Clone-Resistant Secured Booting Based on Unknown Hashing Created in Self-Reconfigurable Platform

Randa Zarrouk[1,*] , Saleh Mulhem[2], Weal Adi[1] and Mladen Berekovic[2]

[1] Institute of Computer and Network Engineering, Technical University of Braunschweig, 38106 Braunschweig, Germany
[2] Institute of Computer Engineering, University of Lübeck,23562 Lübeck, Germany
{randa.zarrouk, w.adi}@tu-bs.de and {mulhem, berekovic}@iti.uni-luebeck.de

Abstract. Deploying a physically unclonable trusted anchor is required for securing software running on embedded systems. Common mechanisms combine secure boot with either stored secret keys or keys extracted from a Physical Unclonable Function (PUF). We propose a new secure boot mechanism that is hardware-based, individual to each device, and keyless to prohibit any unauthorized alteration of the software running on a particular device. Our solution is based on the so-called Secret Unknown Hash (SUH), a self-created random secret unknown hardwired hash function residing as a permanent digital hardware-module in the device's physical layout. It is initiated in the device in a post-manufacturing, unpredictable single event process in self-reconfigurable non-volatile SoC FPGAs. In this work, we explain the SUH creation process and its integration for a device-specific secure boot. The SUH is shown to be lightweight when implemented in a sample scenario as a DM-PRESENT-based hash function. A security analysis is also presented, highlighting the different proposed sample SUH-class entropies .

Keywords: Self-reconfigurable environment, SoC FPGA, Secure-boot, Physically Unclonable Function, Unknown Secret.

1 Introduction

Our trust in electronic systems depends on trusting that the physical electronic platforms' software executes all and only the intended operations. Rigorous trustworthiness starts from the foundation and expands through the system software stacks [1]. After power-on and before executing the end-application software, the bootstrap process should ensure that no software component has been modified in an unauthorized manner. Thus, a secure boot becomes an essential requirement for ensuring a bootstrap secure and robust against possible severe attacks [2][3]. In addition to software integrity, some secure boot variants ensure software authenticity by deploying a key provisioned by the manufacturer of the System on Chip (SoC). Unfortunately, if the key is not unique to each device, then if one device's secure boot's key is compromised, the secure boot of all the other devices using that same key is also compromised. Thus,

* Supported by the German Academic Exchange Service (DAAD)'s research grand.

establishing a device-specific secure boot eliminates such risk. However, provisioning a unique key for every device can be a logistically demanding task, and even if it is uniquely generated, when stored in a Non-Volatile Memory (NVM), reconfigurable, or One-Time Programmable (OTP), the key is prone to some read-back attacks [4].

An alternative device-specific secure boot solution is to derive a device-unique key from a Physical Unclonable Function (PUF). However, despite what PUF technology promises, additional mechanisms are required to ensure the consistency of PUF's response reproducibility over time. Moreover, a non-protected strong PUF falls 2short against modeling attack.

This paper proposes using a self-created random hardwired cryptographic function, so-called Secret Unknown Hash (SUH), to establish a device-specific secure boot for an SoC environment equipped with reconfigurable hardware fabric. SUH's digital nature makes its responses inherently reproducible over time without the need for additional countermeasures, as in the PUF case. The proposed boot mechanism requires SoC with a self-reconfigurable non-volatile (embedded) Field-Programmable Gate Array ((e)FPGA) fabric as a suitable platform for the SUH creation. SUH can be created during a post-manufacturing unpredictable single event process within the device itself, eliminating the need to trust the SoC manufacturer for provisioning a device-unique secret. The self-created SUH hardware module is distributed in the unoccupied FPGA fabric areas, making it harder for adversary reach. The self-reconfigurable property is still not existent in the non-volatile FPGA landscape, but it is expected to submerge soon due to its demanded properties.

2 Summary of Contributions:

In this work:

- We first introduce SUH-based secure boot (Table. 1, Fig.1), and we show state-of-the-art secure boot hardware-based solutions such as PUF-based secure boot.
- We enumerate the PUF limitation (Section 3).
- We show how self-reconfigurable non-volatile SoC FPGA enables SUH creation and explain how SUH ensures a device-specific secure boot. (Section 4)
- As proof of the concept, we present a possible sample implementation variant of a SUH based on a lightweight DM-PRESENT and evaluate its security. (Section 5)
- We introduce a generalized comparison between two cases of the class entropy of SUH. (Table 4)

Motivation: SUH Vs. PUF secure boot. To highlight the importance of the proposed secure-boot mechanism, we summarize the difference between the state-of-the-art PUF-based secure boot with the proposed SUH-secure boot. We propose Table 1 and Fig. 1 to show the characteristics, similarities, and differences of every solution. The hash-based secure boot is to be noted as a reference point since it is the most trivial secure boot mechanism. As we show in Table.1, both PUF-based and SUH-based ensure software integrity and authentication along with electronic device authentication. Meanwhile, unlike PUF, our proposal using SUH has the advantage of providing a

consistent response reproducibility without additional measures, being keyless, and being created in a post-manufacturing setting.

Table 1. Characteristics, similarities, and differences between SUH-based secure boot (our work), hash-based secure boot, and PUF-based secure boot.

Characteristics	Secure boot based on:		
	A hardwired traditional hash function alone	PUF + A hardwired traditional hash function (Fig. 1(a))	**SUH** ***(introduced in this work)*** (Fig. 1(b))
Offers data integrity	yes	yes	**yes**
Offers data authentication	no	yes	**yes**
Offers device authentication	no	yes	**yes**
Requires additional measurements for a consistent response reproducibility	no	yes	**no**
Uses a key	no	yes	**no**
Created in Post-manufacturing setting	no	no	**yes**

Fig.1 shows how the proposed SUH-based solution replaces the PUF and the hash function while providing a software executable's digest that is unique and individual to the electronic device where the SUH is self-created.

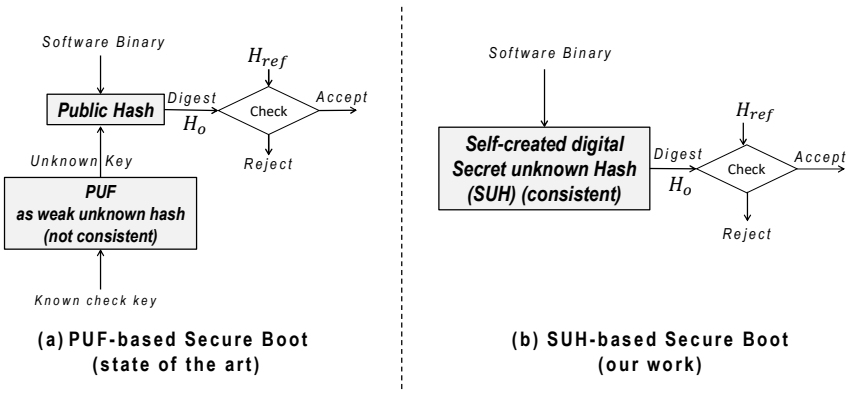

Fig. 1. (a) state of the art of device-specific secure boot and (b) the proposed concept of the device-specific secure boot.

3 Background & Related Work

3.1 Secure boot

Secure boot is one of the Trusted Computing (TC) mechanisms used to establish a chain-of-trust that starts from a root-of-trust and ends with the last executed software. There are a variety of authentication schemes to establish a secure boot: hash-based, Message Authentication Code (MAC)-based, and signature-based [5][6]. In this work, we refer to a device as an electronic unit. Two devices with identical models, identical brands, and identically produced are considered two different electronic units.

Device-specific secure-boot. A software digest is unique to the device if a unique device Identity (ID) is included in the hash computation [5][6]. By generating a device-specific hash value (digest) of the software binary to be authenticated during the boot process, the secure boot is made unique to that device. The benefit is that if the adversary could determine a different pre-image for that device's authentic software digest to use it as malicious software, it does not affect the secure boot of other devices running the same authentic software.

PUF-based Secure boot. Several secure boot scenarios utilizing PUF as a key source exist. Some aim to secure the volatile SoC FPGA bitstream, and others aim to secure the processor. The used key is unknown as it is extracted from the PUF and links the software or the bitstream to that specific device. For instance, in [7], Twisted Bistable Ring PUF (TBR-PUF) generates system keys to protect and authenticate the firmware and the operating system using the Advanced Encryption Standard (AES) cipher in a Galois/Counter Mode (GCM). In [8], a diode-connected nMOS transistors PUF generates a 128-bit unique key to be used with the PRESENT cipher and the MAC algorithm to ensure the firmware's confidentiality, authenticity, and integrity. In [9], a Hardware-Embedded Delay PUF (HELP) is implemented into a Xilinx Zynq 7020 SoC FPGA fabric to generate a key encrypting/decrypting a second bitstream containing the programmable logic and the software to be executed. In [10], a Configurable response-length Linear Feedback Shift Register (LFSR) based PUF (CoLPUF) generates the signing keys for a proposed secure boot framework in a RISC-V SoC environment.

3.2 PUFs and their limits

PUF can be considered the physical equivalent of a one-way hash function [11][12]. The significant random physical components constituting the PUF make it inherently unclonable. Several designs of PUF were proposed, and based on their Challenge Responses Pairs (CRP), researchers have classified them into two categories; *Strong PUF* supports a large number CRPs with a space that scales exponentially relatively to the PUF size and *Weak PUF*, which is, on the contrary, supports a limited number of CRPs in a polynomial or a linear space growth [13]. Although PUF can be a compelling solution for device-specific security, not all PUF are "good enough" for some applications' required security level. A PUF should be selected carefully based on some criteria such as uniqueness, randomness, and diffuseness, besides the limits enumerated below:

Predictability. *"An unprotected strong PUF able to resist modeling would be a real breakthrough"*[13]. Strong PUFs are prone to modeling attacks [14]. Thus, they need protection by integrating additional cryptographic primitives [13]. Nevertheless, some protected, strong PUFs still fail against different types of modeling attacks. For example, the Bistable Ring (BR) PUF was improved twice and still exhibits vulnerabilities against modeling attacks. As a first improvement, the TBR-PUF was proposed [15]. The second amelioration suggests XORing at least 4 BRs, which has shown the BR is immune against only one type of modeling attack: the Support-Vector Machines (SVM ML) attack [16].

Response Reproducibility. PUF being an analog function, its responses are not accurately reproducible due to aging, wear out, voltage variation, and environmental conditions such as temperature and noise. An adversary can take advantage of this vulnerability to attack the PUF [17]. A post-processing function must condition the raw PUF responses into high-quality cryptographic keys and ensure their reproducibility. Such functions can be Error Correction Code (ECC), Fuzzy Extractor (FE), or Helper data [13]. Besides the additional implementation costs of such functions, some FE are risky to use when the PUF response entropy is low [18].

Response length expansion. Some PUFs provide a small response space of a few bits or, most likely one-bit length [13]. If a PUF is used to generate a key for an encryption algorithm, then the required key length n should exceed n=128 bits. Therefore, such PUFs require additional methods for expanding the response space.

3.3 Clone-Resistant Random Secret Unknown Hardwired Crypto-Functions

PUF became a famous example of exploiting the differences in electronic devices' physical properties to serve as a physical-based security solution. Alternatively, the authors in [4] proposed using uncontrollable permanent differences created within the device after manufacturing as a bio-inspired electronic DNA (e-DNA). The e-DNA is self-created, non-repeatable, hard to clone, and easily provable with the ability to evolve during the device's lifetime and spread all along in the device's hardware. Subsequently, a clone-resistant random secret unknown hardwired dynamic/evolving cryptographic functions were proposed in [19] as a device's e-DNA.

In our work, we exploit the concept of the clone-resistant self-created random hardwired cryptographic function, being a hash function, in a reconfigurable environment focusing on its creation and deployment without covering the additional sophisticated characteristics such as the evolving property and the diffusion of the e-DNA in the whole device's body.

4 SUH-based Device-Specific Secure Boot

The proposed secure boot mechanism relies on a self-created SUH-hardware module in a post-manufacturing single even process. A hash function is chosen as a building block for the proposed mechanism since its output is an integrity token used in the

secure boot process. This hash function is randomly selected from a huge class of hash functions such that the probability of guessing which hash has been selected is negligible—and this hash is secretly self-implemented in the reconfigurable hardware's fabric in hard to predict areas, making it a digital unknown secret for that unique device. PUF itself can be regarded as an unpredictable and unknown physical one-way function [12]. Thus, SUH can be considered as a possible digital replacement to PUF technology.

4.1 SUH Creation

A dedicated temporary software package, so-called Genie-H, generates the SUH within the device in a secure environment. Genie-H is then eliminated irreversibly after its task completion, ensuring that the SUH is an unknown secret since its creator has disappeared. The SUH-generation requires an in-device True Random Number Generator (TRNG). The creation steps are depicted in Fig. 2 as follows: A Trusted Authority (TA_1) injects the temporal Genie-H into the SoC FPGA device SoC_u, during a one-time random single event operation. A huge class of hash functions $\{H_1, H_2,.., H_t\}$ of size t is generated, where $t \rightarrow \infty$ in the best case. Genie-H using a TRNG randomly selects one hash function H_j from $\{H_1, H_2,..., H_t\}$. Finally, Genie-H is completely and irreversibly deleted. What remains in the SoC FPGA is a SUH-choice that nobody knows. A TA_2, which can be different from TA_1, enrolls the SUH in a secure environment. The TA_2 stimulates the SUH by a set of challenges, Messages (M), and stores the corresponding responses Digests (D) as CRPs of the enrolled device in a secret M_i/D_i record.

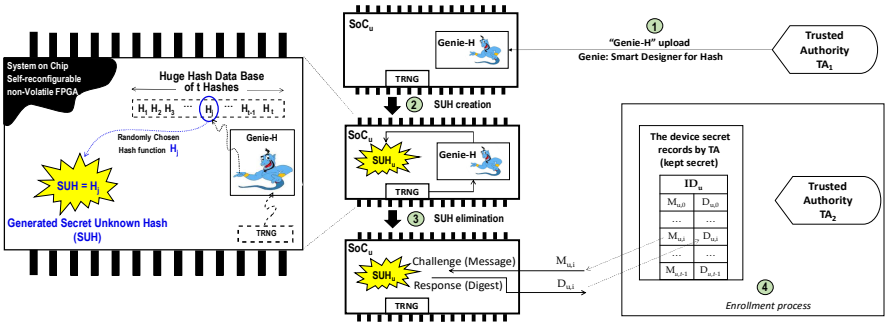

Fig. 2. Creation Steps of SUH on a self-reconfigurable non-volatile SoC FPGA.

Eventually, the only one who knows about which SUH is selected is the already vanished Genie-H. In this setting, nobody knows about the selected SUH even though the hash generation method may be known or published. This creation process exempts the manufacture from provisioning and knowing anything about the device-specific secret for secure boot. It is hard to clone a device whose secret is unknown. The self-created SUH hardwired module is programmed in the FPGA fabric such that it is randomly distributed on the *leftover areas*. Finally, any further reconfiguration/modification is hard-locked forever by burning some *"last fuse."* We refer leftover areas to the FPGA fabric spaces that remain free after primary programming it with the main intended/commercial applications (e.g., video & image processing, medical, telecom, cloud computing). On the contrary of storing a key in non-volatile memory or storing

a design in a well-known FPGA fabric location, SUH is diffused in the leftover areas, which can be disjointed, making it harder to recover SUH by some physical probing points attacks [20].

SUH as a Unique Device ID. Aside from the secure boot authentication, SUH is also used for the device's ID verification via the deployment of the device's corresponding stored M_i/D_i: The verifier asks TA_2 for a M_i/D_i pair from the device's secret record. Then stimulates the device with M_i, and the SUH responds with D_i'. If $D_i'=D_i$, the verifier considers the device as authentic.

4.2 System Model

The SUH creation requires a platform equipped with SoC and a non-volatile (e)FPGA fabric. The FPGA fabric should be able to be self-reconfigurable to enable in-device post-manufacturing secret creation. While non-volatile SoC FPGAs exist, the option of internally self-reconfiguring the non-volatile FPGA fabric is still a futuristic feature. Meanwhile, we implement a SUH candidate on the flash-based SoC FPGA SmartFusion2 since it provides the non-volatility trait and incorporates Differential Power Analysis (DPA) countermeasures to protect the bitstream key(s) to be discovered using side-channel analysis [21]. The generic boot sequence of the SmartFusion2 is shown in Fig. 3. The first piece of code that boots after power-on is the Boot-ROM code defined by the manufacturer and stored as a Metal-Read Only Memory (ROM). The reference digests and executables can be stored in the embedded NVM's reconfigurable ROM sections. In our work, we assume that during the secure boot authentication, after digest generation, the computed digest with the reference digest is compared either within: a protected processor or the FPGA Fabric. In an environment equipped with an OTP memory, the user bootloader code can replace the Boot-ROM code to ensure that the SoC FPGA boot sequence is fully user-defined.

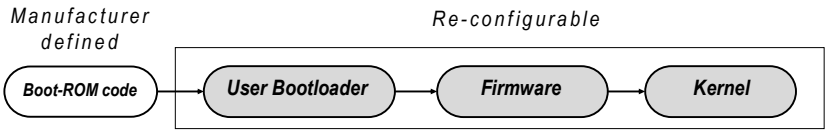

Fig. 3. A generic boot-sequence on the SmartFusion2 SoC FPGA

4.3 Secure Boot Operational Scenario

Setting up the SoC FPGA requires an already in-device self-created SUH, as previously depicted in Fig.2.

System set-up. Fig. 4.(A) demonstrates the system set-up phase. The user needs to upload the different software executables such as the user bootloader, the firmware, and the kernel. Next, the user bootloader runs in a *Set-Up Mode* to generate the firmware's reference digests (D_F) and the kernel's reference digest (D_K). The storage act can be done internally by the bootloader in a self-reconfigurable device since internal

reconfiguration is expected from such technology. After the set-up phase, the system is ready to be used.

Secure boot in action. Fig. 4.(B) illustrates the steps after power-on, and after that, the Boot-ROM has passed the system's control to the user bootloader. The steps are as follows:

a) The bootloader in execution:
(1) The firmware binary is fetched from the eNVM to the SUH to generate firmware digest' (D_F'). **(2)** DF' is compared with DF. **(3)** If DF=DF', then inputs of the SUH are also equal, and thus the firmware's binary has not been altered. Then, the bootloader passes the system's control to the firmware.

b) The firmware in execution:
(4) The kernel binary is fetched to the SUH input, and the correspondent digest' (DK') is generated. **(5)** DK' is compared with the reference digest DK. **(6)** If DK=DK', the firmware passes the system's control to the kernel to boot.

The chain-of-trust can be further extended due to many possible SUH digests ($2^{n/2}$). Therefore, SUH-CRPs can support the OS kernel to authenticate system services, device drivers, and other applications.

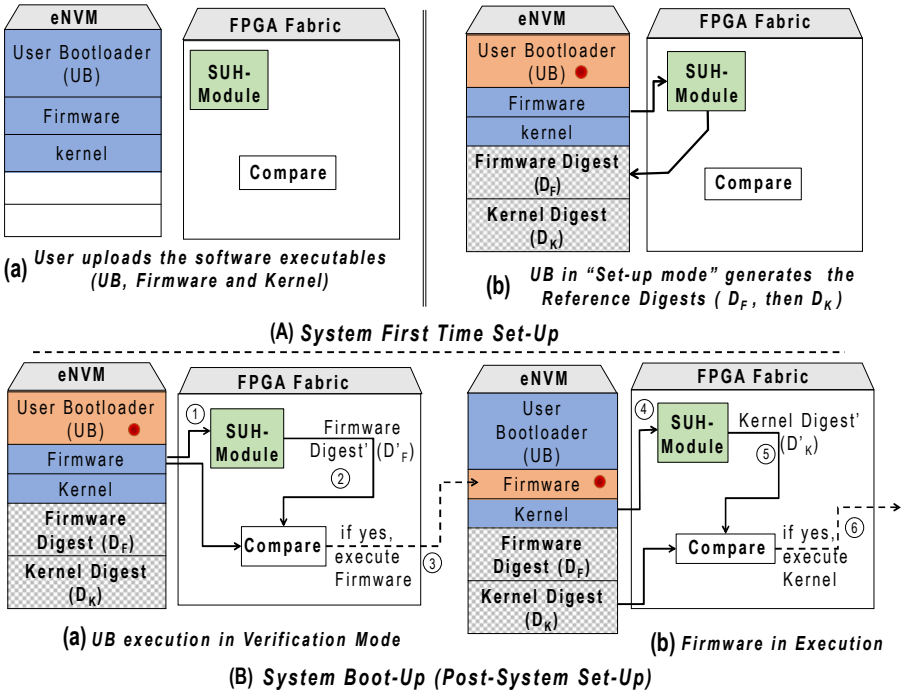

Fig. 4. Secure Boot Sequence deploying SUH

4.4 Adversary Model:

The same system can be a target to different attacks with different aims. Some adversaries may try to replace the authentic software, while others may try to impersonate/clone the device ID. In this section, we consider two attack scenarios, the first, where the adversary aims to run an unauthorized version of the software executable, and the second is where the adversary aims to impersonate the device.

Unauthorized Modification of the Software Executable. An adversary seeks that the chain-of-trust does not detect the maliciously modified version of the original software. This goal can be reached either by exploiting the hardware's storage units or exploiting the hash function's limitations. In the first case, the adversary should replace both the software and its digest successfully. Such attacks are not considered in this work because the verification digest is always stored in a tamper-resistant memory in the secure boot context. It is assumed hard for the adversary to change such memory's content. In the second case, the adversary makes sure to find a modified software version that corresponds to the authentic reference digest; the adversary should find a pre-image of the reference digest, which is different from the authentic software executable. Such attacks can be performed based on the hash function limitations. We consider this case in our model through two adversaries with different capabilities:

Unbounded Adversary (UA): can read the eNVM content and challenge the embedded SUH with chosen inputs and observe the output/digest. In this case, the adversary can consider the SUH as a black box.

Bounded Adversary (BA): can read the eNVM content. However, BA can only observe the SUH input and output without challenging the SUH.

Cloning Attack. An adversary tries to clone the device ID to impersonate that device using the authentic ID on a fraud device. The impersonation attack on a device with integrated SUH means that the adversary can somehow identify the SUH. In this adversary model, we suppose that the SUH creation is maintained in a secure environment. Any attempt to clone the SUH is conducted after its creation, implying that the adversary has no access or knowledge of the TRNG-value chosen by Genie-H to the SUH's random selection. The cloning complexity is then proportional to the number of possible hash functions t if the Genie-H is published. If the Genie-H is not published, then the cloning complexity is proportional to all possible hash functions. In our particular sample example, when using an n-to-n bit cipher for constructing the hash function, the choice space becomes $2^n!$.

5 SUH-Sample Implementation and Variants

Designing a cryptographic hash function is challenging, let alone constructing a huge class of hash functions. Meanwhile, a practical approach to generate such a class is by utilizing a block cipher-based iterated hash function [22][23]. In the following, we introduce one possible variant of a SUH implementation, see Fig. 5, where a PRESENT-like cipher is deployed as a building block of an iterated hash function.

5.1 PRESENT-like based SUH

PRESENT is a lightweight 64-bit iterated block cipher with two key-length possibilities: an 80-bit key (PRESENT-80) and a 128-bit key (PRESENT-128). The cipher is composed of an adding key, a substitution layer, and a permutation layer [24]. The substitution layer, a nonlinear transformation, is performed via 16 Substitution Boxes (S-Boxes) which are mappings of a 4-bit to 4-bit. A Golden S-boxes (GS) is an S-box exhibiting ideal cryptographic properties [25]: **(1)** Differential Characteristic probability DC=¼, **(2)** Linear approximation probability LC=¼, **(3)** Branch Number BN=3. PRESENT's S-box has these properties; thus, replacing the PRESENT's S-box with any GS maintains the PRESENT cipher's same security level. We exploit this property to define a SUH variant as a PRESENT-like based hash function using randomly selected GSs. Different compact PRESENT-based hash functions were introduced in [26]: **(1)** DM-PRESENT-80 and DM-PRESENT-128: deploying a single block cipher to obtain a 64-bit digest. **(2)** H-PRESENT-128: is a double-block-length hash function for a 128-bit digest. **(3)** C-PRESENT-192: is a triple-block-length hash function for a 192-bit digest. For instance, a SUH variant can be a DM-PRESENT where the S-box is preplaced with randomly chosen GS. Genie-H is responsible for randomly choosing 16 GS to populate the S-box layer. Genie-H can distribute the chosen GS in a random order within the S-Box layer and can, in addition, randomly select one of the secure 12 schemes of the block cipher-based hash construction schemes analyzed in [27].

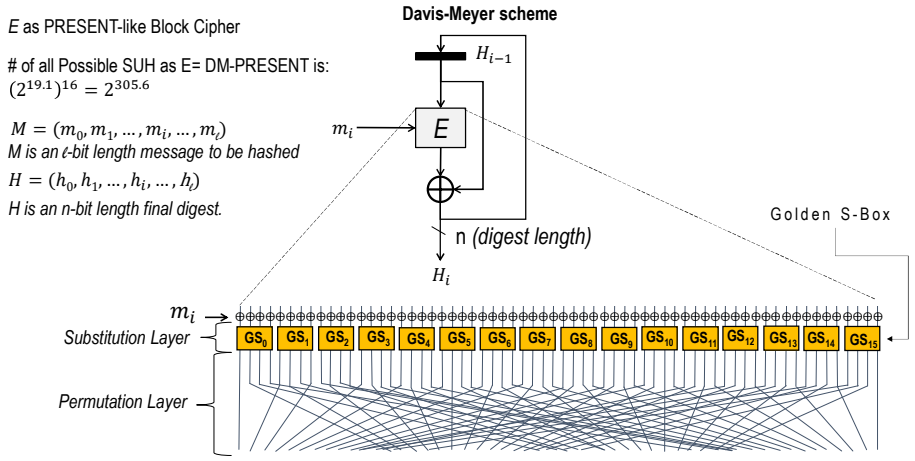

Fig. 5. A sample of a SUH construction based on Davies-Meyer Scheme

5.2 DM-PRESENT-like Hash Implementation

SmartFusion2 SoC FPGA fabric logic elements have 4-input LUTs [28] convenient to the 4-bit to 4-bit S-boxes of the PRESENT-like based SUH variant. Our sample implementation of DM-PRESENT-80-like has consumed in a SmartFusion2 FPGA 526 LUTs and 331 Flip-Flops. Table 2 shows the relatively low percentage of the consumed

resources on the different SmartFusion2 SoC FPGAs. The security measures must be implemented, consuming as little hardware as possible since it is usually a protection measure that accompanies the main application.

Table 2. Hardware resource consumption of SUH-PRESENT-80-LIKE based on SmartFusion2 Soc FPGA family

SmartFusion®2 SoC FPGA family	4-input LUTs	Flip-Flops
M2S005	8.67%	5.46%
M2S010	4.35%	2.73%
M2S025	1.89%	1.19%
M2S050/ M2S060	0.93%	0.58%
M2S090	0.61%	0.38%
M2S150	0.35%	0.22%

5.3 Security Analysis

We discuss the efforts required for an adversary to clone SUH and to defeat its digest. Every electronic device is susceptible to side-channel attacks if no countermeasure is applied. A detailed evaluation of the side-channel attack on our design is out of the scope of this paper. We assume that the emerging self-reconfiguring mechanisms in "non-volatile" FPGA technologies would offer physical properties prohibiting reaching the bitstream after irreversibly locking the reconfiguration process. Physical side-channel attacks can only be fruitfully discussed first after the self-reconfiguration infrastructure is implemented and exists in future practical devices.

Clone-resistant device ID. For example, let us consider that the SUH is constructed based on a DM-PRESENT-80-like:

- If 16 GS-Boxes are selected randomly out of $2^{19.1}$ [25], it results in $(2^{19.1})^{16} = 2^{305.6}$ SUHs.
- If, in addition, an iterative hash's compression function is selected randomly out of $12 \approx 2^{3.5}$ [27], then the hash class cardinality is $t = 2^{305.6}.\ 2^{3.5} \approx 2^{309}$ SUHs. 2^{309} presents the computational cloning attack complexity of a DM-PRESENT-like based SUH.

The attack complexity on cloning the other possible PRESENT-like-based SUH is presented in Table 3.

Table 3. Computational Cloning Attack Complexity of the PRESENT-like based SUH

SUH type	Number of used Golden S-Boxes	Computational Cloning attack Complexity (# Operations)
DM-PRESENT-80-like DM-PRESENT-128-like	16	$(21^{9.1})^{16} = 2^{305.6}.\ 2^{3.5} \approx 2^{309}$
H-PRESENT-128-like	32	$(2^{19.1})^{32} = 2^{611.2}.\ 2^{3.5} \approx 2^{614}$
C-PRESENT-192-Like	96	$(2^{19.1})^{96} = 2^{1833.6}.\ 2^{3.5} \approx 2^{1837}$

The entropy of SUH class. Table 4 addresses the entropy differences that a SUH can offer depending on whether Genie-H is publically known or not. When the Genie-H is not public, all possible hash mappings for a hash function with an m-bit input and an n-bit output are $|H| = 2^{m2^n}$, thus the upper bound of the entropy of |H| in this case, is $m2^n$. In the case that the Genie-H is known, the entropy of |H| is $log_2(t)$, where t is the cardinality of the huge hash class, for instance, in the proposed sample case PRESENT-based SUH variant, |H| ≥ 309.

Table 4. The entropy of possible SUHs as a Clone-Resistant Identities for Different Devices

SUH Class Entropy	Genie-H is Public	Genie-H is not public
SUH 	$log_2 t$	$m2^n$

Authentic Software Digest Protection. Let us again consider the case that the SUH is constructed based on a DM-PRESENT-80-like which produces 64-bit digest length; the computational security on the hash digest is as follow:

Case of Unbounded-Adversary Model. The UA can examine the SUH variant as a black box to conduct an online guessing attack. While under such a model, the SUH security level is equivalent to a traditional hash function, SUH still offers a clone-resistant device ID and a device-specific secure boot which, in the worst-case scenario, where one specific device's hash function is defeated, not all devices secure boot would be defeated.

Case of a Bounded-Adversary model. To defeat the hash function, BA should first determine which SUH is randomly implemented, and this is equivalent to cloning the SUH. The results from Table 4 imply that SUH's successful prediction on DM-PRESENT-80-like construction requires 2^{309} search cycles. In this case, the required computation attack complexity on SUH's digest additionally demands

- $2^{309}.2^{64}=2^{373}$ computations for a successful pre-image attack,
- $2^{309}.2^{32}=2^{341}$ computations for a successful collision attack,
- and $2^{309}.2^{64}=2^{373}$ computations for a successful second pre-image attack.

Table 5. Evaluation of the computational security on the digest of PRESENT-like-based SUHs

SUH type	Hash (bit)	Number of used Golden S-Boxes	UA-Attack [29]			BA-Attack		
			Pre.	Coll.	2^{nd} Pre.	Pre.	Coll.	2^{nd} Pre.
DM-PRESENT-80-like	64	16	2^{64}	2^{32}	2^{64}	2^{373}	2^{341}	2^{373}
DM-PRESENT-128-like H-PRESENT-128-like	128	32	2^{128}	2^{64}	2^{64}	2^{742}	2^{678}	2^{678}
C-PRESENT-192-Like	192	96	2^{192}	2^{96}	2^{192}	2^{2029}	2^{1933}	2^{2029}

Under a BA model, for the same hash-bit length, in this example, 64-bit, the SUH's security level is way beyond the traditional hash function's security. The attack

complexity on the software's digest of other possible PRESENT-like-based SUH is presented in Table 5. The defined UA- and BA-based attack scenarios are only possible online since the electronic device's SUH is unknown. It is only when an adversary succeeds in cloning the SUH that these attacks are possible offline.

6 Conclusion

This paper introduces a new device-specific/unit-individual secure boot mechanism for SoC equipped with self-reconfigurable non-volatile (e)FPGA providing a hardware-based trusted foundation for software execution in electronic devices. The introduced approach deploys an in-device self-created Secret Unknown Hash (SUH) that generates a unique software binary digest for each individual electronic device. We showed that SUH being keyless, hardwired, unknown, randomly generated, can replace other solutions to generate device-specific software digests such as using a PUF and a traditional hash function. SUH has the advantage over PUF of ensuring a consistent response reproducibility without additional error correction codes. Comparing to the traditional hash function, SUH has the advantage of providing a clone-resistant unique device ID. We explained how to construct a SUH using a cipher-based hash function such as the PRESENT-like cipher as proof of concept. We showed a SUH sample implementation to be relatively lightweight. Finally, we proved that the proposed SUH offers a high clone-resistance entropy and a high device-specific digest computational security. These particular findings are further beneficial for remote authentication of a fully user-controlled secure booting of devices deploying unknown secrets. The user is responsible for individualizing the device's clone-resistant unknown secret after its manufacturing instead of trusting the manufacturer for exclusively provisioning it, which motivates using on a large scale the hopefully emerging SoC non-volatile (e)FPGAs with the enabled option of self-reconfigurability

References

1. Arbaugh, W.A., Farber, D.J., Smith, J.M.: A Secure and reliable bootstrap architecture. In: Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy. pp. 65–71. IEEE (1997). https://doi.org/10.1109/secpri.1997.601317.
2. Langner, R.: Stuxnet: Dissecting a cyberwarfare weapon. IEEE Security and Privacy. 9, 49–51 (2011). https://doi.org/10.1109/MSP.2011.67.
3. Quarta, D., Pogliani, M., Polino, M., Maggi, F., Zanchettin, A.M., Zanero, S.: An Experimental Security Analysis of an Industrial Robot Controller. In: Proceedings - IEEE Symposium on Security and Privacy. pp. 268–285. IEEE (2017). https://doi.org/10.1109/SP.2017.20.
4. Skorobogatov, S.P.: Semi-invasive attacks - a new approach to hardware security analysis. , Cambridge (2005).
5. Sanwald, S., Kaneti, L., St, M., Martin, B.: Secure Boot Revisited : Challenges for Secure Implementations in the Automotive Domain. In: 17th escar Europe : embedded security in cars. pp. 113–127. Ruhr-Universität Bochum, Universitätsbibliothek (2019). https://doi.org/10.13154/294-6662.

6. Bhat, A.: Secure Boot , Chain of Trust and Data Protection. In: Embedded World Conference 2019 (2019).
7. Jacob, N., Wittmann, J., Heyszl, J., Hesselbarth, R., Wilde, F., Pehl, M., Sigl, G., Fischer, K.: Securing FPGA SoC configurations independent of their manufacturers. In: 30th IEEE International System-on-Chip Conference (SOCC). pp. 114–119. IEEE (2017). https://doi.org/10.1109/SOCC.2017.8226019.
8. Muller, K.U., Ulrich, R., Stanitzki, A., Kokozinski, R.: Enabling Secure Boot Functionality by Using Physical Unclonable Functions. In: PRIME 2018 - 14th Conference on Ph.D. Research in Microelectronics and Electronics. pp. 81–84. IEEE, Prague, Czech Republic (2018). https://doi.org/10.1109/PRIME.2018.8430370.
9. Owen Jr., D., Heeger, D., Chan, C., Che, W., Saqib, F., Areno, M., Plusquellic, J.: An Autonomous, Self-Authenticating, and Self-Contained Secure Boot Process for Field-Programmable Gate Arrays. Cryptography. 2, 15 (2018). https://doi.org/10.3390/cryptography2030015.
10. Haj-Yahya, J., Wong, M.M., Pudi, V., Bhasin, S., Chattopadhyay, A.: Lightweight Secure-Boot Architecture for RISC-V System-on-Chip. In: 20th International Symposium on Quality Electronic Design (ISQED). pp. 216–223. IEEE, Santa Clara, CA, USA (2019). https://doi.org/10.1109/ISQED.2019.8697657.
11. Guajardo, J.: Physical Unclonable Functions (PUFs). In: Encyclopedia of Cryptography and Security. Springer, Boston, MA (2011). https://doi.org/10.1007/978-1-4419-5906-5.
12. Pappu, R., Recht, B., Taylor, J., Gershenfeld, N.: Physical One-Way Functions. (2002). https://doi.org/10.1126/science.1074376.
13. Delvaux, J., Peeters, R., Gu, D., Verbauwhede, I.: A survey on lightweight entity authentication with strong pufs. ACM Computing Surveys. 48, (2015). https://doi.org/10.1145/2818186.
14. Rührmair, U., Sölter, J.: PUF modeling attacks: An introduction and overview. In: 2014 Design, Automation & Test in Europe Conference & Exhibition (DATE). pp. 1–6. IEEE, Dresden, Germany (2014). https://doi.org/10.7873/DATE2014.361.
15. Schuster, D., Hesselbarth, R.: Evaluation of bistable ring PUFs using single layer neural networks. In: Holz T., Ioannidis S. (eds) Trust and Trustworthy Computing. Trust 2014. Lecture Notes in Computer Science, vol 8564. pp. 101–109. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08593-7_7.
16. Xu, X., Rührmair, U., Holcomb, D.E., Burleson, W.: Security evaluation and enhancement of Bistable Ring PUFs. In: (2015) Security Evaluation and Enhancement of Bistable Ring PUFs. In: Mangard S., Schaumont P. (eds) Radio Frequency Identification. RFIDSec 2015. Lecture Notes in Computer Science, vol 9440. pp. 3–16. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24837-0_1.
17. Roelke, A., Stan, M.R.: Attacking an SRAM-Based PUF through wearout. In: 2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI). pp. 206–211. IEEE, Pittsburgh, PA, USA (2016). https://doi.org/10.1109/ISVLSI.2016.68.
18. Koeberl, P., Li, J., Rajan, A., Wu, W.: Entropy loss in PUF-based key generation schemes: The repetition code pitfall. In: 2014 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST). pp. 44–49. IEEE, Arlington, VA, USA (2014). https://doi.org/10.1109/HST.2014.6855566.

19. Adi, W.: Autonomous Physical Secret Functions and Clone-Resistant Identification. International Journal of Advanced Science and Technology. 14, (2010).
20. Wollinger, T., Paar, C., Guajardo, J.: Security on FPGAs: State-of-the-Art Implementations and Attacks. ACM Transactions on Embedded Computing Systems. 3, 534–574 (2004). https://doi.org/10.1145/1015047.1015052.
21. Microsemi: User Guide SmartFusion2 and IGLOO2 FPGA Security and Best Practices. (2017).
22. Lai, X., Massey, J.L.: Hash functions based on block ciphers. In: (1993) Hash Functions Based on Block Ciphers. In: Rueppel R.A. (eds) Advances in Cryptology — EUROCRYPT' 92. EUROCRYPT 1992. Lecture Notes in Computer Science, vol 658. pp. 55–70. Springer, Berlin, Heidelberg (1993). https://doi.org/10.1007/3-540-47555-9_5.
23. Schläffer, M.: Cryptanalysis of AES-Based Hash Functions, (2011).
24. Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A.: PRESENT : An Ultra-Lightweight Block Cipher. In: (2007) PRESENT: An Ultra-Lightweight Block Cipher. In: Paillier P., Verbauwhede I. (eds) Cryptographic Hardware and Embedded Systems - CHES 2007. CHES 2007. Lecture Notes in Computer Science, vol 4727. pp. 450–466. Springer, Berlin, Heidelberg. (2007). https://doi.org/10.1007/978-3-540-74735-2_31.
25. Saarinen, M.J.O.: Cryptographic analysis of all 4 × 4-bit S-boxes. In: (2012) Cryptographic Analysis of All 4 × 4-Bit S-Boxes. In: Miri A., Vaudenay S. (eds) Selected Areas in Cryptography. SAC 2011. Lecture Notes in Computer Science, vol 7118. pp. 118–133. Springer, Berlin, Heidelberg. (2012). https://doi.org/10.1007/978-3-642-28496-0_7.
26. Bogdanov, A., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J.B., Seurin, Y.: Hash functions and RFID tags: Mind the gap. In: (2008) Hash Functions and RFID Tags: Mind the Gap. In: Oswald E., Rohatgi P. (eds) Cryptographic Hardware and Embedded Systems – CHES 2008. CHES 2008. Lecture Notes in Computer Science, vol 5154. pp. 283–299. Springer, Berlin, Heidelberg. (2008). https://doi.org/10.1007/978-3-540-85053-3_18.
27. Preneel, B., Govaerts, R., Vandewalle, J.: Hash functions based on block ciphers: A synthetic approach. In: (1994) Hash functions based on block ciphers: a synthetic approach. In: Stinson D.R. (eds) Advances in Cryptology — CRYPTO' 93. CRYPTO 1993. Lecture Notes in Computer Science, vol 773. pp. 368–378. Springer, Berlin, Heidelberg. (1994). https://doi.org/10.1007/3-540-48329-2_31.
28. Microsemi: SmartFusion2 SoC FPGA Fabric User's Guide. (2017).
29. Bogdanov, A., Knezevie, M., Leander, G., Toz, D., Varıcı, K., Verbauwhede, I.: spongent : A Lightweight Hash Function. In: (2011) spongent: A Lightweight Hash Function. In: Preneel B., Takagi T. (eds) Cryptographic Hardware and Embedded Systems – CHES 2011. CHES 2011. Lecture Notes in Computer Science, vol 6917. Springer, Berlin, Heidelberg.