Hardware-Enabled Secure Firmware Updates in Embedded Systems

Solon Falas, Charalambos Konstantinou, Maria K. Michael

Hardware-Enabled Secure Firmware Updates in Embedded Systems

Solon Falas, Charalambos Konstantinou, and Maria K. Michael

Dept. of Electrical and Computer Engineering, KIOS Research and Innovation Centre of Excellence, University of Cyprus
FAMU-FSU College of Engineering, Center for Advanced Power Systems, Florida State University
{falas.solon,mmichael}@ucy.ac.cy
ckonstantinou@fsu.edu

Abstract. *Firmware updates on embedded systems are essential for patching vulnerabilities and improving the functionality of devices. Despite the importance of firmware updates, manufacturers and firmware developers often consider firmware security as a secondary task. As a result, firmware often turns into an alluring target for adversaries to inject malicious code into embedded devices. In this work, we present a framework that supports secure and fast firmware update delivery with minimal downtime on embedded devices. The proposed framework makes use of cryptographic primitives implemented on hardware in addition to the device's intrinsic physical characteristics acting as digital authentication fingerprints. Our implementation ensures firmware authenticity, confidentiality, and integrity. A proof-of-concept design is emulated on FPGA demonstrating high performance, strong security guarantees, and minimal hardware overhead.*

Keywords: Embedded systems, firmware updates, hardware security.

1 Introduction

Embedded devices are increasingly integrated into several Cyber-Physical System (CPS) domains such as Industrial Control Systems (ICS), home and automation networks, wireless sensing services, automobiles, etc. The deployment of these devices in mission-critical environments, however, introduces security challenges that require a different approach compared to general-purpose computing systems security [30]. Embedded systems are highly constrained in terms of performance and resources, therefore it is typically not realistic to employ similar security methods as those in general-purpose systems. Embedded device manufacturers and CPS integrators have to incorporate specialized security measures to protect these devices, and thus the CPS application they support. Studies, however, have shown that these security strategies are not a priority for enterprises [22]. This is evident by the growing number of attack incidents related to microprocessor-based embedded devices [38].

A prominent example of attacks against embedded systems is the 2010 Stuxnet incident. This computer worm targeted Programmable Logic Controllers (PLCs) modifying their firmware code to perform malicious actions while also hiding its presence. Stuxnet leveraged zero-day[1] exploits in the PLCs firmware to take control over critical machinery in a nuclear power plant facility, leading to catastrophic failure [20]. Another example is the 2015 attack on Ukraine's power grid [50]. This Advanced Persistent Threat's (APT) objective was to launch a Denial-of-Service (DoS) attack on the power distribution entity's call center, disabling the Uninterruptible Power Supplies (UPS) for the control centers, and corrupting the firmware of Human-Machine Interfaces (HMI) found in Remote Terminal Units (RTU) and serial-to-Ethernet port servers. Due to the firmware corruption, circuit breakers were disabled and a power outage occurred that the customers could not report since the call centers were overloaded. A more recent large-scale attack, the Mirai botnet, was able to turn networked devices into controlled bots [23]. Mirai identified devices connected to the internet and tried to log into them using a table of more than 60 common factory default usernames and passwords. It then proceeded to infect them with the Mirai malware. The devices continued to function correctly except for some occasional sluggishness and increased bandwidth usage. These zombie devices were then directed to certain web-services to overwhelm anti-DoS software and make the service inaccessible. The fact that hundreds of thousands of networked embedded devices still use default credentials is very concerning; the effectiveness of these attacks indicates that embedded system security can no longer be an afterthought.

Firmware in embedded systems is the dedicated software that acts as an abstraction layer between bare metal hardware and software. Firmware is often residing in read-only memory, playing the role of the "operating system" in an embedded device [4,28,35], providing low-level control of the device. Due to this ability, firmware is considered a critical component of a device that has to be routinely updated and maintained in order to fix bugs, address performance-related issues, and even enhance or change the device's intended functionality. However, embedded device owners are often reluctant to update their devices' firmware due to the chance of rendering their devices inoperable in case of an error and because of the extensive downtime that they may experience [17,24]. On the other hand, manufacturers often do not provide firmware updates or support once their devices are released to the market. Even if they do, their updates typically do not conform to security principles including encryption and authentication. A recent survey proves that there have not been significant security gains in the firmware domain for the last 15 years [45]. The firmware security of embedded systems is not addressed to the same level as that of general-purpose computers or BIOS security [36].

[1] "Day Zero" or "Zero-day" is the day which a vulnerability of a system is made known to its vendor or to those who should be interested in mitigating the vulnerability. Hackers discovering these vulnerabilities can exploit them well before they are mitigated. Such exploits are known as "zero-day exploits".

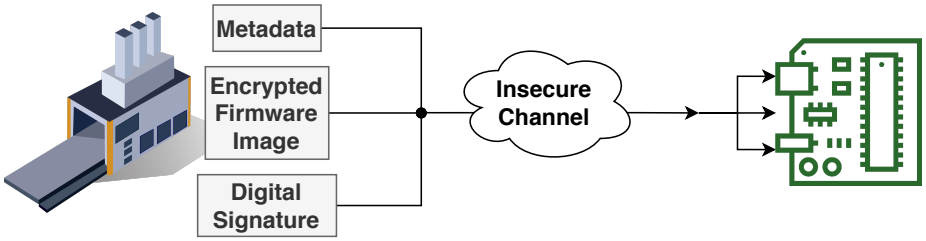

Fig. 1. An overview of the proposed approach. A firmware package is formed by combining necessary metadata, the encrypted firmware image, and the manufacturer's digital signature. The firmware package is transmitted to the embedded system through an insecure channel. The device unpacks it and verifies its source and contents utilizing hardware-implemented cryptographic primitives.

Existing industry efforts aim to secure firmware and other sensitive information stored at the device hardware in the form of secure storage and trusted execution environments. However, these methods have been proven to leak data to malicious attackers using a variety of attacks that take advantage of bugs and exploits in operating systems [39], user applications [33], and even proprietary code that aims at securing cryptographic keys and encrypted data [37]. To tackle these challenges, our work proposes the utilization of hardware-based cryptographic primitives to avoid storing secrets in non-volatile memory and alleviate reliance on software routines. Our proposed framework ensures firmware data integrity and confidentiality in a time-efficient manner by using hardware-implemented cryptographic modules while hardware-intrinsic characteristics are used as "digital fingerprints" to perform authentication procedures.

The overview of our framework is shown in Fig. 1. Firmware updates are transmitted from the manufacturer to the embedded device through an insecure channel. The proposed approach relies on hardware as a root-of-trust to attain high security levels and is motivated towards low-end embedded devices [19]. The framework is designed in a way that user intervention and device downtime are minimized. Hence, there is no need for intermediate authenticators to perform key exchange and management. The embedded device can be deployed without requiring any secure key-enrollment phase. Specifically, we employ hardware-implemented encryption and cryptographic hash functions to provide firmware confidentiality and data integrity, respectively. The means of digital fingerprinting are demonstrated via Physical Unclonable Functions (PUFs) [52]. The unclonable nature of PUFs bounds the firmware packages created by the manufacturer to a single device. If a device gets compromised, same model devices retain security. The PUF used in our proof-of-concept design is a Public PUF (PPUF), meaning that it does not rely on the secrecy of the Integrated Circuit's (IC) physical parameters since the model describing the PPUF is public.

Our framework is implemented and evaluated in an experimental setup using both software and hardware. A general-purpose computer is in charge of the firmware packaging procedure, acting as the device manufacturer. The embedded

device to be updated is emulated on FPGA as a proof-of-concept hardware design. The unpacking process, involving the authentication and decryption, is carried out on the embedded device. Our security analysis of the proposed approach shows strong security guarantees while our experimental measurements demonstrate the feasibility of the approach and applicability for constrained embedded devices.

The rest of the chapter is organized as follows. Related work on firmware update security mechanisms and PUF-based authentication protocols are discussed in Section 2. The underlying security primitives considered in our proof-of-concept design are discussed in Section 3. The proposed methodology for secure firmware updates is presented in Section 4 alongside with the security analysis of the approach. The experimental setup and implementation details are presented in Section 5. Section 6 concludes the chapter.

2 Related work

Firmware images and updates are typically provided online via vendors or manufacturers' websites. It has been shown that web crawlers can be used to gather images of critical equipment by traversing websites that host firmware [13]. These files can be accessed, downloaded, and modified due to the lack of access control measures and encryption. Firmware can also be acquired through physical access to the device [25]. By having access to the firmware image, a malicious adversary can retrieve the inner workings of a device and expose its functionality.

The information acquired from reverse engineering the firmware can lead to revealing zero-day exploits or other known vulnerabilities, that may provide an adversary an "attack path" to the system utilizing the aforementioned device. For example, access to the firmware image binary may allow adversaries to launch firmware modification attacks able to cause severe implications to a system's functionality. Recent works have demonstrated the severity of such attacks in the ICS domain when targeting devices such as PLCs and protection relays [4, 26, 54]. These types of attacks have also been shown to be effective against a large variety of other embedded devices such as printers, cameras, and network switches [13, 14].

Efforts to secure the firmware update mechanisms on embedded devices led to secure storage and trusted execution environments, such as i-NVMM [12] and ARM TrustZone [41]. However, the design of such systems is an attractive target for both invasive and non-invasive attacks [8,27,29]. For instance, since the JTAG protocol is not designed with security in mind, the JTAG test port can provide access to secure memories allowing embedded devices to be reconfigured. Also, it has been demonstrated that attackers are able to exploit implementation-based weaknesses to leak sensitive information through covert channels [40].

To overcome these shortcomings, we propose avoiding reliance on software and pre-stored data in non-volatile memories by moving towards a hardware root-of-trust. Research works towards this end, suggest using the hardware's intrinsic properties to design and support security mechanisms. Such solutions

include PUFs that leverage manufacturing variability to produce secret keys for authentication and encryption purposes. Different types of PUFs are used to produce unique identifiers that can be used in security schemes for authentication and secure code updates. Silicon-based PUFs (ring-oscillator, SRAM, arbiter, etc.) utilize manufacturing variability as an entropy source to create chip-specific identifiers. Examples of non-silicon designs include optical PUFs that exploit the random scattering of light to act as physical one-way functions. These kinds of solutions are successful in several domains such as intellectual property protection and Internet-of-Things (IoT) [9, 32].

Several approaches incorporating PUFs have been proposed to address the problem of storing sensitive information in non-volatile memories. Rostami *et al.* propose a PUF-based authentication and key exchange protocol based on substring matching [47]. The scheme utilizes PUFs for secure communications while alleviating the need for error correction against PUF's inherent noise. In [2], an end-to-end privacy-preserving authentication protocol is described, suitable for resource-constrained devices. The protocol attempts to perform mutual authentication procedures between enrolled embedded devices and a server utilizing reverse fuzzy-extraction mechanisms for key recreation on each side. In the context of IoT, a PUF-based communication protocol is presented in [10]. Before any secure communication is initiated between two devices, each PUF-enabled party has to share its Challenge-Response Pairs (CRPs) with an intermediate server. This server coordinates the communication between the two devices by issuing a public and a private key for each party. Feng *et al.* demonstrate a code update protocol utilizing PUFs [21]. The protocol starts with a temporary session in a secure environment between a server and the embedded device to share symmetric keys. The enrolled device is then employed and can securely communicate and update its code using its PUF for authentication. Che *et al.* show how within-die path delay variations can be utilized to enable a mutual PUF-based authentication protocol [11].

Most of the proposed solutions incorporate either strong or weak PUFs[2] which are highly susceptible to a variety of attacks. For example, weak PUFs have to remain entirely secret and an attacker with physical access could easily extract the required CRPs and break the authentication protocol in place [46]. Strong PUFs are more difficult to reverse engineer or extract information from, but they are highly susceptible to modeling and machine learning attacks. Such attacks involve producing a relatively low number of CRPs from the PUF and then create a machine learning model that will be trained using the gathered CRPs. The model then can quickly derive the remaining CRPs and create a

[2] The strength of a PUF depends on the number of CRPs that can be generated [34]. Weak PUFs produce very few CRPs derived from a physical characteristic. They usually act as fingerprints, e.g., a static bitstring, unique for each device due to manufacturing variability altering the IC's characteristics. Strong PUFs, on the other hand, produce a very large amount of CRP, exponential to their size, whereas weak PUFs produce a linear or polynomial number of CRPs. They are utilized as secret key providers for encryption/decryption purposes.

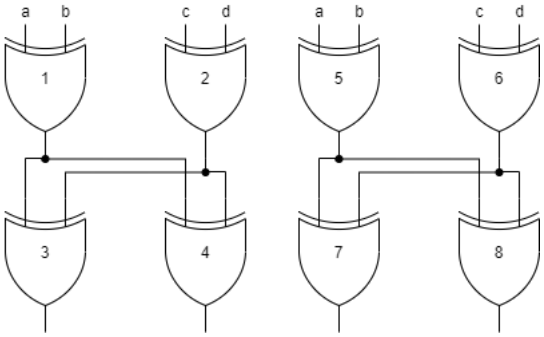

Fig. 2. An example of a small differential Public Physical Unclonable Function (dPPUF). Left (1-4) and right (5-8) circuits are the same in functionality. The challenge (abcd) is presented at both sides at the same time. However, the output of gates 3-4 and 7-8 stabilizes at different times due to manufacturing variability. The fastest propagating signals determine the corresponding response that forms the CRP.

dictionary that will contain an excessive amount of known pairs [48, 49, 55]. The aforementioned proposed protocols also require a secure setup phase; an essential key exchange procedure that must happen in a secure environment. A number of the proposed approaches also require intermediate servers to coordinate key distribution or enable communication between embedded devices. In comparison with the existing works on hardware-based secure communication protocols, our proposed framework neither requires a secure setup phase nor intermediate authenticator servers. The public-key infrastructure of our firmware update scheme alleviates the need for the aforementioned measures due to the public-key components including PPUFs, public-key cryptography, and digital signatures.

3 Underlying Security Primitives

The proposed framework employs cryptographic primitives in hardware in order to implement securely the firmware image exchange protocol. The approach requires both a private and a public key encryption/decryption core, a cryptographic hash function, and a PPUF in order to provide confidentiality, authenticity, and integrity guarantees.

PPUFs are a category of PUFs whose IC characteristics can be made public since they do not rely on their secrecy, unlike traditional PUFs. A PPUF is designed to be fast-to-execute and slow-to-simulate [42]. In the context of this work, a differential PPUF (dPPUF) is utilized due to its characteristics of not requiring ultra-accurate timing mechanisms. For instance, traditional XOR-based PPUFs, as the one shown in Fig. 2, require a very high clock resolution to accurately "catch" the racing signals at the end of the PPUF circuit. A 256-bit dPPUF is adopted from [43], presented in Fig. 3. PUFs are inherently noisy and therefore require error correction mechanisms to stabilize them. To alleviate for

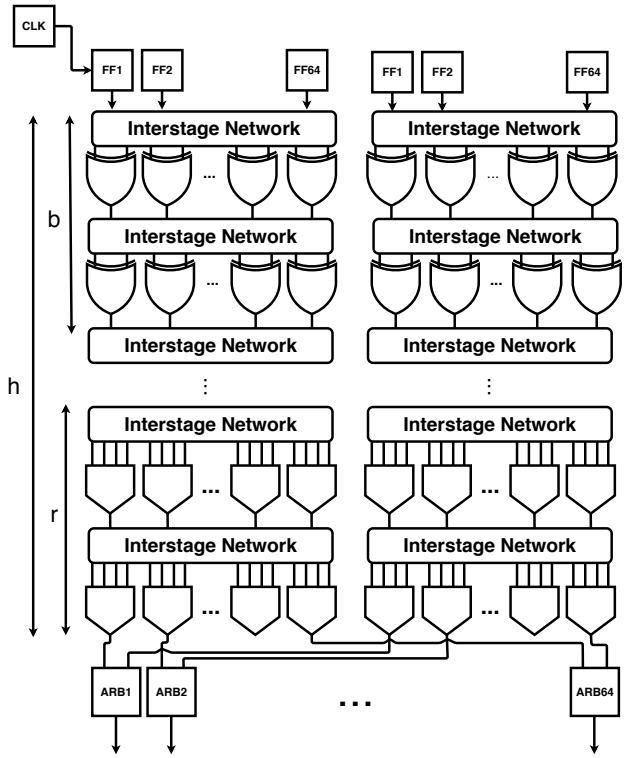

Fig. 3. The differential Public Physical Unclonable Function (dPPUF) architecture consists of consecutive layers of boosters and repressers. The two sides are identical in structure but different in inherent delays (inertial, propagation, switching, etc.). A layer of arbiters is placed at the end to capture the fastest propagating signals and according to the result, create the appropriate response bit string.

the PUF's inherent noise, in this work, and without loss of generality, we consider a Bose–Chaudhuri–Hocquenghem (BCH)-based code-offset fuzzy extractor as an effective PUF error correction mechanism [15].

In order to create a PPUF model, as required in our framework, the manufacturer has to perform gate-level characterization[3]. The measured IC characteristics form a software model that can be stored in a public repository. The software model is the "public part" of the PPUF since it does not provide any advantage to any adversary. This is because of the Execution-Simulation Gap (ESG). ESG is the time advantage the PPUF hardware owner has over a simulating attacker when calculating a CRP. A CRP is formed by the input to the PPUF's model,

[3] Gate-level characterization is the process of characterizing each gate of an IC in terms of its physical properties using lasers, micro-probing, and simulations [31]. Typical characteristics measured include gate width and length, and properties such as leakage power and switching power.

i.e., the challenge, and its corresponding output, i.e., the response. The procedure to produce a CRP is very fast when executed on hardware but significantly slower when done via simulation. In order to take advantage of this disparity we use the challenge as an encryption key while responding to the decrypting party. The inverse operation, i.e., deriving which challenge created the provided response, can only be completed on the actual PPUF hardware since simulating all the possible challenges to find a matching response is infeasible in simulation. Therefore, this ESG can be used as a root-of-trust and the model can be stored publicly without any security implications. ESG can also be manipulated to give as much advantage to the PPUF owner over the simulating adversary as needed. Increasing either the key width or number of challenges that need to be calculated increases the simulation effort for any attacker trying to derive the challenge of a particular response.

In order to provide confidentiality guarantees, we encrypt the firmware image. Encryption is the process of encoding plaintext data making it unintelligible and scrambled in a way that no unauthorized party can understand them. To decrypt the data, a cryptographic key is required. The key acts as a guide, helping the authorized party rearrange and reassemble the encrypted data correctly, so that access to the plaintext is possible. To address confidentiality, we employ the symmetric-key encryption algorithm Advanced Encryption Standard in Galois/Counter Mode (AES-GCM). AES-GCM is an authenticated encryption algorithm providing both data integrity and confidentiality [16]. Its hardware implementation provides a high encryption/decryption data rate while being adequately efficient in the use of hardware resources. AES-GCM is operating with 128-bit blocks and has four inputs: a 128-bit secret key, a 96-bit initialization vector, a plaintext, and optional additional authenticated data. AES-GCM generates two outputs: a message authentication code and a ciphertext. The message authentication code acts as a checksum value that enables integrity checks.

Towards ensuring that the firmware package contains undeniable truth that it originated from the manufacturer (non-repudiation) and thus protecting the device from impersonation attacks, the proposed approach utilizes digital signatures. The concept of digital signatures is depicted in Fig. 4. They are data that accompany the firmware image and provide evidence of their origin. To effectively use digital signatures, a cryptographic hash function is required along with the utilization of a public-key cryptosystem. Essentially, the sender has to hash the data payload, e.g., the firmware image, and encrypt it with a private key. In our setup, the procedure to create the manufacturer's digital signature is the following: (1) the manufacturer has to select a secret private key and (2) a public key. (3) A hash digest of the firmware image is created and encrypted by a public-key cryptosystem using the manufacturer's private key. The public-key cryptosystem utilized in our setup is RSA, also known as Rivest–Shamir–Adleman, while the cryptographic hash function employed is SHA-256, both NIST-approved cryptographic implementations [3, 44]. The verifier, in this case the device, is able to receive the data alongside the digital signature and recreate the digest on

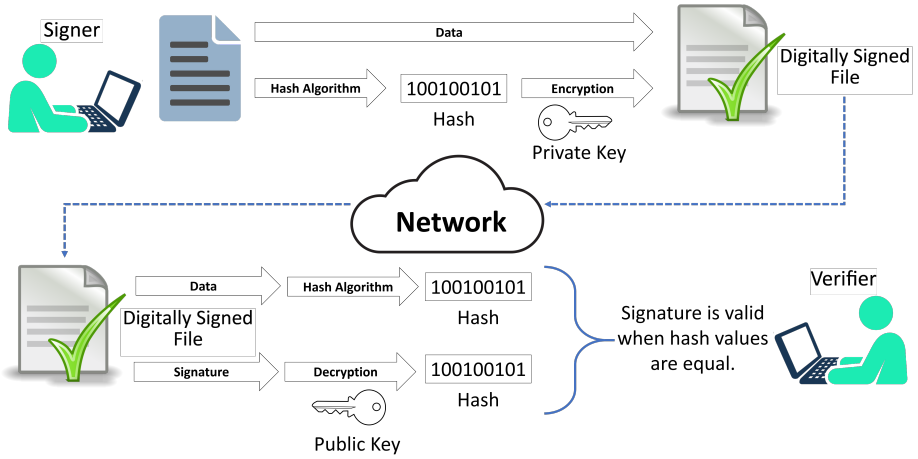

Fig. 4. Digital signature creation involves the usage of a cryptographic hash function and a public-key cryptosystem. A hash of the payload data is encrypted using the signer's private key. The verifier decrypts the digital signature using the signer's public key and hashes the payload data. The signature is considered valid if the resulting hashes match.

its side. Then the signature is decrypted using the manufacturer's public key and compared to that digest. If the hashes match, then the manufacturer is authenticated.

While the aforementioned cryptographic primitives are well established and widely used mechanisms in the area of cryptography and security, alternatives with similar characteristics can still be utilized in our proposed framework. Our approach is designed with modularity and flexibility in mind. Alternative cryptographic hash functions and encryption algorithms can be considered as long as they adhere to the needs of the secure firmware update protocol. For example, alternative encryption algorithms of symmetrical type can be chosen instead of AES-GCM. Examples include Simon [5], a lightweight block cipher released by the National Security Agency (NSA) and optimized for performance in hardware implementations, and Twofish [51], a symmetric key block cipher alternative to AES. The cryptographic hash function and public-key cryptosystem can also be interchanged with similar mechanisms. For instance, lighter alternatives can be used to adjust the design for even more constrained devices, such as a lightweight implementation of Keccak using only 200 permutations [7]. This flexibility gives the ability to the manufacturer to adjust their devices to certain security level constraints and available computational resources depending on each domain and application scenario.

4 Methodology

In this section, we provide the details of the proposed framework under the consideration of a malicious individual trying to manipulate the firmware updating procedure. The main objective is to encrypt a firmware image and deliver it to the embedded device through an insecure channel. An attacker observing the insecure channel should be unable to extract information from the firmware image. Only the intended device can decrypt the firmware image and verify its authenticity.

4.1 Threat Model and Countermeasures

We consider that the firmware packaging procedure from the manufacturer is an error-free process taking place in a secure facility, i.e., the firmware package is prepared correctly. The dPPUF model, however, is publicly available since accessing it does not give the attacker any advantage. The firmware package is transferred to the device over an insecure channel. The attacker can intercept packages on that channel. The attacker's goal is to uncover the firmware image binary and reverse-engineer it to place backdoors and uncover proprietary device operations. Using a malicious firmware binary the attacker tries to impersonate the device manufacturer to transfer a modified firmware package to the embedded system as the legitimate one.

In order to prevent this malicious activity, we employ several cryptographic techniques utilizing their hardware-implemented counterparts. To encrypt the firmware image and protect its binary form from unauthorized access, we use a symmetric cipher, specifically the AES-GCM. The keys required for this encryption procedure will be provided by the dPPUF and its model, making the firmware package chip-specific. In addition, the device receiving the firmware update must also have proof of the firmware package's origins in order to be secure against impersonation attacks. In order to alleviate this issue, we employ asymmetric cryptography as well in the form of digital signatures. A digital signature accompanying a payload gives proof of firmware origins while also being a checksum for integrity checks. The digital signatures in our setup utilize SHA-256 for cryptographic hashing and RSA for asymmetric cryptography.

4.2 Firmware Update Procedure

The firmware update procedure consists of two main parts, each undertaken – in sequence – by the device manufacturer and the device user. The manufacturer constructs a firmware package that contains the encrypted firmware image as well as metadata. Metadata allows the embedded device to authenticate and decrypt the firmware image without revealing useful information to any malicious entity observing the insecure channel used for data transfer. For this methodology to be successful, a combination of security primitives is utilized such as a cryptographic hash function (SHA-256), a symmetric (AES-GCM) and an asymmetric

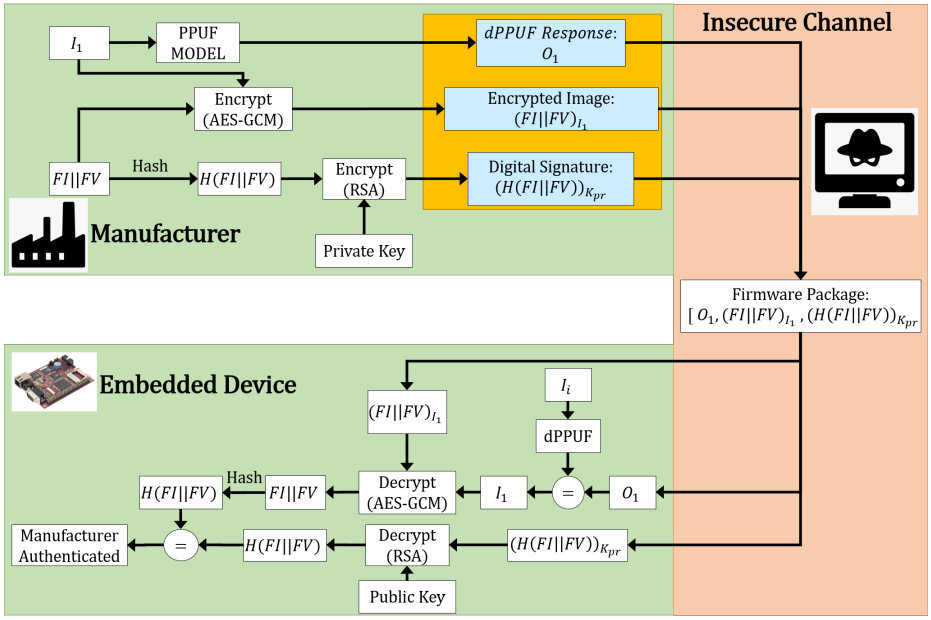

Fig. 5. The firmware package generation flow: the firmware vendor encrypts the composed image utilizing a public PUF model. The firmware unpacking process at the device level: the firmware package is decrypted, verified, and uploaded to the embedded device by utilizing public PUF's (dPPUF) intrinsic manufacturing variability.

(RSA) encryption/decryption module, and a PPUF (dPPUF). These crypto-modules are implemented on the hardware of the embedded device to avoid reliance on software routines and pre-stored data. The firmware transition, from the manufacturer packaging to the embedded device delivery, is shown in Fig. 5.

Secure Firmware Package Generation by Manufacturer: The upper half of Fig. 5 presents the steps required by the manufacturer to produce a valid and secure firmware package. The manufacturer uses a challenge to create a response from the dPPUF model, encrypts the firmware image using that challenge, and creates a digital signature. Then, these 3 output products are bundled together in a firmware package to be sent to the embedded device. In particular, the overall process involves the following:

1. The manufacturer generates a random challenge I_1. This is a 256-bit long binary that is going to be used as input to the dPPUF model. This challenge creates the 256-bit response O_1. This CRP's challenge I_1 is used to encrypt the firmware image such that only the intended device is able to decrypt it. The length of the CRP strings can be increased, if necessary, to further reduce the risk of brute force attacks.

2. The firmware image FI is concatenated with firmware metadata FV and then encrypted with AES-GCM using I_1 as the encryption key. The FI is the firmware binary and FV contains identifiers that will help the device further evaluate the firmware, namely firmware version and revision number. This information can help the device avoid rollback attacks as discussed in Section 4.3.
3. A hash of $FI||FV$ is calculated to create a digital signature. The resulting 256-bit digest of SHA-256 $H(FI||FV)$ is then encrypted using the manufacturer's private key K_{pr} to create $(H(FI||FV))_{K_{pr}}$. The public key encryption scheme used is RSA. The digital signature allows the device to authenticate the manufacturer since it is the only one capable of decrypting the firmware image and create a hash for comparison.
4. The three output products from the manufacturer are packaged and forwarded to the embedded device in this form: $[O_1, (FI||FV)_{I_1}, (H(FI||FV))_{K_{pr}}]$.

Firmware Unpacking Process by the Embedded Device: The generated firmware package is delivered to the dPPUF-enabled embedded system through an insecure network. The device utilizes the response O_1 to decrypt $(FI||FV)_{I_1}$ and authenticate the package's origins using the digital signature $(H(FI||FV))_{K_{pr}}$. The unpacking process steps include the following:

1. The embedded device makes use of the response O_1 to recreate challenge I_1. In order to achieve it, the device iterates throughout all possible input combinations to the dPPUF until it finds a response that matches O_1. This *CRP iteration* is only feasible using dedicated hardware and it is computationally prohibitive to carry out this operation through simulation [6]. Therefore, only the correct recipient device is able to perform this operation efficiently.
2. Once I_1 is derived, the embedded device uses I_1 as the key, for the hardware-implemented AES-GCM module, to decrypt $(FI||FV)_{I_1}$ and get $FI||FV$. Once the FV is obtained, the device is able to check the firmware version of the update and compared it to the firmware currently installed at the device. If the firmware image FI indicates an older version of the device's existing firmware, the update operation is aborted.
3. In parallel to the previous step, the digital signature $(H(FI||FV))_{K_{pr}}$ is decrypted with RSA using the manufacturer's public key. This operation results in $H(FI||FV)$.
4. If the firmware image indicates a firmware update, a cryptographic hash digest of $FI||FV$ is generated using SHA-256. The hash digest is compared to the result of the RSA decryption. If the hashes match, then the embedded device authenticates the origin of the firmware from the legitimate manufacturer.
5. If all the required authentication and decryption procedures are completed, the device can proceed with updating its firmware code.

As explained in Section 3, the dPPUF is a series of cascading gates which, when a challenge is introduced, a response is created. The manufacturer has

access to a software model describing the dPPUF present in the device which is going to be updated. Using a software model to simulate CRPs is significantly slower than performing the same operation directly on hardware. By using the challenge as an encryption key while providing the response in plaintext, we ensure that only the intended device is be able to decrypt the firmware image in a feasible time frame. This is based on the fact that in order to find a challenge for a corresponding response requires iterating through all possible challenges until a matching response is found. Therefore, we leverage the ESG characteristic of PPUFs to keep malicious attackers from getting a plaintext version of the firmware binary. The time advantage of the dPPUF hardware during CRP iteration over an attacker, simulating the same procedure, can vary depending on the dPPUF implementation. For example, a 1024-bit implementation of the dPPUF makes breaking the protocol extremely prohibitive, according to the analysis in [6]. Our implementation focuses on constrained devices, thus the structure is based on a 256-bit design. The firmware package is also chip-specific since every device will have a different PPUF and consequently different CRPs. The manufacturer also appends a digital signature to the encrypted firmware so that the embedded device can authenticate it.

Overall, our proposed framework has the following features: (1) it does not require a secure setup phase for key exchange between the firmware sender and receiver because the decryption key is dynamically generated by the hardware, (2) a malicious attacker observing the insecure channel cannot uncover the firmware image and cannot impersonate the manufacturer, (3) the firmware can only be decrypted by the intended device, and (4) the methodology can be easily adapted to incorporate different cryptographic primitives. Encryption/decryption modules and cryptographic hash functions can be substituted with other equivalent crypto-cores without altering the overall functionality of the design.

4.3 Security Analysis

The framework is designed to adhere to certain security requirements. First, we need to ensure that adversaries eavesdropping in the communication channel and able to intercept the firmware package, cannot reveal the binary of the firmware, and thus identify code subroutines that expose the embedded device's functionality. We also need to ensure that if the firmware package is corrupted or tampered during transmission, the device would be able to detect it. The device must also have the ability to authenticate the package's origins in order to be protected against impersonation and man-in-the-middle attacks. Installing earlier versions of firmware may re-introduce bugs and thus firmware rollback should be avoided.

To fulfill the above requirements we employ hardware-implemented versions of the AES-GCM algorithm, SHA-256 cryptographic hash functions, and RSA public-key cryptosystem. Since the firmware image, alongside the necessary metadata, is encrypted using AES-GCM, an attacker cannot extract the firmware binary from the transmitted package in plaintext form without having access to the key. The key, however, is only known by the manufacturer and only the

intended device can recreate it. RSA and SHA-256 are used to form the manufacturer's digital signature. The role of the digital signature is not only to act as a checksum value that allows the device to check the firmware's data integrity but also as undeniable evidence that the sender is indeed the manufacturer. Digital signatures require a hash digest which acts as a checksum value. The digest is encrypted by the manufacturer's private key. Therefore, it can only be decrypted using the manufacturer's public key, authenticating the package's origins. After completing the decryption and authentication procedures, the embedded device checks FV to determine the firmware version of the update. If it is not an update to the existing firmware, the device halts the updating operation to avoid rollback attacks.

The aforementioned security primitives are utilized as hardware-based cryptographic modules implemented on the embedded device. These primitives alleviate the need for secure storage of secret information such as credentials and encryption keys directly on the device's non-volatile memory. Keys and other secret information are dynamically recreated on the device upon demand. The firmware update operation is also non-dependent on software-based routines, and thus less susceptible to software-based attacks [53].

The utilized dPPUF circuit inherits by design certain security guarantees. The effort to simulate dPPUF using its public model scales exponentially with the dPPUF's depth and width. A small increase in depth or width may prove prohibitive in terms of time, for pre-computing all sets of CRPs. Even with enough computing capabilities for generating CRP lookup tables, the storage requirement would be impractically high. Also, the public model of dPPUF ensures that profile characterization (e.g., power profile) of the circuit would not reveal any side-channel information.

5 Experimental Setup and Results

We implement a proof-of-concept experimental setup in order to validate and evaluate our approach. As shown in Fig. 5, both the firmware packing and unpacking phases have been implemented. The procedures performed by the manufacturer are implemented in software using a 64-bit computer with 3.2GHz Intel Core i5-4460 quad-core processor and 8GB RAM. The unpacking process, on the embedded device's end, is emulated utilizing a Xilinx Kintex7 FPGA with a system clock frequency of $100MHz$.

Our PPUF implementation is using multiple layers of boosters (2-input XOR gate) followed by repressers (small NAND-based circuit [43]), i.e., $b = 1$ and $r = 1$ with the height and width of the dPPUF being $h = 10$ and $w = 256$, respectively. It is implemented on hardware, using the aforementioned FPGA, using artificial transmission and switching delays at each gate, shown in Fig. 3, to emulate manufacturing variations. The delay values are generated by a pseudo-random number generator software to avoid any kind of bias. The model of the dPPUF is constructed in $C++$ as a graph, where its nodes represent the dPPUF gates and their respective delays.

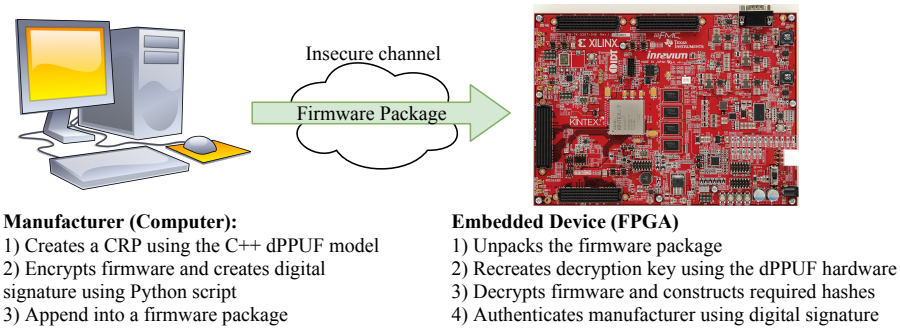

Fig. 6. Experimental and evaluation setup. The hardware-implemented security primitives are developed on a FPGA in order to emulate a dPPUF-enabled device. The firmware packaging procedure are carried out on a computer which is connected to the FPGA through a serial cable.

Fig. 7. Strict Avalanche Criterion (SAC) of dPPUF model. The red dashed line shows the ideal case, where $P(O_i = 1) = 0.5$ for all i.

The effectiveness of a dPPUF design is determined by the entropy that it exhibits. We ensure that the responses cannot be correlated to their corresponding challenges and its CRPs are adequately random by conducting extensive tests. Specifically, we validate the dPPUF's software model with $10k$ input vectors and compare them with the resulting outputs. Then, we utilize the Strict Avalanche Criterion (SAC) to quantify the entropy. SAC is measured by calculating the correlation probability of the corresponding outputs of two input vectors that have a hamming distance equal to 1. Fig. 7 presents the SAC demonstrated by our dPPUF design with an average probability of each output switching equal to 0.3425, similar to the results in the related literature [43].

Every procedure needed for completing the firmware packaging process is done in software. Firstly, the dPPUF model is utilized to create a CRP that will be used for encrypting the firmware package in a chip-specific way. The resulting response O_1 is going to be used as an unencrypted header for the firmware pack-

Table 1. FPGA resource utilization for the firmware unpacking process.

Resources	*AES-GCM*	*SHA-256*	*RSA*	*dPPUF*	*Overall Design*
LUT(#)	2671	1330	547	766	3183
FF(#)	1568	753	527	275	3981

Table 2. Firmware unpacking timings.

Device	**Firmware** (kB)	**Total execution** (ms)
Sercos III	233	72.27308
Zelio Logic	323	72.42759
Modicon	1183	73.90583

age while its corresponding challenge I_1 is also used as a secret key to encrypt the concatenation of the firmware image and the firmware version data $(FI||FV)$ to create $(FI||FV)_{I_1}$. The manufacturer also needs to prepare a digital signature to prove the firmware's origins to the device. The procedure for creating a digital signature involves hashing the concatenation $(FI||FV)$ to create $H(FI||FV)$ and then encrypting it using the manufacturer's private key. The resulting digital signature $(H(FI||FV))_{K_{pr}}$ is appended to O_1 and $(FI||FV)$ to create the firmware package. The required cryptographic algorithms, such as encryption/decryption and hashing, are implemented in Python using the *pycryptodome* library [18]. For our experiments, we choose three commercial firmware files acquired from the vendors' websites to be tested. The firmware images are those of embedded systems designed for ICS environments. The devices are a Sercos III field bus interface module, a Zelio Logic SRA2/SR3 smart relay, and a Modicon M258 logic controller.

We use an FPGA to emulate the embedded device supporting the described hardware-implemented cryptographic primitives. The unpacking process is carried out using Hardware Description Language (HDL) to demonstrate the effectiveness of the approach when running directly on hardware. We make use of Xilinx Vivado Design Suite 2018.3. The hardware resources required for these primitives are presented in Table 1. The hardware overhead for each security primitive is shown as the hardware usage of the overall design. Synthesis and implementation algorithms provided by the the HDL development tools help with optimizing the overall design in terms of area. Also, routing and placement algorithms can remove a lot of redundant hardware between these modules.

The firmware package produced by the manufacturer (e.g., the computer in our setup) is transferred at the receiving party which loads it into the memory and initiates the unpacking process. In our implementation, we first pre-load the firmware image in *block RAM* and then proceed with decryption and authen-

Table 3. Comparison with previous work.

Method	Area Overhead		Performance (ms)
	LUT (#)	FF (#)	
Proposed	3183	3981	73.90583
Aysu *et al.* [2]	3543	1275	61
Che *et al.* [11]	6038	1724	1250

tication procedures. During the unpacking process of the firmware, the dPPUF input challenge, O_1, from the firmware package header is used to recreate the key of AES-GCM, I_1. Then, the dPPUF iterates through all the possible challenges of a given set (10^6 possible inputs in our setup), until it finds a response matching O_1. This is the CRP iteration explained in Section 3, utilizing the ESG to have an advantage over the attacker. Once I_1 is derived, the AES-GCM decryption core is ready to decrypt $(FI||FV)_{I_1}$ and get the plaintext $FI||FV$. In parallel, the RSA core decrypts $(H(FI||FV))_{K_{pr}}$ using the manufacturer's public key K_{pub} to uncover $H(FI||FV)$. This hash derived from the digital signature is going to be compared to the output of the SHA-256 core when the input is the plaintext $FI||FV$. If the resulting hash matches the hash uncovered from the digital signature, then the manufacturer is authenticated and the embedded device is assured of the firmware image's origin and integrity. Finally, the FV is examined to determine the firmware image's version and revision number. If it is determined to be an actual update, then the firmware update procedure can be initiated. If the version or revision number indicate a downgrade, the firmware image is rejected to avoid rollback attacks. The whole experiment is run for each one of the industrial-grade firmware images. The time to finish the decryption and authentication is measured, starting from the package's arrival to the device. The three firmware images are presented in Table 2 along with their respective total unpacking time.

Table 3 provides a comparison with relevant state-of-the-art methods in terms of area and performance overhead. The time measured by [2] and [11] is the time that the device needs to establish a secure connection with the server and authenticate it. On the other hand, we measure our performance as the time a device takes to unpack a firmware package and perform all the necessary authentication and decryption procedures. Therefore, our measurements include time-consuming decryption procedures that put us at a disadvantage when compared to the time measurements reported by [2] and [11]. The performance for the proposed methodology, reported in Table 3, is the time needed to completely unpack the Modicon firmware image and authenticate the manufacturer. In [2] and [11], PUF-based privacy-preserving authentication protocols are being considered. When we only compare area overhead, [2] is lighter; however, it exhibits several drawbacks. As discussed in Section 2, the proposed approach by Aysu *et al.* requires initial setup and an enrollment phase (on top of PUF hardware characterization), steps which our approach does not require. It also necessitates

having a trusted third-party server to complete the authentication handshake. Also, the protocol does not take into account data integrity issues neither implements any countermeasures. In addition, the work utilizes an SRAM-based PUF which is known to be vulnerable to a variety of attacks [1, 46]. An authentication protocol that provides both confidentiality and mutual authentication is presented in [11]. This protocol is utilizing a new type of PUF, the hardware-embedded path delay (HELP) PUF. This kind of PUF derives randomness from path delay variance within a hardware implementation of AES. The work shows comparable resource usage. Nevertheless, the protocol does not address data integrity. Also, our implementation performs significantly better in terms of execution time than the mechanism proposed by [11], while also requiring a lot fewer hardware resources.

6 Conclusions

In this work, we develop a flexible firmware update framework for securely updating embedded systems. The framework makes use of the unique physical characteristics of each embedded system's IC to bind firmware packages to a specific device. By utilizing hardware-implemented cryptographic primitives, we can guarantee the confidentiality and integrity of the firmware image while being transmitted through an insecure channel. Our framework's security analysis demonstrates the validity of the security measures while showing the device's protection mechanisms against impersonation and other types of attacks. A proof-of-concept implementation with a commercial-off-the-shelf firmware of an industrial embedded system verifies the practicality of the approach in resource constraint devices. The performance results show that the proposed framework not only provides security but also fast firmware updates.

Acknowledgment

This work has been supported by the European Union's Horizon 2020 research and innovation programme under grant agreement No 739551 (KIOS CoE) and from the Government of the Republic of Cyprus through the Directorate General for European Programmes, Coordination and Development. Partial support of this research was also provided by the Woodrow W. Everett, Jr. SCEEE Development Fund in cooperation with the Southeastern Association of Electrical Engineering Department Heads.

References

1. Anagnostopoulos, N.A., Arul, T., Rosenstihl, M., Schaller, A., Gabmeyer, S., Katzenbeisser, S.: Low-temperature data remanence attacks against intrinsic sram pufs. In: 2018 21st Euromicro Conference on Digital System Design (DSD). pp. 581–585. IEEE (2018)

2. Aysu, A., Gulcan, E., Moriyama, D., Schaumont, P., Yung, M.: End-to-end design of a puf-based privacy preserving authentication protocol. In: International Workshop on Cryptographic Hardware and Embedded Systems. pp. 556–576. Springer (2015)
3. Barker, E., Dang, Q.: Nist special publication 800-57 part 1, revision 4. NIST, Tech. Rep (2016)
4. Basnight, Z., Butts, J., Lopez Jr, J., Dube, T.: Firmware modification attacks on programmable logic controllers. International Journal of Critical Infrastructure Protection 6(2), 76–84 (2013)
5. Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., Wingers, L.: The simon and speck families of lightweight block ciphers. IACR Cryptology ePrint Archive 2013(1), 404–449 (2013)
6. Beckmann, N., Potkonjak, M.: Hardware-based public-key cryptography with public physically unclonable functions. In: International Workshop on Information Hiding. pp. 206–220. Springer (2009)
7. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Keccak. In: Annual international conference on the theory and applications of cryptographic techniques. pp. 313–314. Springer (2013)
8. Breeuwsma, M., De Jongh, M., Klaver, C., Van Der Knijff, R., Roeloffs, M.: Forensic data recovery from flash memory. Small Scale Digital Device Forensics Journal 1(1), 1–17 (2007)
9. Brisbanne, O.M., Bossuet, L.: Restoration protocol: Lightweight and secure devices authentication based on puf. In: IFIP/IEEE International Conference on Very Large Scale Integration, VLSI-SoC 2017 (2017)
10. Chatterjee, U., Chakraborty, R.S., Mukhopadhyay, D.: A puf-based secure communication protocol for iot. ACM Transactions on Embedded Computing Systems (TECS) 16(3), 67 (2017)
11. Che, W., Martin, M., Pocklassery, G., Kajuluri, V.K., Saqib, F., Plusquellic, J.: A privacy-preserving, mutual puf-based authentication protocol. Cryptography 1(1), 3 (2017)
12. Chhabra, S., Solihin, Y.: i-nvmm: a secure non-volatile main memory system with incremental encryption. In: 2011 38th Annual international symposium on computer architecture (ISCA). pp. 177–188. IEEE (2011)
13. Costin, A., Zaddach, J., Francillon, A., Balzarotti, D.: A large-scale analysis of the security of embedded firmwares. In: 23rd {USENIX} Security Symposium ({USENIX} Security 14). pp. 95–110 (2014)
14. Cui, A., Costello, M., Stolfo, S.J.: When firmware modifications attack: A case study of embedded exploitation. In: NDSS (2013)
15. Dodis, Y., Ostrovsky, R., Reyzin, L., Smith, A.: Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. SIAM journal on computing 38(1), 97–139 (2008)
16. Dworkin, M.J.: Sp 800-38d. recommendation for block cipher modes of operation: Galois/counter mode (gcm) and gmac. NIST (2007)
17. Eaton, C.: Hacked: Energy industry's controls provide an alluring target for cyber-attacks. [Online]: `http://www.houstonchronicle.com/` (2017)
18. Eijs, H.: pycryptodome. `https://github.com/Legrandin/pycryptodome` (2014)
19. Falas, S., Konstantinou, C., Michael, M.K.: A hardware-based framework for secure firmware updates on embedded systems. In: 2019 IFIP/IEEE 27th International Conference on Very Large Scale Integration (VLSI-SoC). pp. 198–203. IEEE (2019)
20. Falliere, N., Murchu, L.O., Chien, E.: W32. stuxnet dossier. White paper, Symantec Corp., Security Response 5(6), 29 (2011)

21. Feng, W., Qin, Y., Zhao, S., Liu, Z., Chu, X., Feng, D.: Secure code updates for smart embedded devices based on pufs. In: International Conference on Cryptology and Network Security. pp. 325–346. Springer (2017)
22. ISACA: Firmware security risks and mitigation: Enterprise practices and challenges. [Online]: https://cybersecurity.isaca.org/csx-resources/firmware-security-risks-and-mitigation---enterprise-practices-and-challenges (October 2016)
23. Kolias, C., Kambourakis, G., Stavrou, A., Voas, J.: Ddos in the iot: Mirai and other botnets. Computer 50(7), 80–84 (2017)
24. Konstantinou, C., Chielle, E., Maniatakos, M.: Phylax: Snapshot-based profiling of real-time embedded devices via jtag interface. In: 2018 Design, Automation Test in Europe Conference Exhibition (DATE). pp. 869–872 (March 2018)
25. Konstantinou, C., Keliris, A., Maniatakos, M.: Taxonomy of firmware trojans in smart grid devices. In: Power and Energy Society General Meeting (PESGM), 2016. pp. 1–5. IEEE (2016)
26. Konstantinou, C., Maniatakos, M.: Impact of firmware modification attacks on power systems field devices. In: Smart Grid Communications, 2015 IEEE International Conference on. pp. 283–288. IEEE (2015)
27. Konstantinou, C., Maniatakos, M.: A case study on implementing false data injection attacks against nonlinear state estimation. In: Proceedings of the 2nd ACM Workshop on Cyber-Physical Systems Security and Privacy. pp. 81–92 (2016)
28. Konstantinou, C., Maniatakos, M.: Security analysis of smart grid. Commun Control Secur Challenges Smart Grid 2, 451 (2017)
29. Konstantinou, C., Maniatakos, M.: Hardware-layer intelligence collection for smart grid embedded systems. Journal of Hardware and Systems Security pp. 1–15 (2019)
30. Konstantinou, C., Maniatakos, M., Saqib, F., Hu, S., Plusquellic, J., Jin, Y.: Cyber-physical systems: A security perspective. In: 2015 20th IEEE European Test Symposium (ETS). pp. 1–8. IEEE (2015)
31. Koushanfar, F., Boufounos, P., Shamsi, D.: Post-silicon timing characterization by compressed sensing. In: Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design. pp. 185–189. IEEE Press (2008)
32. Li, W., Wang, Y., Li, H., Li, X.: p3m: A pim-based neural network model protection scheme for deep learning accelerator. In: Proceedings of the 24th Asia and South Pacific Design Automation Conference. pp. 633–638 (2019)
33. Lipp, M., Schwarz, M., Gruss, D., Prescher, T., Haas, W., Mangard, S., Kocher, P., Genkin, D., Yarom, Y., Hamburg, M.: Meltdown. arXiv preprint arXiv:1801.01207 (2018)
34. McGrath, T., Bagci, I.E., Wang, Z.M., Roedig, U., Young, R.J.: A puf taxonomy. Applied Physics Reviews 6(1), 011303 (2019)
35. McLaughlin, S., Konstantinou, C., Wang, X., Davi, L., Sadeghi, A.R., Maniatakos, M., Karri, R.: The cybersecurity landscape in industrial control systems. Proceedings of the IEEE 104(5), 1039–1057 (2016)
36. Mocker, A.: Tuya: Revised update process hacked again. https://www.heise.de/ (Nov 2019)
37. Moghimi, D., Sunar, B., Eisenbarth, T., Heninger, N.: TPM-FAIL: TPM meets Timing and Lattice Attacks. In: 29th USENIX Security Symposium (USENIX Security 20). USENIX Association, Boston, MA (Aug 2020), https://www.usenix.org/conference/usenixsecurity20/presentation/moghimi
38. National Crime Agency UK: The cyber threat to uk business. National Cyber Security Centre (2017-18)

39. Newman, L.H.: Windows 10 has a security flaw so severe the nsa disclosed it (Jan 2020), https://www.wired.com/story/nsa-windows-10-vulnerability-disclosure/
40. O'Flynn, C., Chen, Z.D.: Side channel power analysis of an aes-256 bootloader. In: Electrical and Computer Engineering (CCECE), 2015 IEEE 28th Canadian Conference on. pp. 750–755. IEEE (2015)
41. Pinto, S., Santos, N.: Demystifying arm trustzone: A comprehensive survey. ACM Computing Surveys (CSUR) 51(6), 1–36 (2019)
42. Potkonjak, M., Goudar, V.: Public physical unclonable functions. Proceedings of the IEEE 102(8), 1142–1156 (2014)
43. Potkonjak, M., Meguerdichian, S., Nahapetian, A., Wei, S.: Differential public physically unclonable functions: architecture and applications. In: Proceedings of the 48th Design Automation Conference. pp. 242–247 (2011)
44. PUB, F.: Secure hash standard (shs). Fips pub 180(4) (2012)
45. Roberts, P.: Huge survey of firmware finds no security gains in 15 years. https://securityledger.com/ (Aug 2019)
46. 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 (2016)
47. Rostami, M., Majzoobi, M., Koushanfar, F., Wallach, D.S., Devadas, S.: Robust and reverse-engineering resilient puf authentication and key-exchange by substring matching. IEEE Transactions on Emerging Topics in Computing 2(1), 37–49 (2014)
48. Rührmair, U., Sehnke, F., Sölter, J., Dror, G., Devadas, S., Schmidhuber, J.: Modeling attacks on physical unclonable functions. In: Proceedings of the 17th ACM conference on Computer and communications security. pp. 237–249. ACM (2010)
49. Sahoo, D.P., Nguyen, P.H., Mukhopadhyay, D., Chakraborty, R.S.: A case of lightweight puf constructions: Cryptanalysis and machine learning attacks. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 34(8), 1334–1343 (2015)
50. Sanjab, A., Saad, W., Guvenc, I., Sarwat, A., Biswas, S.: Smart grid security: Threats, challenges, and solutions. arXiv preprint arXiv:1606.06992 (2016)
51. Schneier, B., Kelsey, J., Whiting, D., Wagner, D., Hall, C., Ferguson, N.: Twofish: A 128-bit block cipher. NIST AES Proposal 15(1), 23–91 (1998)
52. Suh, G.E., Devadas, S.: Physical unclonable functions for device authentication and secret key generation. In: Design Automation Conference, 2007. DAC'07. 44th ACM/IEEE. pp. 9–14. IEEE (2007)
53. Van Bulck, J., Minkin, M., Weisse, O., Genkin, D., Kasikci, B., Piessens, F., Silberstein, M., Wenisch, T.F., Yarom, Y., Strackx, R.: Foreshadow: Extracting the keys to the intel {SGX} kingdom with transient out-of-order execution. In: 27th {USENIX} Security Symposium ({USENIX} Security 18). pp. 991–1008 (2018)
54. Wang, X., Konstantinou, C., Maniatakos, M., Karri, R., Lee, S., Robison, P., Stergiou, P., Kim, S.: Malicious firmware detection with hardware performance counters. IEEE Transactions on Multi-Scale Computing Systems 2(3), 160–173 (2016)
55. Xu, X., Burleson, W.: Hybrid side-channel/machine-learning attacks on pufs: A new threat? In: 2014 Design, Automation & Test in Europe Conference & Exhibition (DATE). pp. 1–6. IEEE (2014)