A Hardware-based Framework for Secure Firmware Updates on Embedded Systems

Solon Falas[*], **Charalambos Konstantinou**[†] **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
E-mail: sfalas01@ucy.ac.cy, ckonstantinou@fsu.edu, mmichael@ucy.ac.cy

***Abstract*—The ability to update firmware in embedded systems allows end-users to patch device vulnerabilities and improve functionality. However, this process is often exploited by adversaries in order to inject malicious firmware code into embedded devices. In this paper, we present a framework which enables highly secure and fast firmware update delivery with minimal downtime on embedded devices. The proposed framework utilizes device intrinsic physical characteristics to authenticate firmware packages along with integrated cryptographic modules to ensure the firmware confidentiality and integrity. A proof-of-concept design is implemented on FPGA, which demonstrates high performance with reasonable overheads, while our analysis shows strong security guarantees.**

***Index Terms*—Embedded systems, firmware updates, hardware security.**

I. Introduction

Embedded devices are increasingly integrated in several domains including Industrial Control Systems (ICS), home and automation networks, wireless sensing services, etc. The deployment of embedded systems in mission-critical environments introduces unique security challenges: such devices are highly constrained in terms of performance and resources. Hence, it is often infeasible to employ the same security measures used in general-purpose computing systems. This is evident by the growing number of attack incidents related to microprocessor-based embedded devices (e.g., Stuxnet in 2010, Ukraine power grid cyberattack in 2015, etc.) [1].

Firmware in embedded systems is the dedicated software, typically residing in read-only memory, that “sits on top” of the device’s hardware. Traditional update mechanisms allow the bootloader to follow a similar to the *boot sequence* process in order to flash and validate a new firmware image. The process, however, to securely update firmware in embedded devices while minimizing system downtime remains a challenge for many manufactures [2]. Insecure update mechanisms, i.e., that do not provide authentication and integrity of the patch, maybe exploited maliciously in order to bypass security mechanisms within the firmware code and eventually allow adversaries to gain full control of the device. To tackle these challenges, our framework proposes the utilization of hardware-based cryptographic primitives to ensure integrity and confidentiality while hardware-intrinsic characteristics are used as “digital fingerprints” to perform authentication procedures. The firmware image is encrypted by the firmware vendor and combined with metadata to form the firmware package. This package can be transferred through an insecure channel to the embedded device. Upon arrival, the embedded device utilizes the metadata to authenticate the firmware package origins by means of digital fingerprinting techniques such as Physical Unclonable Functions (PUFs) [3], followed by decryption of the package so that the firmware update can be performed. In this manner, a hardware-based root-of-trust towards assuring firmware integrity and preventing rogue firmware updates is established.

The proposed approach is motivated towards low-end embedded devices, possibly deployed in mission-critical environments which allow for limited performance overheads. User intervention and device downtime are minimized, as a setup phase or intermediate authenticators for key exchange and management are not required. We leverage the effects of manufacturing variability and the concept of PUFs so that firmware update packages are chip-specific. Thus, if a device gets compromised, same-model devices will retain security. Furthermore, we utilize public PUFs (PPUFs) as hardware primitives to build a public-key based protocol. The security of the scheme does not rely on the secrecy of the Integrated Circuit (IC) physical parameters.

The proposed framework has been implemented and evaluated using both software and hardware. The firmware package procedure was implemented in software to produce the secure firmware package when a firmware image is provided. Hardware proof-of-concept on FPGA was used for the unpacking process to be performed by the embedded device where a differential Public PUF (dPPUF) [4] was implemented, along with SHA-256 hashing algorithm and an AES-GCM decryption core. The dPPUF acts as the digital fingerpint of the device in order to perform the authentication procedure while AES-GCM and SHA-256 are used for decryption and integrity checks. Our security analysis shows strong security guarantees while our experimental measurements demonstrate the feasibility of the approach for embedded devices with limited computational resources and minimal downtime requirements.

The rest of the paper is organized as follows. Related work on firmware update security and firmware modification attacks are discussed in Section II. The underlying security primitives used in our implementation are discussed in Section III. The proposed methodology for secure firmware updates is presented in Section IV. Our experimental setup and results are presented in Section V and Section VI concludes the paper.

II. Related work

Firmware files are often provided online via the vendors or manufacturers' websites. Previous work has demonstrated that web crawlers could gather images of critical equipment [5]. Firmware can also be acquired through physical access to the device [6]. Access to the firmware allows adversaries to modify the code causing severe implications to the system functionality. For instance, the severity of such attacks has been shown in the ICS domain for devices such as programmable logic controllers and protection relays [7], [8]. Attacks have also been demonstrated in a variety of other embedded devices such as printers, cameras, and network switches [5], [9].

Efforts to secure the firmware loading and update mechanisms on embedded devices led to crypto-bootloaders. However, inherent vulnerabilities in the design of such systems makes them attractive targets to both invasive and non-invasive attacks [10]. For instance, if JTAG access is left enabled embedded devices can be reprogrammed. Also, adversaries may exploit implementation-based weaknesses able to leak secret information through covert channels [11]. Intrinsic properties at the hardware level have been successfully used for several security procedures. As an example, PUF's inherent entropy (e.g. silicon-based PUFs: ring-oscillator, SRAM, arbiter, etc.) to produce unique identifiers has been used in schemes to authenticate and secure code updates in several domains such as IP protection and Internet-of-Things (IoT) [12], [13].

Rostami *et al.* have demonstrated a PUF-based authentication and key exchange protocol [14]. The scheme relies on substring matching and it is designed to be robust against inherent noise without the use of correction modules. End-To-End design of a privacy-preserving authentication protocol suitable for resource-constrained devices is described in [15]. The protocol is based on a mutual authentication procedure between devices and a server, and utilizes reverse fuzzy-extraction for key recreation. Feng *et al.* have developed a code update protocol based on PUFs which includes a temporary session between a server and a device in a secure environment using a symmetric key [16]. PUF challenge-response pairs (CRPs), helper data, and reverse fuzzy extractors are used as authentication modules. In the context of IoT, a PUF-based communication protocol has been presented in [17]. Before any secure communication (between two devices) is initiated, each device has to share its CRPs with an intermediate server. The server then calculates private and public keys based on those CRPs and returns them to the appropriate device. Che *et al.* have utilized the randomness derived from within-die path delay variations in order to enable a mutual PUF-based authentication protocol [18]. In comparison with existing work on hardware-based firmware update schemes, our work does not require a setup phase for key exchange neither intermediate authenticators. Our public-key based firmware update framework minimizes device downtime and achieves strong security guarantees while providing flexibility in the selection of the underlying security primitives.

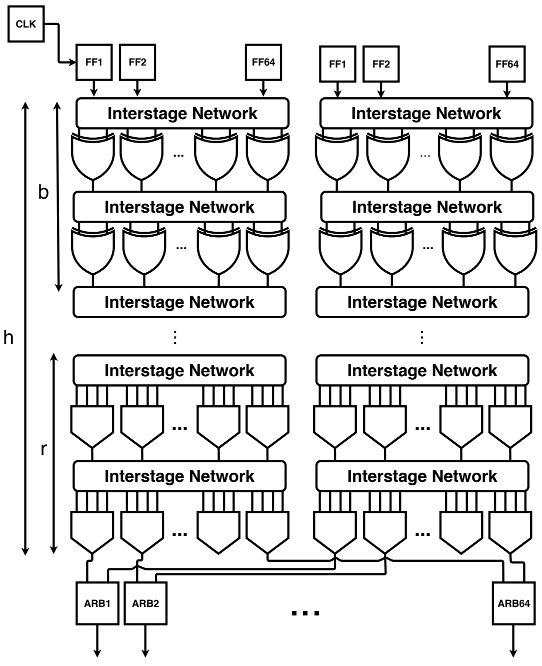

Fig. 1. The dPPUF architecture consists of consecutive layers of boosters and repressers. The two sides are identical in structure but different in inherent delays (inertial, propagation, switching, etc.). A layer of arbiters is placed at the end to capture the fastest propagating signals and according to the result, create the appropriate response bit string.

III. Underlying Security Primitives

PPUFs are public-key based primitives designed to be *fast-to-execute* and *slow-to-simulate systems* [4]. The model associated with the PPUF hardware is publicly known. The difference between the time needed to calculate the response from the hardware and the time needed to simulate the same response (Execution Simulation Gap - ESG) is used as a source of secure authentication. Thus, in addition to the unclonable nature, the speedup of the PPUF hardware over any simulated digital model serves as the authentication pillar. Using a PPUF enables us to avoid the need for intermediate authenticators, key-distributing servers or a secure enrollment phase (beyond the hardware characterization phase of the PPUF), in which pre-shared keys are placed at each end of the communicating parties to enable secure interactions. This allows the firmware manufacturer to package and send a firmware update to a device without having to pre-install sensitive data in a device's memory or remove the need for time-consuming handshaking procedures. The PPUF model used in our work is the *differential PPUF (dPPUF)*. dPPUFs eliminate the requirement of precise timing and ultra-sensitive triggering mechanisms using a layer of arbiters connected to the dPPUF output. Fig. 1 illustrates the dPPUF architecture used in our methodology, which is a 256-bit design adopted from [19]. PUF responses are inherently noisy, therefore error correction mechanisms are required to improve stability. Multiple approaches have been proposed, such as Error Correction Codes (ECC) and special PUF designs [20]. Without any loss of generality, we use a *BCH-based code-offset fuzzy extractor* due to its fast, lightweight, and secure nature to convert PUF responses into high-quality keys [21].

The symmetric-key algorithm used in our framework is *Advanced Encryption Standard in Galois/Counter Mode (AES-GCM).* It is an authenticated encryption algorithm designed to provide securely and efficiently both data integrity and confidentiality. Its hardware implementation requires reasonable resources and can achieve high speeds with low cost and low latency, although this is one of many approaches. AES-GCM has four inputs: a secret key, an Initialization Vector (IV), a plaintext, and optional Additional Authenticated Data (AAD). AES-GCM generates two outputs: a message authentication code and a ciphertext.

The proposed framework also utilizes *SHA-256 cryptographic hash function* to verify data integrity at several steps of the firmware packing and unpacking process. For example, the code and version of the firmware are hashed to generate the IV as a distinct identifier for each AES-GCM operation. In order to add an extra layer of security in the GHASH function of AES-GCM, a hash value of the AES-GCM encryption key is also generated and included in the AAD input in a Hash-based Message Authentication Code (HMAC) fashion. Details on how the underline security primitives interact within the proposed framework are provided in Section IV.

Our approach is modular and flexible; the proposed framework can consider alternative encryption algorithms of symmetrical type, hash functions, or public model of hardware-based primitives given specific security level constraints and available computational resources. Fig. 2 presents the overview of how dPPUF, for example, is utilized as a part of the firmware authentication process between the manufacturer and the embedded system. The Gate Level Characterization (GLC)[1] of the dPPUF performed by the manufacturer results in a software model stored in a public register. With the software model acting as the "public part" of our security scheme, we can avoid time-consuming key-sharing between the communicating parties while rendering the recreation of keys unfeasible for attackers due to the rigorous simulation required. The responses of the dPPUF derived from the model are used to form the symmetric secret key (e.g., via bitwise XOR), acting as the "private part", that will be utilized for the *encryption* of the firmware and *hashing* of the required metadata. The dPPUF hardware at the device level follows a similar procedure. It reconstructs the key and performs authenticity checks. If the checks are valid and fall within the desired time constraint T_0, then firmware is decrypted and uploaded to the system.

IV. Methodology

In this section, we provide the details of the proposed scheme's two main procedures, each undertaken – in sequence – by the device manufacturer and the device customer. The key concept is to construct a firmware package that will contain the encrypted firmware image as well as metadata provided by the manufacturer. This information will allow the embedded device to authenticate and decrypt the firmware image without revealing any useful information to any malicious entity observing the insecure channel used for data transfer.

[1]Gate-level characterization (GLC) is the process of characterizing each gate of an IC in terms of its physical properties, such as gate width and length, or its manifestation properties, such as leakage power and switching power, usually using micro-probing measurements and simulations [22].

Authentication Procedure

Manufacturer	Device
1. Generate challenge $c = [x_0, x_1]$ 2. Compute response with Public Model $p = [y_0, y_1]$ 3. Bit-wise XOR $(y_0 \oplus y_1) = k$ 4. Send challenge and hashed key (h_k)	5. Use dPPUF to compute response $p' = [y'_0, y'_1]$ 6. Bit-wise XOR $(y'_0 \oplus y'_1) = k'$ 7. Compare h_k and h'_k 8. If $h_k = h'_k$ and $(\tau_5 + \tau_6 + \tau_7) \leqslant T_0$, accept update

Fig. 2. Authenticating a public model PUF (e.g., dPPUF) system between two parties: device manufacturer and embedded device.

A. Secure Firmware Package Generation by Manufacturer

The left-hand side of Fig. 3 presents the steps required by the manufacturer to produce a valid and secure firmware package. In particular, the overall process involves the following:

1) The first input to the SHA-256 is the concatenation of the firmware image f_i and the firmware version f_v: $f_i || f_v$. The initialization vector IV for AES-GCM encryption is the first 96-bit vector output of this hash operation. The generated IV is also used as part of the firmware package header r.
2) The output response signal $p = [y_0, y_1]$ of the dPPUF model with input challenge $c = [x_0, x_1]$, where $n =$ 256-bits, is used as the encryption key of AES-GCM using bitwise XOR, $k = y_0 \oplus y_1$.
3) The signal output k from the simulated dPPUF is hashed using SHA-256, to provide h_k.
4) The firmware package header r includes the IV, the hash response output of dPPUF h_k, and the dPPUF input challenge $c = [x_0, x_1]$, i.e., $r =< IV, h_k, c >$.
5) The AES-GCM encryption core takes four inputs: IV and k as the encryption parameters, the firmware code and version as the data to be encrypted and authenticated $m =< f_i || f_v >$, and the package header r as the AAD to be authenticated.
6) The final form of the firmware package f_p includes the package header r, the AES-GCM encrypted message m_{enc}, and the AES-GCM generated authentication tag GHASH t_{128}: $f_p =< r, m_{enc}, t_{128} >$.

B. Firmware Unpacking by Embedded System

The generated firmware package f_p is delivered to the dPPUF-enabled embedded device. Even if transferred through an insecure network, it ensures that the authentic firmware image is loaded to the embedded device without being corrupted. The package is decomposed as shown in the right-hand side of Fig. 3 in order to decrypt, authenticate, and validate the integrity of the firmware data. The steps of this procedure are:

1) From the package header $r =< IV, h_k, c >$ of the firmware package f_p, the input challenge $c = [x_0, x_1]$,

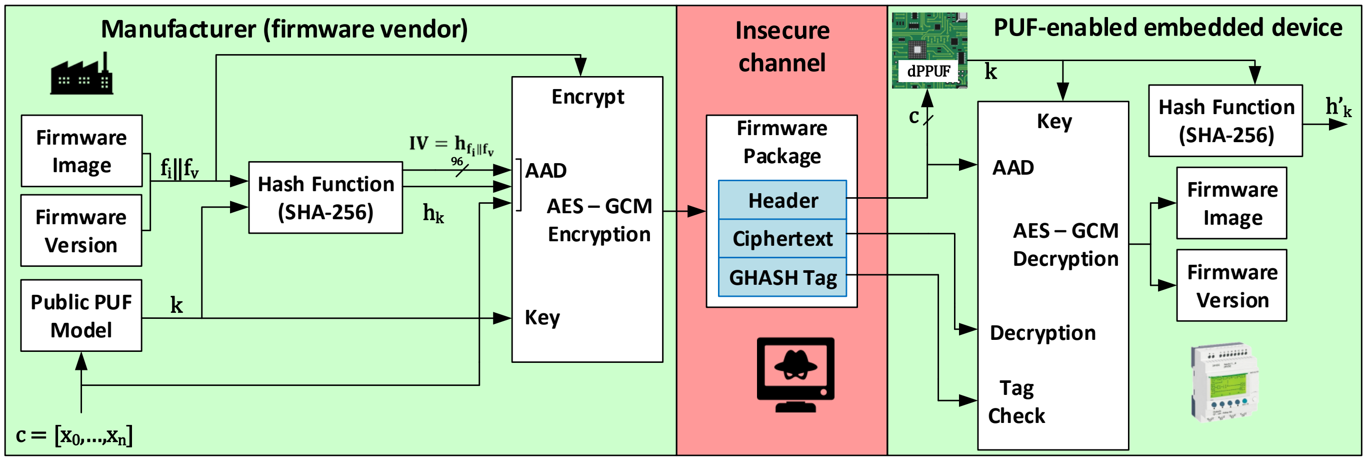

Fig. 3. The firmware package generation flow: the firmware vendor encrypts the composed image utilizing a public PUF model. The firmware unpacking process at the device level: the firmware package is decrypted, verified, and uploaded to the embedded device by utilizing public PUF's (dPPUF) intrinsic manufacturing variability.

$n = 256$, is fed to the hardware dPPUF in order to recreate the response $p = [y_0, y_1]$ and hence, the symmetric secret key $k = y_0 \oplus y_1$ to be used in AES-GCM. Note that p is the error-corrected version of the dPPUF output with the help of the fuzzy extractor.

2) The hash of the key, h_k, included in the firmware header r is compared with the SHA-256 hash output (h'_k) of the dPPUF response vector k. If the hashes do not match or the time $\tau = \tau_p + \tau_k$ to generate p and k exceeds T_0 $(\tau > T_0)$, an alarm will be triggered and appropriate actions, specified by the integrator, can be initiated. In our implementation, the firmware update procedure is halted and the package is discarded. If the number of failed attempts exceeds a certain threshold value, the device enters an emergency lockdown mode to prevent replay attacks.
3) The AES-GCM core decrypts $m_{enc} = < f_i || f_v >_{enc}$ using the IV from header r and the decryption key k generated by the dPPUF. The header r is also used as the AAD input to the decryption, similarly to the encryption process.
4) The output message authentication tag $\hat{t}_{128}$ of AES-GCM is compared with the authentication tag t_{128} included in the f_p. If the integrity tags do not match, the update procedure is halted and the package is discarded due to data corruption.
5) In the scenario that the authentication tags match, the decrypted data $m =< f_i || f_v >$ are considered valid.
6) If all the above steps are successful, the updated f_{v_i} replaces the installed version f_{v_j} if $i > j$. In the scenario that $i \leq j$, similarly to step 3, an alarm is raised to avoid rollback attacks. Otherwise, the firmware image f_i is loaded to the device.

The security of this scheme relies on the properties of its individual parts including AES-GCM and SHA-256. An HMAC-based scheme is also introduced to further extend the security of the authentication with the hashed key acting as the inner key and GHASH tag as the outer key. The unclonable nature and speed of the dPPUF enable a secure hardware-based update scheme. Our methodology does not require a secure setup phase for key exchange between the firmware sender and receiver because the decryption key is generated dynamically by the hardware. Furthermore, we leverage the ESG of dPPUF using a watchdog timer; if the dPPUF response and the key computation in time τ exceeds T_0 (5 cycles), the operation is halted (step 2).

C. Security Analysis

Threat Model: We consider that the firmware packaging from the manufacturer is an error-free process taking place in a secure facility: that is the firmware package is prepared correctly. The firmware is transferred to the device over an insecure channel. An attacker, able to intercept the package, aims to reverse-engineer the firmware image in order to place back-doors and uncover proprietary device operations.

Analysis: For security requirements, we need to ensure that adversaries able to intercept the communication medium and acquire the firmware cannot reveal code subroutines while also guarantying the image integrity. Thus, the proposed process requires that: the firmware header must provide no useful information to the attacker, restrict the retrieval of the encryption key, and in the event of a corrupted package, the device should be able to detect it. The utilized NIST-approved security standards AES-GCM and SHA-256, which are well-documented established security primitives, can fulfill the above. The header cannot be leveraged by attackers: IV and h_k rely on the one-way property of hash functions, hence it is practically impossible to reconstruct both the encryption key k and $f_i || f_v$. Also, the attacker's acquisition of the input challenge c provides not benefit since even an accurate simulation will determine the responses slower than the real-time behavior of the PPUF hardware [23]. Finally, we incorporate mechanisms, as described in Section IV-B (see step 4), that are able to detect firmware corruption.

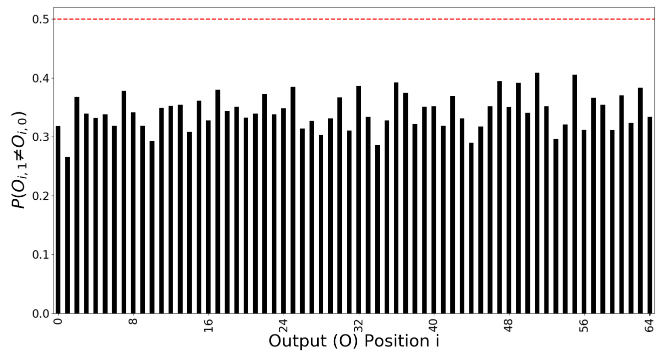

Fig. 4. Strict Avalanche Criterion (SAC) of dPPUF model. The red dashed line shows the ideal case, where $P(O_i = 1) = 0.5$ for all i.

The 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. In addition, the public model of dPPUF ensures that profile characterization (e.g., power profile) of the circuit would not reveal any new side-channel information. Man-in-the-middle attacks are hard to occur because the watchdog timer does not provide enough time for an attacker to interact (transmit and receive data in time) with the dPPUF-enabled device.

V. Experimental Setup and Results

To validate and evaluate this approach, we implement both the firmware packaging and unpacking procedures. Specifically, the packaging is implemented in software on a 64-bit machine with 3.2GHz Intel Core i5-4460 quad-core processor, with 8GB RAM, while the unpacking process scheme is implemented on a Xilinx Kintex7 FPGA with a system clock frequency of $100MHz$.

The simulation model of dPPUF is developed in *C++*. We conduct comprehensive simulations using multiple layers of boosters (2-input XOR gate) followed by repressers (small NAND-based circuit [19]), i.e., $b = 1$ and $r = 1$ with the height and width of the dPPUF being $h = 10$ and $w = 256$, respectively. The model is validated with $10k$ input vectors. Fig. 4 presents the Strict Avalanche Criterion (SAC); the correlation probability of the corresponding outputs of two input vectors that differ at exactly one position, $P(O_{i,1} \neq O_{i,0})$. The average probability of each output switching is 0.3425, similar to the results in the related literature [19].

A random challenge $c = [x_0, x_1]$ is fed to the dPPUF simulation model to generate the output response pair $p = [y_0, y_1]$, needed to create secret k. This secret is transferred to the rest of the framework. It is used as input to the hash and encryption algorithms implemented in Python using *pycryptodome*. The output of the software scheme at the manufacturer's side is the final binary file of the firmware image $f_p =< r, m_{enc}, t_{128} >$ where $r =< IV, h_k, c >$ and $m =< f_i || f_v >$. The total

TABLE I
FPGA resource utilization for the firmware unpacking process

Resources	*AES-GCM*	*SHA-256*	*dPPUF*	*Overall Design*
Slice LUTs	2671 (45.26%)	1330 (22.53%)	766 (12.98%)	5901
Slice Registers	1568 (33.63%)	753 (16.15%)	256 (5.49%)	4663
IOBs (Internal)	403	298	514	0
IOBs (External)	0	0	0	130
Block RAM/FIFO	5(100%)	0	0	5
BUFG	1	1	1	1

TABLE II
Firmware Unpacking timings

Device	**Firmware** (kB)	**Decryption** (ms)	**Total execution** (ms)	**dPPUF overhead** (%)
Sercos III	233	0.49186	0.49291	0.21325
Zelio Logic	323	0.68526	0.68630	0.15165
Modicon	1183	2.50159	2.50263	0.04156

data overhead from the header r and tag t_{128} is 124 bytes. We choose three commercial firmware files acquired from the vendors' websites to be tested. The images are designed for embedded systems deployed in environments such as ICS: a Sercos III field bus interface module, a Zelio Logic SR2/SR3 smart relay, and a Modicon M258 logic controller.

Each gate of the dPPUF model shown in Fig. 1 is described with unique switching delays to emulate manufacturing variation. Table I shows the hardware resources utilized by Xilinx Vivado Design Suite 2018.3 for performing the synthesis and implementation of the design. The hardware overhead for each security primitive is shown, as well as their respective percentage of hardware usage in regards to the overall design. As expected, the AES-GCM decryption module dominates other modules in terms of hardware overhead due to being the most heavy-weight component used in our design. The SHA-256 module uses a reasonable amount of hardware while the overhead of the implemented dPPUF is 12.98% of the total design, on Slice LUTs, and 5.49% on Slice Registers, which is minimal compared with hardware designs including the described crypto-modules.

The unpacking process is developed using hardware description language (HDL) to demonstrate the effectiveness of our approach directly on hardware. The produced firmware package 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* with read/write width of 128 bits. Since read and write cannot occur simultaneously in the same cycle in systems with low throughput such as resource-constrained embedded devices, the motivation of this approach is to bottleneck the system performance and determine if constrained embedded devices would be prohibitive in terms of execution time.

During the unpacking process of the firmware, the dPPUF input challenge c from the firmware header r is used to recreate the key of AES-GCM k. The encrypted firmware file, the IV, and the authentication tag header r are fed to AES-

TABLE III
COMPARISON WITH PREVIOUS WORK

Method	Area Overhead			Performance (Bits/Secs)
	LUTs	Registers	Block Ram	
Proposed	5901	4663	5	37.82×10^6
[15]	3543	1275	8	2,585.2
[18]	6038	1724	N/A	N/A

GCM to decrypt and authenticate the firmware package. The performance overhead, when compared with existing schemes that store the key within memory modules, is negligible. Table II shows the decryption and total execution time for unpacking each of the three firmware images. The major overhead is because of the decryption algorithm. Our proposed scheme using the dPPUF-based generated key k has, on average, a 0.136% timing overhead for the tested firmware images when compared to a decryption-only approach.

Finally, we provide in Table III a comparison with the most relevant state-of-the-art methods in terms of area and performance overhead. Performance is measured in bits/secs which is the rate of data sent from the prover (firmware package) to the verifier (embedded system) over the total time required for this operation. For our measurements, the Sercos III firmware package was selected, as seen in Table II.

In [15] and [18], PUF-based privacy-preserving authentication protocols are being considered. When solely comparing area overhead, [15] is lighter; however, it does require an initial setup and an enrollment phase (on top of PUF hardware characterization), steps our approach does not require, while relying on a third-party trusted server to complete the authentication handshake. Thus, the required hardware is offloaded to components other than the PUF-enabled devices. Moreover, the encryption mechanism is a weaker/lightweight 128-bit SIMON cipher. Our methodology shows a significant advantage in performance while preserving higher levels of security. In terms of area, [18] is comparable. The difference is attributed to the lack of a hash function module. A direct comparison with other related work, such as [16] is not possible due to limited quantitative data. The work in [16] does not require any third-party or secure enrollment phase; however, all cryptographic algorithms are implemented in software.

VI. CONCLUSIONS

In this paper, we developed a secure firmware update scheme that leverages the IC physical characteristics of embedded systems. Each update is unique to the embedded device ensuring the integrity and confidentiality of the firmware image. Our approach's flexibility provides the option of selecting alternative underlying security primitives, based on security and cost/performance requirements. A proof-of-concept implementation with commercial embedded devices firmware images has been presented. Experimental results verify its practicality and effectiveness, which demonstrates that our framework can be deployed in resource-constraint and/or mission-critical environments with minimal performance overhead.

ACKNOWLEDGMENT

This work has been supported by the European Unions 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.

REFERENCES

[1] National Crime Agency UK, "The cyber threat to uk business," *National Cyber Security Centre*, 2017-18.
[2] C. Eaton, "Hacked: Energy industry's controls provide an alluring target for cyberattacks," [Online]: http://www.houstonchronicle.com/, 2017.
[3] G. E. Suh and S. Devadas, "Physical unclonable functions for device authentication and secret key generation," in *Design Automation Conference, 2007. DAC'07. 44th ACM/IEEE*. IEEE, 2007, pp. 9–14.
[4] M. Potkonjak and V. Goudar, "Public physical unclonable functions," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1142–1156, 2014.
[5] A. Costin *et al.*, "A large-scale analysis of the security of embedded firmwares." in *USENIX Security Symposium*, 2014, pp. 95–110.
[6] C. Konstantinou, A. Keliris, and M. Maniatakos, "Taxonomy of firmware trojans in smart grid devices," in *Power and Energy Society General Meeting (PESGM), 2016*. IEEE, 2016, pp. 1–5.
[7] Z. Basnight *et al.*, "Firmware modification attacks on programmable logic controllers," *International Journal of Critical Infrastructure Protection*, vol. 6, no. 2, pp. 76–84, 2013.
[8] C. Konstantinou and M. Maniatakos, "Impact of firmware modification attacks on power systems field devices," in *Smart Grid Communications, 2015 IEEE International Conference on*. IEEE, 2015, pp. 283–288.
[9] A. Cui, M. Costello, and S. J. Stolfo, "When firmware modifications attack: A case study of embedded exploitation." in *NDSS*, 2013.
[10] C. Konstantinou and M. Maniatakos, "Hardware-layer intelligence collection for smart grid embedded systems," *Journal of Hardware and Systems Security*, pp. 1–15, 2019.
[11] C. O'Flynn and Z. D. Chen, "Side channel power analysis of an aes-256 bootloader," in *Electrical and Computer Engineering (CCECE), 2015 IEEE 28th Canadian Conference on*. IEEE, 2015, pp. 750–755.
[12] W. Li, Y. Wang, H. Li *et al.*, "P 3 m: a pim-based neural network model protection scheme for deep learning accelerator," in *Proceedings of the 24th ASP-DAC*. ACM, 2019, pp. 633–638.
[13] O.-M. Brisbanne and L. Bossuet, "Restoration protocol: Lightweight and secure devices authentication based on puf," in *IFIP/IEEE International Conference on Very Large Scale Integration, VLSI-SoC 2017*, 2017.
[14] M. Rostami *et al.*, "Robust and reverse-engineering resilient puf authentication and key-exchange by substring matching," *IEEE Transactions on Emerging Topics in Computing*, vol. 2, no. 1, pp. 37–49, 2014.
[15] A. Aysu *et al.*, "End-to-end design of a puf-based privacy preserving authentication protocol," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2015, pp. 556–576.
[16] F. Wei *et al.*, "Secure code updates for smart embedded devices based on pufs," Cryptology ePrint Archive, Report 2017/991, 2017.
[17] U. Chatterjee, R. S. Chakraborty, and D. Mukhopadhyay, "A puf-based secure communication protocol for iot," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 16, no. 3, p. 67, 2017.
[18] W. Che *et al.*, "A privacy-preserving, mutual puf-based authentication protocol," *Cryptography*, vol. 1, no. 1, p. 3, 2016.
[19] M. Potkonjak *et al.*, "Differential public physically unclonable functions: architecture and applications," in *Proceedings of the Design Automation Conference*. ACM, 2011, pp. 242–247.
[20] B. Colombier, L. Bossuet, V. Fischer *et al.*, "Key reconciliation protocols for error correction of silicon puf responses," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 8, pp. 1988–2002, 2017.
[21] Y. Dodis *et al.*, "Fuzzy extractors: How to generate strong keys from biometrics and other noisy data," *SIAM journal on computing*, vol. 38, no. 1, pp. 97–139, 2008.
[22] F. Koushanfar, P. Boufounos, and D. Shamsi, "Post-silicon timing characterization by compressed sensing," in *Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design*. IEEE Press, 2008, pp. 185–189.
[23] N. Beckmann and M. Potkonjak, "Hardware-based public-key cryptography with public physically unclonable functions," in *International Workshop on Information Hiding*. Springer, 2009, pp. 206–220.