SCA Secure and Updatable Crypto Engines for FPGA SoC Bitstream Decryption

SCA Secure and Updatable Crypto Engines for FPGA SoC Bitstream Decryption

Anonymous Author(s)

ABSTRACT

FPGA system on chips (SoCs) are ideal computing platforms for edge devices in applications which require high performance through hardware acceleration and updatability due to long operation in the field. A secure update of hardware functionality can in general be achieved by using built-in cryptographic engines and provided secret key storage. However, reported examples have shown that such cryptographic engines may become insecure against side-channel attacks at any later point in time. This leaves already deployed systems vulnerable without any clear mitigation options. To solve this, we propose a comprehensive concept that uses an alternative and side-channel protected cryptographic engine within the FPGA logic instead of the built-in one for the crucial task of bitstream decryption. Remarkably this concept even allows to update the cryptographic engine itself. As proof of concept, we describe an application to the Xilinx Zynq-7020 FPGA SoC in detail using a *leakage resilient* decryption engine. The lack of accessible secret key storage poses a significant challenge and requires the use of a *physical unclonable function (PUF)* to generate a device intrinsic secret within the FPGA logic. At the same time this means that no manufacturer provided secret key storage or cryptography is required anymore; only a public key for signature verification of the first stage bootloader and initial static bitstream. We provide empirical results proving the side-channel security of the protected cryptographic engine as well as an evaluation of the PUF quality. The full design and source code is made available to encourage further research in this direction.

CCS CONCEPTS

• **Security and privacy → Side-channel analysis and countermeasures**; *Embedded systems security*; • **Hardware → Reconfigurable logic applications**.

KEYWORDS

secure boot, leakage resilience, PUF, AES, Zynq

ACM Reference Format:

Anonymous Author(s). 2019. SCA Secure and Updatable Crypto Engines for FPGA SoC Bitstream Decryption. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

High performance edge computing is becoming increasingly important in automotive mobility, industrial control and other application domains. Vast amounts of sensor data need to be processed instantly within embedded systems for extraction of relevant information and fast reaction times. This is different to many current applications like speech recognition where data is sent to centralized cloud backends for processing. At the same time it is difficult to pre-determine all required edge functionality for devices which operate in the field for many years. For example, statistical analysis of sensor data through deep neural networks requires high computational capabilities best implemented in hardware, and will likely require fundamental updates during the field lifetime, e.g. to counteract so-called adversarial learning attacks on specific neural network instances or to improve recognition capabilities. FPGA SoCs are considered to be valuable platforms in this regard. They offer powerful application CPUs integrated with hardware acceleration on an FPGA and update capabilities for improved functionality and/or security of the hardware as well as the software. The security of intellectual property (IP) which is implemented in such FPGA-based hardware against read-out and manipulation is the focus of this contribution.

FPGAs (i.e. SRAM-based ones) are configured with the hardware implementation at every power-up. The respective bitstream must be stored in a chip-external non-volatile memory (NVM). Since adversaries with physical access to the field devices must be expected, a read-out is often realistic and the contained IP prone to reverse-engineering. To counteract this, FPGA devices usually provide bitstream encryption and authentication features using dedicated built-in hardwired cryptographic engines. Unfortunately, it has been shown for several devices from different manufacturers that such cryptographic engines could be attacked using side-channel analysis (SCA) [22, 23, 28–30] which is widely known to attackers with physical access. Due to this, it is highly advisable to have a concept for retrofitting and updating the cryptographic engine which is used for bitstream decryption. This applies even if no successful attack is currently known against the cryptographic engine of a certain device. Hence, using an improved engine implemented within the FPGA logic instead of the built-in hardwired engine is the goal as soon as security issues arise. Unfortunately, with most currently available FPGAs and FPGA SoCs, it is difficult to use alternative cryptographic engines within the FPGA logic for this core functionality of bitstream decryption. While partial FPGA configuration is helpful and mostly available, on-chip key storage for symmetric decryption keys is often not accessible from the FPGA logic or not trusted in the long-term. Furthermore, it has been shown that readout of both eFuses and battery backed RAM (BBRAM), which are commonly used to implement key storage, can be possible with state of the art equipment [15, 32]. These circumstances make it difficult to retrofit cryptographic engines with

a clear security benefit without inadvertently creating new attack vectors.

In this contribution we provide a sound concept for integrating an alternative side-channel protected AES implementation into an FPGA SoC for decrypting hardware configuration bitstreams during startup and later operation. This concept allows to securely update the main FPGA functionality and even allows to update the cryptographic engine itself (i.e. to use a different algorithm later). We describe a proof of concept implementation on a Xilinx Zynq-7020 FPGA SoC[1] and discuss the generalization to other devices. It is important to note that this is the first comprehensive solution allowing to *dismiss all manufacturer provided secret key storage* options while enabling *side-channel secure field updates of user IP cores* and updates of the decryption engine.

As an alternative cryptographic engine for authenticated decryption of bitstreams, we use one that is side-channel protected by principles from *leakage resilient cryptography*. Protection through leakage resilience has gained significant attention in the last number of years, also due to the fact that contrary to more mainstream protection mechanisms like masking, it does not require high quality random numbers. This requirement usually poses additional implementation challenges that we avoid. We provide side-channel evaluation results of the protected engine on the Xilinx Zynq device using high-precision laboratory equipment as evidence for its side-channel security.

For complete independence from manufacturer-provided secret key storage, and since many devices do not allow access to a dedicated key storage facility from the FPGA logic, we use a *PUF*-based secret key storage in the FPGA logic. Note that in the case of the targeted Xilinx Zynq-7000, there is no secret key storage which is accessible from FPGA logic. Hence PUF-based key storage is the only option for alternative bitstream decryption engines. PUFs provide a mechanism for key storage by leveraging the manufacturing differences of an integrated circuit (IC) to derive a device intrinsic secret. This eliminates the need for a secure NVM. As evidence for the cryptographic quality of the PUF, we provide evaluation results from 20 Xilinx Zynq-7000 devices.

The concept describes the necessary integration into the FPGA SoC start-up and the precise use of partial reconfiguration (PR). Whenever a core functionality such as bitstream decryption employing secret keys is implemented in the FPGA logic instead of using the hardwired built-in options, this part of the FPGA logic configuration must be authenticated and integrity-protected. Otherwise, an attacker could modify or replace those initial hardware parts to leak secret keys. This makes using built-in bitstream authentication features unavoidable. However, when using public key signatures (e.g. Xilinx Zynq-7000 supports RSA signature authentication), this only requires storing and processing public keys on-chip. No secret private keys are stored on-chip, hence, leading to minimal additional attack surface. In our concept, the integrity of the bootloader and initial FPGA configuration containing the side-channel protected decryption core, PUF, and PR handling are verified in this manner.

[1]The full design and source code will be made available for further research and independent analysis at the time of publication.

2 RELATED WORK

The general idea behind retrofitting an FPGA SoC with custom cryptographic cores to protect user IP has been outlined by Xilinx [25] in a white paper. Their concept includes an AES core together with a PUF and dynamic PR to secure user IP cores. They also suggest to use a manufacturer provided authentication scheme like RSA to authenticate the static bitstream consisting of the custom cryptographic cores. However, little detail is given and no working implementation of this scheme was provided. Therefore a large number of technical challenges, e.g. the PUF error correction and buffering the authentication of bitstream segments, remained unaddressed. The white paper also lacks an analysis of the overall system security, and the various attack vectors which we discuss later are not considered in [25]. An implementation of that scheme was later published by Jacob et al. [10] where they use AES in Galois counter mode (GCM) for decryption and authentication with a twisted bistable ring PUF for the key storage. Their implementation is basic in the way that it provided no comprehensive evaluation of the used PUF and contains no countermeasures against side-channel attacks. Further the FPGA is updated after the decryption of the partial bitstreams but before its integrity is verified, potentially damaging the device if the bitstream has been tampered with. Owen et al. [12] follow a similar approach, but use the PUF to generate a key in a way that is sensitive to all changes within the bitstream to achieve a self-authenticating design. While they therefore do not require the manufacturer provided authentication, they do however need physical access to the chip for the encryption of images since it can only be performed on-chip using a separate bitstream that contains the encryption core. This makes their proposal unsuitable to provide updates for devices in the field as it would entail that said devices need to be brought back to a secure environment for the update.

In contrast to the work of Owen et al., we use the PUF to embed an external secret key and only perform decryption on-chip thereby allowing off-chip encryption of updates. Additionally and extending the work of Jacob et al., we also perform an analysis of the stability and reliability of the PUF on a set of 20 Zynq devices, and implement side-channel countermeasures where required, in order to prevent leakage of secret information.

The work of Kashyap et al. [14] deals with security issues that arise if a partial bitstream is received during the runtime of a device and has to be stored in insecure external memory. They protect the partial bitstream by re-encrypting it with a fresh random key and keeping that key and certain values for integrity checks in internal memory. Their work assumes that the device has securely booted and is already running, thus their scope is a lot narrower than our approach which establishes security from boot until full configuration. Additionally, they also keep the keys in volatile internal memory, so after a reboot the configuration is lost.

3 SYSTEM OVERVIEW

In this section we show how the user IP cores in the FPGA design can be protected using custom cryptographic cores on an FPGA SoC and provide a reference design for the Xilinx Zynq-7020. Custom cores are used for secure key storage and decryption instead of the manufacturer provided built-in key storage and decryption core.

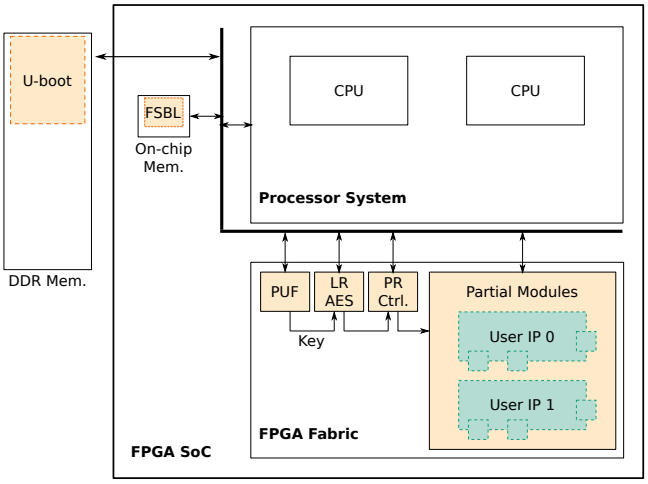


Figure 1: System overview.

Figure 1 provides a system overview of the FPGA SoC consisting of the FPGA fabric and processing system on the same die. The cryptographic cores, namely the leakage resilient AES (LR-AES), PUF and PR controller (depicted in orange), are part of the initial static configuration of the FPGA. This part of the FPGA configuration is independent of the so-called *user design* which contains the actual design to fulfill the application's purpose. All respective user cores (depicted in green) are loaded using dynamic PR once they have been successfully decrypted and authenticated using the initial static parts.

Prior to deployment of the device, a user supplied encryption key is linked to the PUF as a one-time operation. A dedicated software routine is loaded onto the device which retrieves the PUF helper data for that key. The PUF internally creates a device intrinsic secret which in combination with that helper data is later used to regenerate the supplied encryption key (this process is illustrated in Figure 3). Neither the encryption key nor the PUF secret are permanently stored on the device which means that the secret cannot be recovered via offline attacks. After embedding the encryption key, an RSA public key is burned into the eFuses of the device and authenticated boot is enforced. This prevents the execution of an unauthentic boot loader and importantly also stops adversaries from embedding their own key into the PUF after the device is deployed. This does not prevent an authentic user from changing the key at a later point in time if required, once they have physical access to the device.

Figure 2 describes the boot flow of our Xilinx Zynq-7000 implementation once it is deployed. It shows the different stages of the secure boot process starting from power-up until the system has successfully booted and user software is running. After power-up the boot process begins with the hardwired BootROM code in the CPU that is provided by the manufacturer. To establish a chain of trust the first software and logic configuration that is loaded onto the system needs to be protected against manipulation by an adversary. As a result, the BootROM code has to be able to authenticate the first user provided boot code before control is handed over. The BootROM code itself cannot be altered. Hence, all implementations

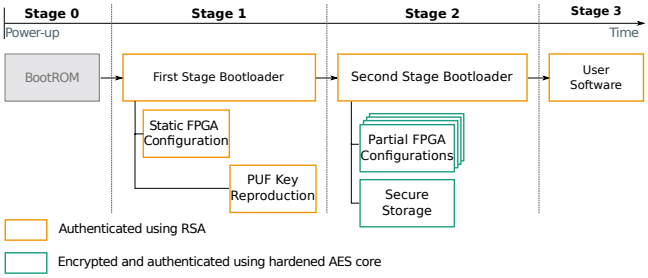


Figure 2: Boot flow of the hardened boot process.

must rely on manufacturer-provided authentication of the first stage boot loader (FSBL), in our case RSA signature verification. However, since the signature check only requires the public part of the RSA key pair, *no secret keys* are stored or processed. Our concept is only based on the assumption that manufacturer-provided storage for the *public key* can be trusted. The FSBL, static bitstream and second stage bootloader (U-boot) are authenticated using this RSA verification routine. After verification, the FSBL is loaded to the on-chip memory and control is handed to it.

The FSBL is a modifiable bootloader that initializes the system, peripherals and FPGA fabric, following which it authenticates and loads the static configuration to the FPGA fabric. We modify the FSBL to transfer the helper data to the PUF module and trigger the regeneration of the encryption key. The helper data is publicly accessible and is stored and authenticated together with the FSBL. Following the successful reproduction of the encryption key, the PUF is locked until the next reset. This prevents any entity from misusing the PUF at a later instance of time in order to attempt to reproduce the key. The encryption key generated by the PUF is directly transferred to the decryption engine and does not leave the FPGA fabric as can be seen in Figure 1. The side-channel hardened AES core is now capable of authenticating and decrypting user hardware IP. The boot process continues with the FSBL authenticating and loading the second stage bootloader, U-boot. As the on-chip memory is too small for U-boot it is usually loaded to external memory.

U-boot is an open-source bootloader commonly used for embedded systems whose functions include system initialization and loading the kernel. In addition to this, we use U-boot to securely load the partial bitstreams. After completing the system initialization, U-boot begins sending encrypted partial bitstreams containing user IP to the hardened AES core for authentication and decryption. Upon successful verification, the plain partial bitstreams are transferred from the AES core directly to the PR controller i.e. they never leave the FPGA fabric and are not transferred on any shared resources of the FPGA SoC. The PR controller dynamically reconfigures the relevant part of the FPGA without interrupting other regions and services. Depending on the application, those user cores may for example contain secret data for higher software layers. The last step in the boot process is for U-boot to load the software stack on the CPU. Currently the software is only authenticated and not encrypted; this is explored in greater detail in Section 8.

This boot sequence comprises the same boot stages as the standard flow supported by Xilinx. The difference is that we leverage

PR and how the partial bitstreams are decrypted and validated, namely by custom cores instead of vendor provided ones. Looking back at Figure 2, in case of the standard boot flow, all user cores would typically be loaded during Stage 1 following their validation using the built-in cryptographic cores. In contrast, we only load the static bitstream consisting of the custom cryptographic cores during this stage, all user cores are loaded during Stage 2 following their successful verification using the custom cores.

If over the lifetime of the device an update of a user core is necessary, the new version can be encrypted off-chip using the encryption key that was embedded during the enrollment process. It can then be sent to the device as remote update without requiring physical access. Remarkably, the side-channel hardened AES core for actual bitstream decryption can also be remotely updated by updating the static bitstream. When changes to the core are made, a new version of the entire static bitstream has to be generated, signed with the RSA private key and then transferred to the device to replace the old one. Updating the PUF, however, requires a secure connection or trusted environment. Due to its nature, changes to the PUF will inevitably change the intrinsic PUF secret and will prevent the recovery of the encryption key with existing helper data. In that case, the encryption key enrollment has to be repeated as it was done for a fresh device.

4 BUILDING BLOCKS

In this section, we describe the functionality of the hardware building blocks that make up the static FPGA configuration of our system: the PUF that generates a device intrinsic secret, the LR-AES which protects confidentiality and integrity of user cores and the PR controller which loads the user cores using dynamic PR.

4.1 Generating a device intrinsic secret

A PUF is a security primitive that utilizes manufacturing process variations to generate a unique digital fingerprint intrinsic to a physical piece of hardware [5]. As this natural variation among silicon chips is outside the control of even the manufacturer, they are inherently difficult to clone. PUF constructions can be broadly split into two categories; challenge-response type PUFs which produce a device unique response for a given input challenge (often referred to as 'strong' PUFs), and identity-generator type PUFs which produces few or just a single identity string for the device (often referred to as 'weak' PUFs)[2]. While these constructions and how they are implemented provide different trade-offs for a designer to explore, for the work here a method to generate a secret key is required. Hence we focus on identity-generator type PUFs where no input challenge is required and modeling type attacks are not in scope.

A number of publications suggest using PUFs for the generation of secret keys, for example the works in [2, 17]. In order to be of use for security applications, a PUF must fulfill certain properties with respect to reliability, entropy and uniqueness. While the evaluation of these properties varies from work to work, they generally include Hamming distance measurements between responses from the same device (indicating the reliability) and between different devices (indicating the uniqueness). These broad measurements metrics can hide subtle biases in the responses however, so care must be taken when performing the analysis. Recent work has looked to formalize this measurement analysis through the ISO standardization process [4].

Rather than using a PUF circuit to directly generate a device dependent key, a *fuzzy commitment* scheme is used where the PUF is used to mask the user-generated secret key [13]. A fuzzy commitment scheme is required as the PUF output is dependent on manufacturing variations hence there is an error probability when reproducing the PUF response, which empirically has been shown to vary across temperature and voltage variations. The fuzzy commitment scheme, as shown in Figure 3, uses an error correcting code (encoder) to introduce redundancy to the key prior to masking with the PUF output. This expanded and masked key can be publicly stored as helper data to regenerate the secret key as required at a later stage. However, where the PUF response doesn't have full entropy, information leakage can occur through this helper data. Should a low entropy PUF be used, key recovery through the publicly available helper data can be possible. Hence in practice, a de-biasing stage should be used in conjunction with an entropy extractor to securely generate a key. There are a number of approaches to implement such a stage, such as Index-Based Syndrome (IBS) coding [35], von Neumann corrector [18] or wiretap coset codes [9] for example. In this work a de-biasing stage is not included as the bias of the PUF is close to ideal as shown in Section 6.1.

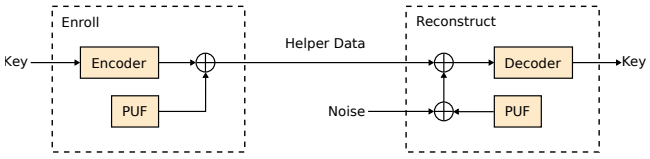


Figure 3: Fuzzy commitment scheme.

The PUF design used in this work is based on the work in [8] which creates a cross coupled feedback loop contained within a single slice of an FPGA, that generates a single random PUF bit. For the error-correction, a $(23, 12, 7)$ Golay linear block code is used, where 12 bits of the message are encoded to a 23-bit codeword, with a Hamming distance of at least 7 between any codeword pair. This allows up to three errors per word to be corrected. Hence the 256-bit key is expanded to a 498-bit codeword and combined with 498-bit produced by the PUF for the generation of the helper data. Once generated, this public helper data is then incorporated as part of the FSBL to enable key generation as outlined in Section 3. A Golay encoder can be implemented very efficiently in hardware using a shift register, with the decoder block utilizing the same encoder block combined with an additional 12-bit look-up table of depth 2048. Both encoder and decoder architectures run in constant time to prevent unintended information leakage from the core.

4.2 Protecting confidentiality and integrity of HW IP

To protect integrity and confidentiality, hardware IP is usually encrypted and protected by a message authentication code (MAC).

[2]Note the terms strong and weak here refer to the entropy requirements rather than the security of the PUF constructions.

This mitigates offline attacks, but since edge nodes might be deployed in hostile environments we also need to protect the decryption against side-channel attacks.

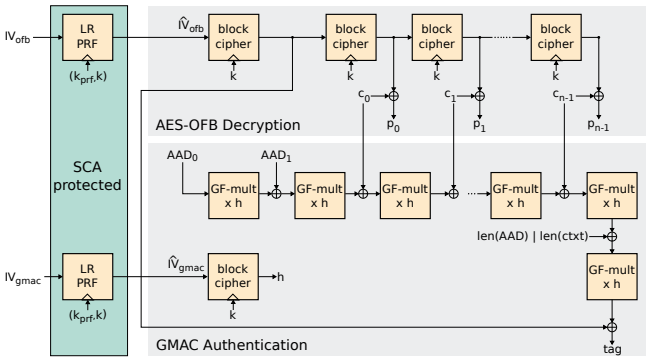


Figure 4: Side-channel secure authenticated decryption.

Figure 4 shows a dataflow diagram of the authenticated decryption core that consists of three components: a side-channel secure stage with a leakage resilient pseudo-random function (LR-PRF), decryption using AES in output-feedback (OFB) mode and authentication using the Galois MAC (GMAC). Leakage resilient cryptography is an algorithmic countermeasure against SCA attacks. It aims to bound the leakage per execution such that an attacker cannot accumulate information about the processed secret. We use the LR-PRF proposed by Unterstein et al. [34] to derive a secret pseudorandom state from public initialization vectors (IVs). Specifically, that means processing two public IVs, IV_{ofb} and IV_{gmac}, with the LR-PRF to get two secret IVs, denoted by $\hat{\mathrm{IV}}_{ofb}$ and $\hat{\mathrm{IV}}_{gmac}$, for the subsequent stream cipher and MAC. This LR-PRF is similar to the one presented by Medwed et al. [20], but it provides security against sophisticated attacks using high precision EM measurements at the cost of a higher latency. In our use case this is acceptable because the computational overhead occurs only once during the boot process. Internally the LR-PRF uses AES-128 encryption, so for the stream cipher we re-use the same AES core in OFB mode which makes the area overhead almost negligible.

For common SCA attacks the precondition is that the attacker can guess some internal value of the algorithm that depends on the secret and an input (e.g. the plaintext) that is known to the attacker. The general idea behind our construction is that only the initial LR-PRF stage needs to be side-channel secure (through means of leakage resilient cryptography), because behind it, no public inputs are processed by the block cipher. This gives no surface to mount an SCA attack on the unprotected stages as there are no known inputs and thus guessing internal values is not possible. A detailed security analysis and a laboratory SCA of our target platform is presented in Section 6.2.

For message authentication we chose GMAC [19] because it can be efficiently implemented in hardware since it only requires an additional Galois field multiplier. Differing from the specification, where the MAC key h is derived by encrypting a plaintext with all zeros, we derive the MAC key by processing another public IV_{gmac} with the LR-PRF and then encrypt the resulting $\hat{\mathrm{IV}}_{gmac}$ to get h. We diverted here because, as explained above, we need to prevent an unsecured encryption where the inputs are known to the attacker. This is achieved by processing the public input with the side-channel secured PRF first.

A general problem when decrypting and authenticating data in one pass is that the authentication tag can only be checked after processing all the data. But sending unauthenticated data to the PR controller, in our case the Xilinx proprietary internal configuration access port (ICAP), can be dangerous. Configuring the FPGA with unauthentic data could damage the FPGA due to short-circuits caused by false configurations as stated in [23]. Hence we need to buffer the decrypted data until it is verified before we pass it on. This buffering has to be done in on-chip memory (OCM), otherwise it would be prone to manipulation by e.g. probing of the memory bus. Therefore we implemented a first in, first out (FIFO) buffer in front of the PR controller that releases data only after its tag has been verified. To keep the allocated block RAM (BRAM) for the FIFO small, we split the bitstream into segments which are decrypted and authenticated individually. While the FIFO is sending authenticated data to the PR controller, decryption of the next segment already starts, thus reducing the latency. To prevent IV re-use, the IV must be unique for every segment. For this purpose, IV_{ofb} is split up into a 96-bit random value and a 32-bit counter value that is incremented with every segment. That also means that at the beginning of every segment, the LR-PRF is evaluated to generate a fresh $\hat{\mathrm{IV}}_{ofb}$. Updating IV_{gmac} in between segments is not necessary because it is only used to generate the GMAC key h which we keep constant for the entire bitstream (note that in the original GMAC scheme h is always derived by encrypting an all zero plaintext).

U-boot only continues the boot process if the verification of all the segments of the partial bitstream succeeds. If the verification of any of the segment fails, U-boot aborts the boot process and system goes into a secure lockdown mode. A power cycle is necessary to remove the system from this state.

4.3 Partial reconfiguration of user IP cores

The PR controller receives the segments of the decrypted partial bitstream from the FIFO and then passes them to the Xilinx ICAP interface. The ICAP is a proprietary interface from Xilinx typically used for configuration readback and reconfiguration of the FPGA. The ICAP has direct access to the configuration memory through the configuration registers, hence can be used to dynamically reconfigure the FPGA. In this work, the ICAP only operates in the device reconfiguration mode and hence no configuration data can be read back through the ICAP.

5 IMPLEMENTATION

Here, we describe the prototype implementation for the Xilinx Zynq-7020 FPGA SoC. The resource utilization is noted in Table 1. Overall, we only use around 22 percent of the available slices and 1.5 percent of the available BRAM of this device, which is among the smaller, low-cost devices of the Xilinx Zynq-7000 product range. Hence, and despite using a smaller device, 78 percent of the slices are still available to implement user IP cores.

Table 1: Resource utilization.

Module	Slices		BRAM
	LUTs	Registers	RAM36
LR Authenticated Decryption	4568	2810	0
PicoPUF + Fuzzy Commitment	3917	4179	1
PR Controller	50	53	0
ICAP FIFO buffer	99	124	1
Overall (incl. interconnects)	9274	7911	2

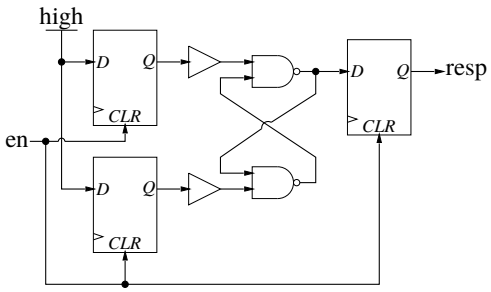
All the hardware building blocks are configured and managed over the AXI interface by software drivers that are patched to a standard U-boot and FSBL. The helper data for the PUF is generated once and then hard coded into the FSBL. As the BRAM modules on the Xilinx Zynq-7000 are each 4.5 KB large, we split the partial bitstreams into segments of 4 KB with an additional 16 bytes for the tag appended to each segment. This configuration leads to a storage overhead of less than 0.4 percent.

6 SECURITY EVALUATION

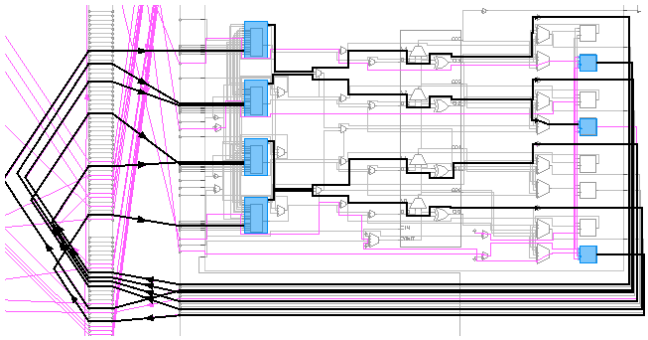
This section presents the crucial elements of the security evaluation. We use an attacker model that allows physical access to the device and full control over all inputs. The attacker is capable of mounting passive side channel attacks by taking measurements of the device. We will in our analysis consider the most powerful method, namely high-precision EM measurements directly over the decapsulated die. For our DUT we had to remove the packaging which technically makes it an invasive attack, but there are derivates within this product family that are shipped as 'naked' flip chip. In that case the die is bonded upside down and the backside silicon is directly accessible for measurements, making it a non-invasive attack. In this work we do not consider active or invasive attacks like glitching, laser fault injection or Focused Ion Beam (FIB) attacks.

6.1 Evaluation of the PUF

In this work, the FPGA-based PUF ID design proposed in [8] is employed to regenerate the secret key. It comprises of 498 elementary 1-bit PUF cells, which are used to mask the output of the Golay linear block code used to encode the 256-bit key as shown in Figure 3. Each PUF cell is designed to fit compactly in one FPGA slice as shown in Figure 5. Figure 5a shows the architecture of each PUF cell comprising of four logic and three register components. The physical layout of a single PUF cell is shown in Figure 5b with the registers and logic depicted in blue, located in a single slice, and the timing-critical routing paths highlighted in black. The routing is fixed with the use of scripts as part of the Vivado design flow to ensure that all cells have identical routing paths, which have been selected to maximize the entropy while retaining sufficient reliability to allow minimal error correction of subsequent evaluations. As the register components are all clocked with the global synchronous clock, this allows them to be placed in the same slice. Subsequently, this enables all timing critical routing between the PUF components to be placed within the local interconnect rather than the general purpose interconnect.



(a) Architecture.



(b) Physical layout.

Figure 5: PicoPUF design.

To assess the suitability of the PUF architecture for our side-channel protected boot design, a 256-bit output of the core PUF module was generated for testing prior to error correction being applied[3]. This output was repeated 1001 times for each of 20 Xilinx Zynq-7000 devices. Analyzing the reliability of each bit, it was found that, at room temperature, ≈ 80% of bits returned the same value for *all* evaluations on *each* device. While temperature and voltage variations will introduce additional noise to the PUF responses, empirical evidence indicates that constraining the design to a single slice helps minimize these effects [8]. The expected fractional Hamming distance (i.e. the noise) between a single given evaluation and the reference response was empirically estimated to be 0.0217, or 5.56 bits over the 256. Given that the Golay encoder can correct 3 in every 23 bits, this should be a sufficient margin for a reliable key generation process assuming the response bits are independent and identically distributed. The response bias was close to ideal, with the average value of a given bit on a given device expected to be 0.498, while the expected fractional Hamming distance between the outputs of two different devices was empirically estimated to be 0.497. Therefore, it is expected that the helper data generated by different devices is sufficiently independent such that an adversary cannot learn anything more about the key given access to the helper data from multiple devices.

SCA attacks on PUF designs can be viewed as attacks on the core PUF instance, and attacks on the post-processing. While attacks

[3]The individual security modules were evaluated separately, the full design requires 498 PUF cells.

such as directly reading out PUF bits using a FIB are outside the scope of the attacker model, there has been work directly attacking the generation of the PUF output. For example, delay based arbiter designs are attacked using power analysis in [3], while recovering frequencies from ring oscillator (RO) PUF designs is investigated in [21]. However, as the PUF design used in this work evaluates in a single clock cycle, it is not expected to be susceptible to attacks on its core bit generation. Of greater threat are attacks on the post-processing stage [21, 31]. Typically such attacks are differential attacks and require the attacker to manipulate the helper data. In our design, the helper data is authenticated before use and thus protected from tampering. This reduces the number of different observable traces to one, effectively preventing all differential power analysis (DPA) attacks on the post-processing. As the post-processing consists of linear operations in our design, even differential attacks are expected to require a large number of traces to recover the PUF output [26]. Under these circumstances, simple power analysis (SPA) attacks using only a single trace, even though it can be averaged to reduce noise, seem infeasible.

6.2 Leakage resilient authenticated decryption

We designed our leakage resilient authenticated decryption scheme following the principle described in [24]. As shown in Figure 4, we rely on an initial "leak-free" stage using a LR-PRF that is protected against SCA. It establishes a pseudo-random intermediate state ($\hat{\text{IV}}_{ofb}$ and $\hat{\text{IV}}_{gmac}$) that is unknown to the attacker and allows us to use unprotected decryption and authentication for the actual workload. Internally, the LR-PRF consists of a so called GGM tree PRF [7] and optionally one or more length-doubling pseudorandom generators (2-PRGs) as described in [34]. Both the PRF and the (one or more) 2-PRG stages are implemented using the same hardware AES core that is also used in the stream cipher part. This means we only need to instantiate one AES core that is then time-shared between the different modules. The number of 2-PRG stages is a design choice that allows to compensate entropy loss after successful side-channel attacks on the very first AES execution which inevitably has to operate on public inputs. The required number of stages depends on the leakage behavior of the device and can be determined through laboratory analysis. Each stage adds a fresh 128-bit key to increase the remaining entropy within the internal secret state. We opted for two 2-PRG stages and thus require two 128-bit keys.

The side-channel security of the LR-PRF is based on two principles: limited data complexity (i.e. the number of operations with the same key, but different inputs that an attacker can observe) and algorithmic noise from parallel S-boxes. As pointed out in [34], a laboratory analysis is mandatory to assess the security level. We implemented the LR-PRF using an AES core with parallel S-boxes on the Xilinx Zynq-7000 and placed the AES as dense as possible as shown in the floorplan in Figure 6. Following the method described in [34] we conducted a localized EM analysis using a near-field EM probe placed on the decapsulated die. We first identified the locations of the S-boxes through a grid scan and then ran template attacks on them using measurements from those locations. For each S-box we collected 400, 000 traces for the profiling and 100, 000 traces for the attack at its respective location. More traces typically lead to better results but the number of traces that can be used is limited by the measurement time. However, during our experiments we found this number to be sufficient and that an increase in either the profiling or the attack set did not improve the attack.

Usually attacks with limited data complexity do not successfully recover all key bytes directly. Instead, some brute force effort is required to combine the most probable candidates for each key byte into full keys and then test each key candidate for its correctness. An evaluator that knows the correct key can use a key rank estimation algorithm [6] to determine the brute force effort even in cases where an actual enumeration of all candidates up to the correct key is not computationally feasible. The security level is then given as the brute force effort after running the template attack, i.e. the number of combined keys an attacker has to try until reaching the correct key. The natural attack vector in our case is the initial execution of the first 2-PRG stage, as this is the only stage where the AES is encrypting public plaintexts. However, this operation only encrypts two different plaintexts; the data complexity is 2. We found that the remaining security level after an attack on this 2-PRG stage is still 2^{120}, or 120 bits. This is significantly higher than the results reported on the Xilinx Spartan 6 platform in [34] where the security level was only 2^{48} and we attribute this difference to the smaller feature size (28 nm compared to 45 nm) and the different placement strategy. Note that these results were achieved with a standalone design and not the entire system. We expect that the increased noise generated by the full design will only make attacks harder. This means that the LR-PRF provides sufficient

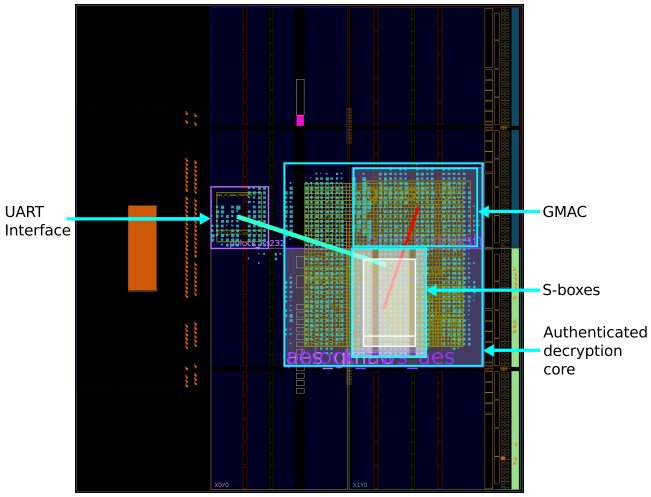


Figure 6: Placement of the AES core for the SCA.

remaining guessing entropy after an attack even if we use the GGM tree directly without the additional 2-PRG stage and key upstream. Nevertheless, we opted to keep the 2-PRG and second key to increase the security margin in case of more sophisticated attacks. The hardware overhead is minimal since we re-use the existing AES engine and thus only pay for this added security with latency and "key storage" in the form of a larger PUF.

Table 2: Overview of system security.

Threat vector	Mitigation
Stage 0. BootROM	
Load malicious FSBL	Integrity check of FSBL using RSA
Stage 1. FSBL and static FPGA configuration	
Learning the PUF	Key reproduction triggered by FSBL, following which PUF is locked
Readout encryption key	Key is directly transferred to AES
Recovery of PUF secret	Key enrollment in the field is blocked
Stage 2. U-boot	
Loading unauthenticated configuration	FIFO is used to buffer configuration

After the LR-PRF stage, we use AES in OFB mode for the actual decryption of the bitstream. We opted for the OFB mode, instead of e.g. GCM mode, because the plain- and ciphertext is not directly an input to the block cipher. As long as an attacker does not know both plain- and ciphertext, the in- and outputs to the block cipher remain secret and provide no surface for side channel attacks. Even partial knowledge, as is the case in counter modes where the initial counter value may be unknown but the increment of the counter is known, can be sufficient to mount an attack [11]. If on any platform portions of the plaintext are known, e.g. because they are all zeros, we propose not encrypting those parts and instead adding them to the additional authenticated data. An open question remains considering the effect of plaintexts that are unknown, but not uniformly random because they e.g. consist of opcodes or addresses that do not utilize the entire value space. We are not aware of any published attacks that exploit such a scenario and imagine such an attack to be hard, nevertheless it remains as an interesting topic for future research.

6.3 Overall system security

The security evaluation of the building blocks, namely PUF and leakage resilient authenticated decryption cores, were presented in Section 6.1 and Section 6.2 respectively. However, the integration of the individual blocks can give rise to new attack vectors. Now we review the overall system security by recalling the boot process and listing the different threat vectors post-integration together with the implemented mitigation techniques. A summary of the findings for each boot stage is listed in Table 2.

Load malicious FSBL or U-boot: An adversary may attempt to bypass the implemented security mechanism by replacing the boot images with malicious ones. This is prevented by using the manufacturer provided RSA signature verification. After power-up, the BootROM verifies the signature of the FSBL, only after successful verification of the signature is the control handed off to the FSBL. This check is enabled by burning an eFuse and cannot be disabled. Subsequently, the FSBL checks the signature of the U-boot. An adversary trying to forge any of those signatures would require the private key which is not stored on the device.

Learning the PUF: In our proof of concept implementation we use an identity-generator type PUF which is not susceptible to modeling attacks. However, if a challenge-response type PUF was used instead, then unrestricted access to its interface could be exploited to learn a model of the PUF and thereafter simulate it to recover its secret. To prevent this, we build in a mechanism to limit the access and interface to the PUF. Following the loading of the static bitstream by the FSBL, the FSBL is allowed to trigger a PUF key reproduction only once. After the successful reproduction of the key, the PUF is locked by the hardware and is not accessible until the next power-up.

Readout of the encryption key: An attacker may attempt to corrupt the bitstream so as to include additional lines to readout the key. This however is prevented by verifying the integrity (at the same time as its authenticity) of the bitstream before it is loaded. Thus, what remains is to attempt to read out the key at run-time, e.g. via existing shared interfaces. To protect the key from being read by an unauthorized entity, the reproduced key is directly transferred to the LR-AES core. The key never leaves the FPGA fabric and is not transferred over any shared resources of the FPGA SoC, this ensures that no other entity has access to the key.

Recovery of the PUF secret: As can be seen in Figure 3, the helper data that is generated during key enrollment is an XOR combination of the encoded encryption key and the PUF secret. If an attacker could enroll his own keys, then it is trivial to calculate the PUF secret from the retrieved helper data. This is prevented by only allowing authorized FSBL code to be executed on the device. No authenticated code for key enrollment exists in the deployed device however (the respective code is executed before setting an authenticated code only configuration). The locking of the PUF after key reproduction also disables the ability to enroll keys during runtime.

Load unauthenticated data to FPGA: Authenticated encryption is used to protect confidentiality and integrity of the partial bitstreams where the decryption and authentication are performed in parallel. As the authentication tag can only be checked at the end, an adversary can send unauthentic data and it will be decrypted before the invalid tag is noticed. If the switch boxes and LUTs of the FPGA are falsely configured, it could result in short-circuits in the fabric and thereby destroying parts of the device. To prevent decrypted data from being directly transferred to the PR controller before the authenticity can be verified, the bitstream is divided into segments. Each segment has its own verification tag and is

buffered after decryption until it has been successfully authenticated. Only after the successful verification of a segment, are the buffered contents transferred to the PR controller.

7 GENERALIZATION

While we target a Xilinx Zynq-7020 FPGA SoC for our proof-of-concept, the method is agnostic to a specific FPGA manufacturer/family as long as it provides certain common functionalities. We now discuss the portability of the presented design to other platforms and the additional steps that need to be taken. The two key requirements to port this design are that the target provides:

(1) Authentication and integrity checking of the static bitstream using public key cryptography
(2) Partial reconfiguration

In the initial boot step, an authenticity and integrity check of the static bitstream containing the custom cryptographic cores is necessary. We used signature verification with public key cryptography on the Xilinx Zynq-7000 device to achieve this. Public key cryptography for this purpose is a widely adopted feature and is present in the majority of FPGA SoCs that are currently available. The benefit of using public key cryptography is that no additional *secret* key material is necessary and the operation does not need to be protected against key-recovery attacks.

PR is necessary to reconfigure parts of the FPGA fabric at runtime with user cores that were decrypted by our protected engine. This feature is supported by the leading manufacturers of FPGA SoCs such as Xilinx and Intel (formerly Altera). Thus this design can be ported to these devices with low effort.

As the Xilinx Zynq-7000 devices do not provide any user key storage (only 32 bits of general purpose eFuses are available), we opted to use a PUF for key storage. Newer devices, however, often provide user accessible secure key storage. Some devices like the Xilinx Zynq Ultrascale+ and Stratix 10 from Intel even include hard-core PUFs and allow secure boot using these PUFs. In such cases the custom PUF can be omitted if the key storage is trusted to be secure. If a custom PUF is used on a different platform, it should be re-evaluated to ensure that the key has sufficient entropy. Additionally, when porting the design to a different technology the leakage behavior of the decryption core can change and the side-channel security should also be re-evaluated as shown in [34]. Considering the large security margin that we found on the Xilinx Zynq-7000 platform, we do not expect any issues on other comparable or newer technologies because smaller feature size makes localized attacks harder.

8 TOWARDS SOFTWARE ENCRYPTION AND RUNTIME SECURITY

Our proposal is currently limited to bitstream decryption. User software is authenticated but not encrypted. For many applications this is sufficient, since rarely the entire software stack is confidential. If full confidentiality is desired for software, this usually goes hand in hand with runtime integrity protection and encryption of chip external RAM, which are hard problems on their own. However, if our hardened core is to be used for software decryption, the OFB mode of operation might not be the best choice and could lead to new attack vectors. The side-channel security of the decryption core relies on the fact that an attacker does not know the inputs to the underlying block cipher. In OFB mode this holds, as long as the attacker cannot observe both plain- and ciphertext. Otherwise, the XOR of both reveals the output of the block cipher. With that information a regular DPA on the last round of the cipher becomes possible. For the case of bitstream encryption, the decrypted bitstream is directly sent to the PR controller, i.e. the plaintext never leaves the hardware and is not exposed to the attacker. For software decryption, this is hard to guarantee since the plaintext is transferred back to the CPU and most probably ends up in external RAM.

A straightforward mitigation is to use key whitening to hide the in- and outputs to the cipher from the attacker. XEX [27] and XTS [1] are modes of operation that achieve this. However, there are several published side-channel attacks on those constructions. Luo et al. show an attack on the tweaking function of XTS that exploits the simple structure of the Galois field multiplication by 2 [16]. This attack can be prevented by using XEX which uses multiplications with arbitrary values. In contrast, Unterluggauer et al. attacked the AES in such a scenario directly by concatenating DPA attacks on the last two rounds [33]. This attack is feasible irrespective of the tweaking function but it requires the attacker to change the input while keeping the IV constant (e.g. writing the same sector multiple times in the case of disk encryption). Whether or not this is a relevant attack vector depends on the application. To provide a sound solution for software encryption, these issues need to be addressed and, depending on the specific use case, additional measures have to be taken which are out of the scope of this work.

9 CONCLUSION

Side-channel attack resistant cryptographic cores and PUF-based key storage are now available in the newest generation of devices of leading vendors Xilinx, Intel and Microsemi. However, all of them are closed-source and none are updatable and it is usually not feasible to upgrade already deployed devices to a new hardware platform. As a step towards a vendor agnostic solution, we present an SCA secure and fully updatable mechanism to securely configure the FPGA logic starting from power-up until the whole system is booted and running. To achieve this, we leverage the PR options of FPGAs, in conjunction with a PUF generated device intrinsic key and a leakage resilient decryption core to securely load hardware IP cores. The presented work, to the best of our knowledge, is the first that allows side-channel secure field updates of user IP cores and the decryption engine without relying on any manufacturer provided *secret key* storage. The concept requires very limited trust in the manufacturer and provides the necessary flexibility if demands change or new attacks arise. This approach is orthogonal to upgrading to newer and more expensive feature-rich devices, and is also suited to retro-fit older devices as it uses features that are already widespread in current hardware.

REFERENCES

[1] 2008. IEEE Standard for Cryptographic Protection of Data on Block-Oriented Storage Devices. *IEEE Std 1619-2007* (April 2008), c1–32. https://doi.org/10.1109/IEEESTD.2008.4493450
[2] Aydin Aysu, Ege Gulcan, Daisuke Moriyama, Patrick Schaumont, and Moti Yung. 2015. End-To-End Design of a PUF-Based Privacy Preserving Authentication Protocol. In *Cryptographic Hardware and Embedded Systems - CHES 2015 - 17th*

International Workshop, Saint-Malo, France, September 13-16, 2015, Proceedings (Lecture Notes in Computer Science), Tim Güneysu and Helena Handschuh (Eds.), Vol. 9293. Springer, 556–576. https://doi.org/10.1007/978-3-662-48324-4_28

[3] Georg T. Becker and Raghavan Kumar. 2014. Active and Passive Side-Channel Attacks on Delay Based PUF Designs. *IACR Cryptology ePrint Archive* 2014 (2014), 287. https://eprint.iacr.org/2014/287

[4] Jean-Luc Danger, Sylvain Guilley, P Nguyen, and Olivier Rioul. 2016. PUFs: Standardization and Evaluation. In *2016 Mobile System Technologies Workshop (MST)*. 12–18. https://doi.org/10.1109/MST.2016.11

[5] Blaise Gassend, Dwaine E. Clarke, Marten van Dijk, and Srinivas Devadas. 2002. Silicon physical random functions. In *Proceedings of the 9th ACM Conference on Computer and Communications Security - CCS 2002, Washington, DC, USA, November 18-22, 2002*, Vijayalakshmi Atluri (Ed.). ACM, 148–160. https://doi.org/10.1145/586110.586132

[6] Cezary Glowacz, Vincent Grosso, Romain Poussier, Joachim Schüth, and François-Xavier Standaert. 2015. Simpler and More Efficient Rank Estimation for Side-Channel Security Assessment. In *Fast Software Encryption - 22nd International Workshop, FSE 2015, Istanbul, Turkey, March 8-11, 2015, Revised Selected Papers (Lecture Notes in Computer Science)*, Gregor Leander (Ed.), Vol. 9054. Springer, 117–129. https://doi.org/10.1007/978-3-662-48116-5_6

[7] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. 1984. How to Construct Random Functions (Extended Abstract). In *25th Annual Symposium on Foundations of Computer Science, West Palm Beach, Florida, USA, 24-26 October 1984*. IEEE Computer Society, 464–479. https://doi.org/10.1109/SFCS.1984.715949

[8] Chongyan Gu and Máire O'Neill. 2015. Ultra-compact and robust FPGA-based PUF identification generator. In *2015 IEEE International Symposium on Circuits and Systems, ISCAS 2015, Lisbon, Portugal, May 24-27, 2015*. IEEE, 934–937. https://doi.org/10.1109/ISCAS.2015.7168788

[9] Matthias Hiller and Aysun Gurur Önalan. 2017. Hiding Secrecy Leakage in Leaky Helper Data. In *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings (Lecture Notes in Computer Science)*, Wieland Fischer and Naofumi Homma (Eds.), Vol. 10529. Springer, 601–619. https://doi.org/10.1007/978-3-319-66787-4_29

[10] Nisha Jacob, Jakob Wittmann, Johann Heyszl, Robert Hesselbarth, Florian Wilde, Michael Pehl, Georg Sigl, and Kai Fischer. 2017. Securing FPGA SoC configurations independent of their manufacturers. In *30th IEEE International System-on-Chip Conference, SOCC 2017, Munich, Germany, September 5-8, 2017*. IEEE, 114–119. https://doi.org/10.1109/SOCC.2017.8226019

[11] Joshua Jaffe. 2007. A First-Order DPA Attack Against AES in Counter Mode with Unknown Initial Counter. In *Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings (Lecture Notes in Computer Science)*, Pascal Paillier and Ingrid Verbauwhede (Eds.), Vol. 4727. Springer, 1–13. https://doi.org/10.1007/978-3-540-74735-2_1

[12] Don Owen Jr., Derek Heeger, Calvin Chan, Wenjie Che, Fareena Saqib, Matthew Areno, and Jim Plusquellic. 2018. An Autonomous, Self-Authenticating, and Self-Contained Secure Boot Process for Field-Programmable Gate Arrays. *Cryptography* 2, 3 (2018), 15. https://doi.org/10.3390/cryptography2030015

[13] Ari Juels and Martin Wattenberg. 1999. A Fuzzy Commitment Scheme. In *CCS '99, Proceedings of the 6th ACM Conference on Computer and Communications Security, Singapore, November 1-4, 1999.*, Juzar Motiwalla and Gene Tsudik (Eds.). ACM, 28–36. https://doi.org/10.1145/319709.319714

[14] Hirak Kashyap and Ricardo Chaves. 2016. Compact and On-the-Fly Secure Dynamic Reconfiguration for Volatile FPGAs. *ACM Trans. Reconfigurable Technol. Syst.* 9, 2, Article 11 (Jan. 2016), 22 pages. https://doi.org/10.1145/2816822

[15] Heiko Lohrke, Shahin Tajik, Thilo Krachenfels, Christian Boit, and Jean-Pierre Seifert. 2018. Key Extraction Using Thermal Laser Stimulation: A Case Study on Xilinx Ultrascale FPGAs. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2018, 3 (2018), 573–595. https://doi.org/10.13154/tches.v2018.i3.573-595

[16] Chao Luo, Yunsi Fei, and A. Adam Ding. 2017. Side-channel power analysis of XTS-AES. In *Design, Automation & Test in Europe Conference & Exhibition, DATE 2017, Lausanne, Switzerland, March 27-31, 2017*, David Atienza and Giorgio Di Natale (Eds.). IEEE, 1330–1335. https://doi.org/10.23919/DATE.2017.7927199

[17] Roel Maes, Anthony Van Herrewege, and Ingrid Verbauwhede. 2012. PUFKY: A Fully Functional PUF-Based Cryptographic Key Generator. In *Cryptographic Hardware and Embedded Systems - CHES 2012 - 14th International Workshop, Leuven, Belgium, September 9-12, 2012. Proceedings (Lecture Notes in Computer Science)*, Emmanuel Prouff and Patrick Schaumont (Eds.), Vol. 7428. Springer, 302–319. https://doi.org/10.1007/978-3-642-33027-8_18

[18] Roel Maes, Vincent van der Leest, Erik van der Sluis, and Frans M. J. Willems. 2016. Secure key generation from biased PUFs: extended version. *J. Cryptographic Engineering* 6, 2 (2016), 121–137. https://doi.org/10.1007/s13389-016-0125-6

[19] David A. McGrew and John Viega. 2004. The Galois/Counter Mode of Operation (GCM). *Submission to NIST Modes of Operation Process* (2004).

[20] Marcel Medwed, François-Xavier Standaert, Ventzislav Nikov, and Martin Feldhofer. 2016. Unknown-Input Attacks in the Parallel Setting: Improving the Security of the CHES 2012 Leakage-Resilient PRF. In *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*. 602–623. https://doi.org/10.1007/978-3-662-53887-6_22

[21] Dominik Merli, Dieter Schuster, Frederic Stumpf, and Georg Sigl. 2011. Side-Channel Analysis of PUFs and Fuzzy Extractors. In *Trust and Trustworthy Computing - 4th International Conference, TRUST 2011, Pittsburgh, PA, USA, June 22-24, 2011. Proceedings (Lecture Notes in Computer Science)*, Jonathan M. McCune, Boris Balacheff, Adrian Perrig, Ahmad-Reza Sadeghi, M. Angela Sasse, and Yolanta Beres (Eds.), Vol. 6740. Springer, 33–47. https://doi.org/10.1007/978-3-642-21599-5_3

[22] Amir Moradi, Alessandro Barenghi, Timo Kasper, and Christof Paar. 2011. On the vulnerability of FPGA bitstream encryption against power analysis attacks: extracting keys from xilinx Virtex-II FPGAs. In *Proceedings of the 18th ACM Conference on Computer and Communications Security, CCS 2011, Chicago, Illinois, USA, October 17-21, 2011*, Yan Chen, George Danezis, and Vitaly Shmatikov (Eds.). ACM, 111–124. https://doi.org/10.1145/2046707.2046722

[23] Amir Moradi and Tobias Schneider. 2016. Improved Side-Channel Analysis Attacks on Xilinx Bitstream Encryption of 5, 6, and 7 Series. In *Constructive Side-Channel Analysis and Secure Design - 7th International Workshop, COSADE 2016, Graz, Austria, April 14-15, 2016, Revised Selected Papers (Lecture Notes in Computer Science)*, François-Xavier Standaert and Elisabeth Oswald (Eds.), Vol. 9689. Springer, 71–87. https://doi.org/10.1007/978-3-319-43283-0_5

[24] Olivier Pereira, François-Xavier Standaert, and Srinivas Vivek. 2015. Leakage-Resilient Authentication and Encryption from Symmetric Cryptographic Primitives. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*. 96–108. https://doi.org/10.1145/2810103.2813626

[25] Ed Peterson. 2015. *Leveraging asymmetric authentication to enhance security-critical applications using Zynq-7000 all programmable SoCs.* Xilinx.

[26] Emmanuel Prouff. 2005. DPA Attacks and S-Boxes. In *Fast Software Encryption: 12th International Workshop, FSE 2005, Paris, France, February 21-23, 2005, Revised Selected Papers (Lecture Notes in Computer Science)*, Henri Gilbert and Helena Handschuh (Eds.), Vol. 3557. Springer, 424–441. https://doi.org/10.1007/11502760_29

[27] Phillip Rogaway. 2004. Efficient Instantiations of Tweakable Blockciphers and Refinements to Modes OCB and PMAC. In *Advances in Cryptology - ASIACRYPT 2004, 10th International Conference on the Theory and Application of Cryptology and Information Security, Jeju Island, Korea, December 5-9, 2004, Proceedings (Lecture Notes in Computer Science)*, Pil Joong Lee (Ed.), Vol. 3329. Springer, 16–31. https://doi.org/10.1007/978-3-540-30539-2_2

[28] Sergei Skorobogatov and Christopher Woods. 2012. Breakthrough Silicon Scanning Discovers Backdoor in Military Chip. In *Cryptographic Hardware and Embedded Systems - CHES 2012 - 14th International Workshop, Leuven, Belgium, September 9-12, 2012. Proceedings (Lecture Notes in Computer Science)*, Emmanuel Prouff and Patrick Schaumont (Eds.), Vol. 7428. Springer, 23–40. https://doi.org/10.1007/978-3-642-33027-8_2

[29] Sergei Skorobogatov and Christopher Woods. 2012. In the blink of an eye: There goes your AES key. *IACR Cryptology ePrint Archive* 2012 (2012), 296. http://eprint.iacr.org/2012/296

[30] Pawel Swierczynski, Amir Moradi, David Oswald, and Christof Paar. 2015. Physical Security Evaluation of the Bitstream Encryption Mechanism of Altera Stratix II and Stratix III FPGAs. *TRETS* 7, 4 (2015), 34:1–34:23. https://doi.org/10.1145/2629462

[31] Lars Tebelmann, Michael Pehl, and Georg Sigl. 2017. EM Side-Channel Analysis of BCH-based Error Correction for PUF-based Key Generation. In *Proceedings of the 2017 Workshop on Attacks and Solutions in Hardware Security (ASHES '17)*. ACM, New York, NY, USA, 43–52. https://doi.org/10.1145/3139324.3139328

[32] S. M. Trimberger and J. J. Moore. 2014. FPGA Security: Motivations, Features, and Applications. *Proc. IEEE* 102, 8 (Aug 2014), 1248–1265. https://doi.org/10.1109/JPROC.2014.2331672

[33] Thomas Unterluggauer and Stefan Mangard. 2016. Exploiting the Physical Disparity: Side-Channel Attacks on Memory Encryption. In *Constructive Side-Channel Analysis and Secure Design - 7th International Workshop, COSADE 2016, Graz, Austria, April 14-15, 2016, Revised Selected Papers (Lecture Notes in Computer Science)*, François-Xavier Standaert and Elisabeth Oswald (Eds.), Vol. 9689. Springer, 3–18. https://doi.org/10.1007/978-3-319-43283-0_1

[34] Florian Unterstein, Johann Heyszl, Fabrizio De Santis, Robert Specht, and Georg Sigl. 2018. High-Resolution EM Attacks Against Leakage-Resilient PRFs Explained - And an Improved Construction. In *Topics in Cryptology - CT-RSA 2018 - The Cryptographers' Track at the RSA Conference 2018, San Francisco, CA, USA, April 16-20, 2018, Proceedings*. 413–434. https://doi.org/10.1007/978-3-319-76953-0_22

[35] Meng-Day (Mandel) Yu and Srinivas Devadas. 2010. Secure and Robust Error Correction for Physical Unclonable Functions. *IEEE Design & Test of Computers* 27, 1 (2010), 48–65. https://doi.org/10.1109/MDT.2010.25