$ (i.e. assume that we only need to operate at $20°C$), then we could use the binary $[1023, 278, t = 102]$-BCH code, which requires only 1023 bits of SRAM memory to generate 278 bits of information.

PRIVACY AMPLIFICATION. There has been extensive research on universal hash functions (see for example [21]). However, their suitability for hardware implementations has not been thoroughly investigated. To our knowledge, [22, 23] are the only ones that consider their hardware implementation. However, no one seems to have considered their implementation on FPGAs. Thus, we will consider what the best architecture for FPGAs is in future work.

6. CONCLUSIONS

In this paper, we have proposed new and efficient protocols for the IP-protection problem based on PK cryptographic primitives. In addition, we have described known PUF constructions with particular attention to those that are based on the properties of SRAM because of their presence in current FPGAs. We have tested this construction on FPGAs with embedded block RAM memories which are not reset at power-up. We have seen similar phenomena in ASICs and expect similar behavior on any other device which contains uninitialized SRAM memory. At present, we have identified other properties of SRAM memory, which have the potential to be used as a PUF-source. This will be investigated in future work. We will also explore in the future the exact complexity of implementing a fuzzy extractor on an FPGA. Finally, we notice that the unique identifiers derived from the PUFs could be useful for tracking purposes.

7. REFERENCES

[1] KPMG Electronics, Software & Services and Alliance for Gray Market and Counterfeit Abatement, “Managing the Risks of Counterfeiting in the Information Technology Industry, White Paper,” 2005, available at http://www.agmaglobal.org/.

[2] E. Simpson and P. Schaumont, “Offline Hardware/Software Authentication for Reconfigurable Platforms,” in *Cryptographic Hardware and Embedded Systems — CHES 2006*, ser. LNCS, L. Goubin and M. Matsui, Eds., vol. 4249. Springer, October 10-13, 2006, pp. 311–323.

[3] A. B. Kahng, J. Lach, W. H. Mangione-Smith, S. Mantik, I. L. Markov, M. Potkonjak, P. Tucker, H. Wang, and G. Wolfe, “Watermarking techniques for intellectual property protection,” in *Design Automation Conference — DAC ’98*. New York, NY, USA: ACM Press, 1998, pp. 776–781.

[4] J. Guajardo, S. S. Kumar, G.-J. Schrijen, and P. Tuyls, “FPGA Intrinsic PUFs and Their Use for IP Protection,” in *Cryptographic Hardware and Embedded Systems — CHES 2007*, ser. LNCS. Springer, To appear 2007.

[5] M. Jakobsson, J. P. Stern, and M. Yung, “Scramble All, Encrypt Small,” in *Fast Software Encryption — FSE’99*, ser. LNCS, L. R. Knudsen, Ed., vol. 1636. Springer, March 24-26, 1999, pp. 95–111.

[6] L. Carter and M. N. Wegman, “Universal Classes of Hash Functions,” *J. Comput. Syst. Sci.*, vol. 18, no. 2, pp. 143–154, 1979.

[7] R. S. Pappu, B. Recht, J. Taylor, and N. Gershenfeld, “Physical one-way functions,” *Science*, vol. 297, no. 6, pp. 2026–2030, 2002, available at http://web.media.mit.edu/~brecht/papers/02.PapEA.powf.pdf.

[8] P. Tuyls, G.-J. Schrijen, B. Skoric, J. van Geloven, N. Verhaegh, and R. Wolters, “Read-Proof Hardware from Protective Coatings,” in *Cryptographic Hardware and Embedded Systems — CHES 2006*, ser. Lecture Notes in Computer Science, vol. 4249. Springer, October 10-13, 2006, pp. 369–383.

[9] Y. Dodis, M. Reyzin, and A. Smith, “Fuzzy extractors: How to generate strong keys from biometrics and other noisy data,” in *Advances in Cryptology —- EUROCRYPT 2004*, ser. LNCS, C. Cachin and J. Camenisch, Eds., vol. 3027. Springer-Verlag, 2004, pp. 523–540.

[10] J.-P. M. G. Linnartz and P. Tuyls, “New Shielding Functions to Enhance Privacy and Prevent Misuse of Biometric Templates,” in *Audio-and Video-Based Biometrie Person Authentication — AVBPA 2003*, ser. LNCS, J. Kittler and M. S. Nixon, Eds., vol. 2688. Springer, June 9-11, 2003, pp. 393–402.

[11] T. Kean, “Cryptographic rights management of FPGA intellectual property cores,” in *ACM/SIGDA tenth international symposium on Field-programmable gate arrays — FPGA 2002*, 2002, pp. 113–118.

[12] D. R. L. Brown, R. P. Gallant, and S. A. Vanstone, “Provably Secure Implicit Certificate Schemes,” in *Financial Cryp-*

tography — FC 2001, ser. LNCS, F. S. P, Ed., vol. 2339. Springer, February 19-22, 2001, pp. 156–165.

[13] N. Gura, S. C. Shantz, H. Eberle, S. Gupta, V. Gupta, D. Finchelstein, E. Goupy, and D. Stebila, "An End-to-End Systems Approach to Elliptic Curve Cryptography," in *Cryptographic Hardware and Embedded Systems — CHES 2002*, ser. LNCS, S. K. B, Ç. K. Koç, and C. Paar, Eds., vol. 2523. Springer, August 13-15, 2002, pp. 349–365.

[14] R. Lien, T. Grembowski, and K. Gaj, "A 1 Gbit/s Partially Unrolled Architecture of Hash Functions SHA-1 and SHA-512," in *Topics in Cryptology — CT-RSA 2004*, ser. LNCS, T. Okamoto, Ed., vol. 2964. Springer, February 23-27, 2004, pp. 324–338.

[15] B. Gassend, D. E. Clarke, M. van Dijk, and S. Devadas, "Silicon physical unknown functions," in *ACM Conference on Computer and Communications Security — CCS 2002*, V. Atluri, Ed. ACM, November 2002, pp. 148–160.

[16] D. Lim, J. W. Lee, B. Gassend, G. E. Suh, M. van Dijk, and S. Devadas, "Extracting secret keys from integrated circuits," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 13, no. 10, pp. 1200–1205, October 2005. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1561249

[17] B. Gassend, D. Clarke, M. van Dijk, and S. Devadas, "Controlled Physical Random Functions," in *ACSAC '02: Proceedings of the 18th Annual Computer Security Applications Conference*. Washington, DC, USA: IEEE Computer Society, 2002, p. 149.

[18] Y. Su, J. Holleman, and B. Otis, "A 1.6pJ/bit 96% Stable Chip-ID Generating Cicuit using Process Variations," in *ISSCC '07: IEEE International Solid-State Circuits Conference*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 406–408.

[19] B. Cheng, S. Roy, and A. Asenov, "The impact of random doping effects on CMOS SRAM cell," in *European Solid State Circuits Conference*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 219–222.

[20] T. Ignatenko, G. Schrijen, B. Skoric, P. Tuyls, and F. Willems, "Estimating the Secrecy-Rate of Physical Unclonable Functions with the Context-Tree Weighting Method," in *IEEE International Symposium on Information Theory*, Seattle, USA, July 2006, pp. 499–503.

[21] W. Nevelsteen and B. Preneel, "Software Performance of Universal Hash Functions," in *Advances in Cryptology — EUROCRYPT'99*, ser. LNCS, J. Stern, Ed., vol. 1592. Springer, May 2-6, 1999, pp. 24–41.

[22] H. Krawczyk, "LFSR-based Hashing and Authentication," in *Advances in Cryptology - CRYPTO '94*, ser. LNCS, Y. Desmedt, Ed., vol. 839. Springer, August 21-25, 1994, pp. 129–139.

[23] J.-P. Kaps, K. Y., and B. Sunar, "Energy Scalable Universal Hashing." *IEEE Trans. Computers*, vol. 54, no. 12, pp. 1484–1495, 2005.