A Reconfigurable and Secure Firmware Updating Framework for Advanced Metering Infrastructure

Prosanta Gope
Department of Computer Science
The University of Sheffield, UK
p.gope@sheffield.ac.uk

Biplab Sikdar
Department of ECE
National University of Singapore
bsikdar@nus.edu.sg

***Abstract*—Smart meters play an important role in modern power grids by providing fine-grained power consumption data and enabling services such as dynamic pricing and demand-side management. The smart metering devices are firmware-driven, where it is important that the devices be able to securely update their firmware on a regular basis to fix bugs, and improve as well as add services. In this paper, we propose a *new* privacy-aware secure firmware-updating framework called PRSUF (Privacy-aware Reconfigurable Secure-Firmware Updating Framework) to securely update the firmware in smart metering devices. The proposed the framework allows a hardware intrinsic secret to being updated and stored in a secure and efficient way. One of its key differentiating features is that, unlike existing mechanisms, the proposed scheme does not require storing any keys in the meter's non-volatile memory (NVM), thereby making it is secure against a number of physical and side-channel attacks. As compared to state-of-the-art solutions, the proposed security framework has notable features such as reconfigurability, protection against cloning and downgrading, detection of theft of services and tampering with the firmware and the hardware, etc.**

***Index Terms*—Advanced metering infrastructure, Smart Meter, Secure firmware update, Downgrading attack, Privacy.**

I. Introduction

The Industrial Internet of Things (IIoT) promises to usher in a slew of new communications technologies centred on industrial applications. IIoT-based smart grid (SG) technology, in particular, is expected to play a key role in the next-generation power grid system. The smart grid is an electrical framework built on the Internet of Things (IoT) that integrates information and communication technology (ICT) with bi-directional intelligent systems across energy generation, transmission, distribution, and consumption in order to create a system that is sustainable, secure, dependable, robust, and cost-effective. A key feature of smart grids is the integration of two-way power flow along with communications and control [2]-[4]. Intelligent two-way devices such as smart meters, sensors, and actuators are used throughout a smart grid from power production to consumption. As a critical infrastructure, the smart grid and its components are immensely attractive to adversaries. For instance, incidents of power outages in Ukraine in December 2015 were directly triggered by cyber-attacks, leaving thousands of people without electricity for prolonged periods of time and causing enormous financial losses. Other studies have highlighted the possibility of catastrophic but feasible cyber-attacks on the power grids (e.g., the UK and US grids), which would lead to large scale power outages and financial losses [1]. Therefore, it is essential to examine existing vulnerabilities as well as any potential threats to safety in the smart grid infrastructure and develop solutions to mitigate them [11]. The Advanced Metering Infrastructure (AMI) plays an important role in smart grids by providing a real-time, two-way communication channel from the grid operator to the consumer, thereby enabling services such as dynamic pricing, demand-side management, etc. Usually, there are two main components in the smart meter infrastructure: the smart meter and a server operated by a service provider or utility company. A smart meter is an electronic device that collects and records various data from the consumer premises, such as electricity consumption, voltage levels, current, and power factor, which are then sent to the service provider. The service provider uses a database server to store data, issue bills, update prices dynamically, and use the data to facilitate supply and demand management. However, the communication between the smart meter and service provider passes through the public Internet and wireless links and provides adversaries opportunities to intercept, eavesdrop, alter, and delete the messages, and then perform other serious attacks such as impersonation, Denial of Service (DoS), cloning, and tracking attacks.

A. Problem Statement and Motivation

The endpoints of AMI such as the smart meters are particularly vulnerable to attacks due to their limited computational capabilities and lack of sophisticated in-built security features. The Internet connectivity of smart meters can be exploited by attackers to discover new weaknesses and exploit emerging vulnerabilities reported about specific devices [4]. Smart meters are made up of a number of electronic components, such as a network interface card (NIC) and a micro-controller, which run firmware programs that control, monitor, and process the data in the device. Moreover, the firmware plays an important role in enabling the implementation of higher layers (such as the drivers, user interface, etc.). Vendors of smart meters periodically need to develop and upgrade the firmware in order to address discovered faults, enhance operation, and add new features to

978-1-6654-3254-2/22/$31.00 ©2022 IEEE

the device. However, unauthorized modification of firmware and its installation on vulnerable devices is a serious threat [23], with numerous examples of real-world demonstrations of attacks [24-26]. For example, dishonest users may try to modify their devices to avoid payments and malicious adversaries may push insecure firmware masquerading as firmware upgrades [5]. While efforts have been made to secure firmware updates, existing approaches have multiple drawbacks (e.g., the use of a common key to encrypt different firmware versions). In addition, the lack of awareness and excessive downtime and overhead associated with installing updates results in many smart-metering devices not receiving firmware updates or security patches [10]. In order to address the existing security issues associated with smart meter firmware, we propose a privacy-aware reconfigurable secure-firmware updating framework, called PRSUF, that can securely bind a firmware with the smart meter's unique, hardware-intrinsic cryptographic key. The use of device-specific keys tied to the hardware of the device protects against illegal firmware upgrades. In order to prevent the downgrading of firmware, when a new firmware version is distributed, the proposed scheme reconfigures the PRSUF, which results in a renewed hardware-intrinsic cryptographic key. With this feature, the smart meter will not be able to run the previous version of the firmware since the cryptographic key has changed.

The rest of the paper is organized in the following way. Section II provides a brief description of physically unclonable functions (PUFs), fuzzy extractors, and the adversary model considered in this paper. Section III presents the details of the proposed PRSUF along with a protocol concept. Security analysis of the proposed scheme is presented in Section IV. Section V provides the implementation details and finally, concluding remarks are given in Section VI.

II. Background and Related Work

A. Physically Unclonable Functions

Physically Unclonable Functions are most often physical circuits that map a binary input (i.e., a bit-string challenge) to a single or multi-bit output, called the response [6-7]. PUFs are based on exploiting the variations in the physical structure of each integrated circuit that is caused during the manufacturing process to create challenge-response mappings that are unique to a particular integrated circuit. These micro-variations in the physical structures allow strong entropy to be derived in order to generate unpredictable challenge/response behaviour, rather than through the use of a mathematical function. A PUF can be denoted as $R \leftarrow P(C)$ where R and C denote the challenge and response, respectively. The goal of PUFs is to enable strong security for authentication solutions that remain lightweight for scenarios where cryptographic resources are limited. Their assumed properties include being unclonable, unpredictable, reliable, and tamper-evident. PUFs can be divided into two types, Strong PUF and Weak PUF. This is not a distinction of their security, but rather how many CRPs they support. Weak PUFs support very few (or sometimes only one) CRPs, making them useful for a key generation [7]. For strong PUFs, the number of CRPs is significantly large that even attackers can access, having throughout knowledge of CRPs is impossible. In addition, since the strong PUF has a large set of CRPs and they are randomly selected in usage, the probability that the attacker has knowledge about the CRPs currently used is even smaller [7]. According to the exponential amount of CRPs explained above, the strong PUF is commonly used for device authentication. With the property of random deviation in manufacturing hardware, the strong PUF has an unclonable physical structure designed to prevent an attacker from easily modelling its behaviour. In this paper, we use a weak PUF for our construction of the secure firmware updating framework.

B. Fuzzy Extractor

The operation of PUFs can be quite sensitive to environmental and operating conditions, leading to a noisy PUF output. For reliability in noisy PUF environments, a fuzzy extractor (FE) may be used to eliminate of noise in the PUF response [8-9]. The fuzzy extractor technique is a tuple (M, Γ, t), where M represents a metric space, Γ denotes the bit length of the input, and t is the error tolerance threshold. This technique mainly includes two algorithms [41]: a probabilistic generation function Gen$(\cdot)$ and a deterministic reconstruction function Rec$(\cdot)$. The success of FE is based on the similarity of the original and the noisy output of the PUF.

C. Security Objectives

We mainly consider *five* important security requirements, i.e., *forward unpredictability*, *backward unpredictability*, *non-resettability*, *protection against downgrading attacks*, and *privacy of the smart meters* against any eavesdropping or tractability attacks.

- *Forward unpredictability*: In case of forward unpredictability, even if adversary $\mathcal{A}$ knows an adaptively chosen set of input/output pairs of the PRSUF for any previous state, $\mathcal{A}$ must not be able to predict the final output key (K) and current state (S).
- *Backward unpredictability:* In case of backward unpredictability, $\mathcal{A}$ must not be able to predict the output key (K), and the state S for any previous state (before reconfiguration), even if the adversary can adaptively obtain input/output pairs for the current state of the PRSUF.
- *Privacy:* Smart meters handle the private information of their users. Hence, it is desirable that communication between the server and the devices must be anonymous/confidential. An eavesdropper should not be able to identify the device and also should not be able to trace the device.
- *Non-resettability*: It must be infeasible for an adversary $\mathcal{A}$ to set the state of the PRSUF to the desired value.
- *Protection against downgrading attacks*: An adversary $\mathcal{A}$ should not be able to downgrade the firmware (running on a smart meter) to older versions and take advantage of the previously available features with security holes.

TABLE I
COMPARISON

Properties	[15]	[16]	[20]	[21]	[22]	Ours
Unpredictability	χ	√	χ	√	√	√
Privacy	χ	χ	χ	χ	√	√
Non-Resettability	√	χ	√	√	χ	√
Downgrading Attack Progression	√	χ	χ	√	χ	√
Physical-Security of SM	χ	χ	χ	χ	χ	√

- *Other objectives*: Apart from the security objectives listed above, the proposed scheme also considers a few more imperative security properties such as security against impersonation or forgery attacks, cloning attacks, replay attacks, etc.

D. Assumptions

First, it is assumed that a PRSUF instance is valid only for a dedicated session of the execution of the secure firmware update protocol, and it cannot be used in another session. In our proposed reconfiguration protocol, all activities made during the setup (i.e., enrollment) phase are assumed to be unavailable to any adversary. Therefore, an adversary can attempt to attack only during the reconfiguration phase. Finally, we assume that a PUF is present in every smart metering device, referred to as a system-on-chip (SoC). If the PUF is disconnected from the smart metering device, it is presumed to become useless and destroyed. Furthermore, we also assume that the adversary may have physical access to the smart meter and the proposed PRSUF construction. However, any attempt to tamper with the integrated PUF, or any changes to the device's function, renders it worthless.

E. Related Work

The use of PUFs has been explored in literature in the context of several security objectives. These include PUF-based solutions for authentication, firmware attestation, data provenance, as well as data integrity [11-14]. A traffic-aware firmware update mechanism for mobile IoT environments is proposed in [17] and is aimed at controlling the spread of malware. The mechanism relies on intermediate nodes for firmware updates and has high time complexity. There exists a large body of work on detecting attacks in IoT networks (see [18] for a survey), such as the one in [19], which uses a machine-learning-based model to detect such attacks. However, these mechanisms are not secure against physical attacks on the devices. Finally, analytic frameworks for patching have been proposed in the literature, such as in [20]. However, such techniques are based on the frequent exchange of update messages, which is not always practical. In the context of firmware updates for smart meters, a safe firmware update technique for devices connected to an alternating current network is proposed in [15] to avoid malicious firmware upgrades. In their scheme, a pre-defined sequence of variations in base frequency opens an update window in which devices receive firmware update requests. In [16], the authors described a technique for remote firmware updating for devices

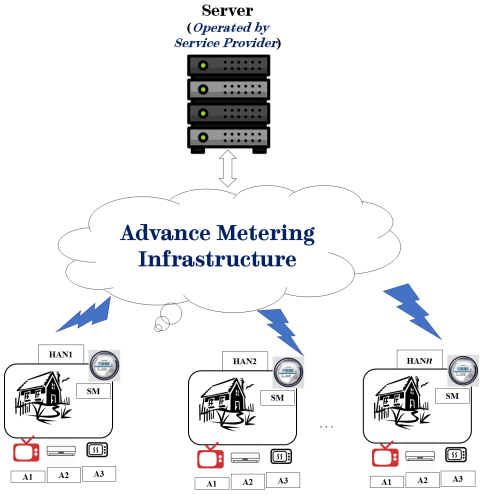

Fig. 1. Smart metering infrastructure.

in AMI networks and introduced firmware update management and network service management systems. Likewise, [21] provided an example of how to perform a remote firmware update over the AMI network. In [22], the authors introduced a new smart meter firmware update mechanism using random linear network coding and attribute-based signcryption (CP-ABSC). To the best of our knowledge, this paper is the *first* that combines privacy, reconfigurability, and security in designing a reliable firmware update protocol for smart-metering infrastructure (as shown in Table 1).

III. Proposed Scheme

In this section, we first present the proposed reconfigurable secure-firmware updating framework and subsequently, we introduce a protocol to show how the proposed PRSUF can be applied in practice.

A. System Model

The system model considered in this paper is shown in Fig. 1, and consists of *three* major entities: a server operated by a utility service provider (USP), a group of home area networks (HANs), and smart meters (SMs). Border router elements connect smart-metering devices to the verifier over the Advanced Metering Infrastructure (e.g., based on 6LoWPAN and ZigBee).

B. PRSUF Construction

We now introduce a reconfigurable construction of a secure-firmware updating framework (as shown in Fig. 2), which is the underlying foundation of the proposed protocol (presented in Section III.B). The proposed construction consists of *five* major components: a control logic circuit and a conventional weak PUF (such as static random-access memory (SRAM) PUF), fuzzy extractor, two non-volatile memories (NVMs), and a one-way collusion-resistant hash function. The control logic circuit consists of a dedicated function: $\mathrm{Recon}(\cdot)$. To change the settings of the PRSUF, the circuit uses the

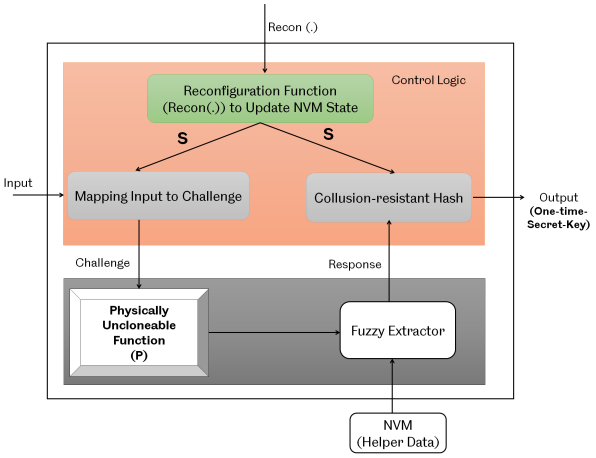

Fig. 2. PRSUF: Reconfigurable firmware updating framework.

$\mathrm{Recon}(\cdot)$ function for reconfiguring its current state S to a new independent random state $S^{'} \leftarrow \mathrm{Recon}(S)$ (when required), and this state $(S^{'})$ is then maintained in a secure private NVM (non-volatile memory) of the device. The error correction on the noisy PUF output R is performed by using a *Fuzzy Extractor* (FE) and helper data stored in a public NVM. Next, the output of the FE and the current state is used to generate a secret round key (K) through a one-way collusion-resistant hash function, which is only valid for a specific round. The secret round key is used to encrypt/decrypt the new version of the firmware with instructions or patches. During the execution of the secure firmware update protocol (described in Section III.B), the $\mathrm{Recon}(\cdot)$ function is called to update the PRSUF's state and reconfigure it, i.e., $S^{'} \leftarrow h(S)$. The function $(\mathrm{Recon}(\cdot)$ and the one-way collusion-resistant hash function $h(\cdot))$ are publicly known (e.g., well-known hash functions). An adversary $\mathcal{A}$ will not be able to control or change the state to a value of its choice. Details related to the construction of the proposed PRSUF are depicted in Fig. 2 and the implementation details of PRSUF are presented in Section V.

C. Secure Firmware Updating Protocol for the Smart-metering Devices

This section presents the details of the operation of the proposed PRSUF and describes how it can be applied at the protocol level to securely update firmware for smart meters. The proposed protocol consists of two phases: *Setup Phase* and *Reconfiguration Phase*.

1) **Setup Phase:** A client/customer first needs to register its smart meter through a predefined setup phase, before it is used in the field. In the setup phase of the protocol, the smart meter M_x first sends a request message with its unique identity, i.e., $\mathrm{Set}_1 : \{Reg_{req}, M_x\}$ to the server. After receiving message Set_1, the server generates an initial state S_0 and a random number τ, and then derives the first state $S_1 = h(M_x||S_0)$ and the first secret round key $K_1 = h(M_x||S_1)$. Then, the server encrypts the first version of the firmware swv_1 using the secret key K_1, i.e., $\Delta = E_{K_1}[swv_1]$ and also generates a one-time temporary identity $TID^1 = h(\tau||M_x||mk)$, where mk denotes the master key of the server. Finally, the server composes a message $\mathrm{Set}_2 : \{S_0, TID^1, \Delta\}$, sends it to the device through a secure channel, and also stores $\{D_x, S_1, TID^1\}$ in its database for further communication. Upon receiving message Set_2, the smart-meter generates the first state $S_1 = h(M_x||S_0)$ and a random challenge C. Then, the smart meter extracts the PUF output $R \leftarrow P_M(C)$ and then derives the helper data $w = \mathrm{FE.Gen}(R, M_x)$. Finally, the device stores $\{w, TID^1, \Delta\}$ in its public NVM and also stores $\{S_1, C\}$ in its private NVM. Now, the smart meter can decrypt Δ using the secret round key K_1 and install the firmware swv_1.
2) **Reconfiguration (Firmware Updating) Phase:** In order to upgrade the firmware from swv_i to swv_{i+1}, the system needs to be completely reconfigured for the usage of a new key by executing the reconfiguration protocol (as shown in Fig. 4). In this regard, both the smart meter and the server need to update the shared secret round key from the previous round (i.e., K_i) to K_{i+1}. This phase of the protocol consists of the following steps.

Step #1: The client/consumer who wants to upgrade the firmware running on its smart meter forms a update request message, i.e., $\mathrm{MSG}_1 : \{\mathrm{Req_{up}}, TID^i\}$ and then sends MSG_1 to the server.

Step #2: Upon receiving message MSG_1, the server first checks the one-time temporary identity TID^i in its database and if the sever cannot finds TID^i, then it aborts the execution of the protocol. Otherwise, the server loads the current state S_i in its memory and computes $K_i = h(D_x||S_i)$. Then, the server creates a "*Rcnf*" message and encrypts the message using K_i. Next, the sever derives $\sigma_i = h(\Delta_i||K_i)$ and composes a response message $\mathrm{MSG}_2 : \{\Delta_i = E_{K_i}[Rcnf], \sigma_i\}$ and sends it to the smart meter.

Step #3: After receiving message MSG_2, the smart meter first loads the challenge C in its memory from the private NVM and then extracts the noisy PUF output $R^{'} = P_M(C)$ and derives its id $M_x = \mathrm{FR.Rec}(R^{'}, w)$ and the secret round key $K_i = h(M_x||S_i)$. Then, the smart meter verifies the parameter σ_i. If the verification is successful, then the device decrypts Δ_i and obtains the "*Rcnf*" message. Subsequently, it generates a "*Rconf_ok*" message and reconfigures the current state S_i to $S_{i+1} = h(S_i)$, and computes $K_{i+1} = h(M_x||S_{i+1})$, $E_{K_{i+1}}[Rcnf\_ok]$, and $\sigma_{i+1} = h(E_{K_{i+1}}(Rcnf\_ok)||K_{i+1})$. Finally, the device composes a message $\mathrm{MSG}_3 : \{E_{K_{i+1}}[Rcnf\_ok], \sigma_{i+1}\}$ and sends it to the sever.

Step #4: Upon receiving message MSG_3 from the smart meter, the server reconfigures S_i using $S_{i+1} = h(S_i)$ and generates the new shared round key $K_{i+1} = h(M_x||S_{i+1})$. Then, the server decrypts $E_{K_{i+1}}[Rcnf\_ok]$,

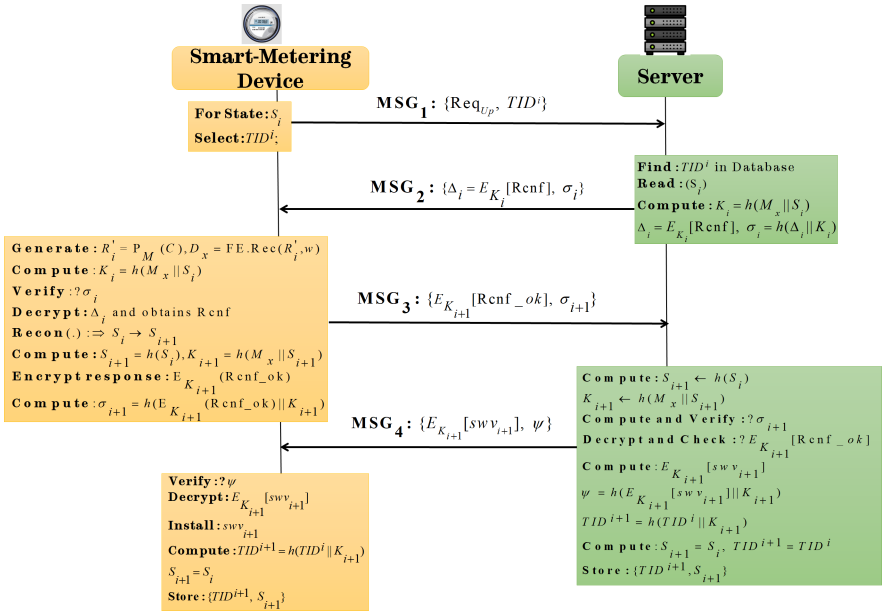

Fig. 3. Reconfiguration phase of the proposed secure firmware updating protocol.

and checks the response. If the server successfully validates the response, then the server uses the updated key K_{i+1} to encrypt the new version of the firmware with instructions or patches, i.e., $E_{K_{i+1}}[swv_{i+1}]$ and also derives $\psi = h(E_{K_{i+1}}[swv_{i+1}]||K_{i+1})$ and $TID^{i+1} = h(TID^i||K_{i+1})$. Next, the sever composes a message $\text{MSG}_4 : \{E_{K_{i+1}}[swv_{i+1}], \psi\}$ and sends it to the smart meter through the Internet. Subsequently, it replaces TID^i with TID^{i+1} and S_i with S_{i+1}.

Step #5: After receiving message MSG_4, the smart meter first checks ψ (in order to validate the integrity of the message). If the smart meter validates the parameter ψ successfully, then it decrypts $E_{K_{i+1}}[swv_{i+1}]$ and runs the updated version of the firmware. Finally, the device reconfigures its identity $TID^{i+1} = h(TID^i||K_{i+1})$ and updates TID^i with TID^{i+1} and S_i with S_{i+1}.

IV. Security Evaluation

This section analyzes the security of our proposed scheme. In this regard, we show our PRSUF construction (the underlying foundation of the proposed firmware updating protocol) is secure, and for that, we define an unpredictability game.

A. Unpredictability Property of PRSUF

The unpredictable behaviour of a PRSUF is defined by a game between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$, which is a desirable imperative quality to provide security against any known attacks (such as man-in-the-middle attacks or impersonation attacks).

Setup: Challenger $\mathcal{C}$ issues a PRSUF to adversary $\mathcal{A}$.

Queries: $\mathcal{A}$ queries the PRSUF Φ times using challenges α_i (where $1 \leq i \leq \Phi$) and receives the PUF output β_i ($\beta_i \leftarrow PRSUF_M(\alpha_i)$).

Output: At the end of the game, $\mathcal{A}$ outputs a CRP pair (α^*, β^*).

We say $\mathcal{A}$ wins the game if he/she can output a valid PUF response β^*. Otherwise, the behavior of the PUF is unpredictable and no polynomial adversary can predict the PUF output with significant success probability.

Definition 1. *A PUF is said to be (q, ϵ)-unpredictable if there is no ppt (probabilistic polynomial time) adversary $\mathcal{A}$ that issues at most q queries to the PRSUF and can win the game with probability greater than ϵ.*

Backward and Forward Unpredictability: Next, we define backward- and forward-unpredictability of a PRSUF in terms of a two-stage game between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$. In the first stage, $\mathcal{A}$ is given oracle access (i.e., access to the interface) of the PRSUF, from which $\mathcal{A}$ can obtain challenge/response pairs (CRPs) at will. This stage models the ability of $\mathcal{A}$ to obtain challenges and responses (with respect to a fixed internal PRSUF state) by passive eavesdropping. We also give $\mathcal{A}$ access to the internal PRSUF state in order to model hardware attacks against the PRSUF implementation. Once $\mathcal{A}$ has learned enough CRPs, the challenger performs the reconfiguration operation and finally gives $\mathcal{A}$ oracle access to the reconfigured PRSUF such that $\mathcal{A}$ can obtain CRPs of the reconfigured PRSUF. At the end of the game, $\mathcal{A}$ needs to output a non-trivial CRP $(\alpha^{\#}, \beta^{\#})$.

More formally, $\mathcal{A} = ((\mathcal{A}_{\S}, \mathcal{A}_{\dagger}))$ consists of two probabilistic polynomial time algorithms, where $\mathcal{A}_{\S}$ interacts with the PUF before reconfiguration and $\mathcal{A}_{\dagger}$ thereafter. A engages in the following experiment:

Setup: The adversary $\mathcal{A} = (\mathcal{A}_{\S}, \mathcal{A}_{\dagger})$ is given an arbitrary state ς of the PRSUF by the challenger $\mathcal{C}$ who sets up an PRSUF. Then, in Phase 1, $\mathcal{A}_{\S}$ queries the PRSUF up to q_x

times and at the end of this phase, $\mathcal{A}_{\S}$ stops and outputs to the log file $\mathcal{F}$ that is used as input to $\mathcal{A}_{\dagger}$.

Reconfiguration: Now, the challenger $\mathcal{C}$ resets the PRSUF, which updates its internal state to ς^*. Then, in Phase II, $\mathcal{A}_{\dagger}$ is initialized with state ς^* and the log file $\mathcal{F}$ from $\mathcal{A}_{\S}$. Now, $\mathcal{A}_{\dagger}$ is allowed to query the reset PRSUF up to q_y times.

Outputs: At the end of the game, $\mathcal{A}_{\dagger}$ outputs a non-trivial CRP $(\alpha^{\#}, \beta^{\#})$ of the PRSUF.

We say that $\mathcal{A}$ wins the *forward-unpredictability* game if $\beta^{\#}$ is a valid PRSUF response to query $\alpha^{\#}$ that was not included in the q_y queries. Therefore, with this unpredictability, once the PRSUF is reset, the adversary cannot output a valid CRP for the reset PRSUF. On the other hand, we say that $\mathcal{A}$ wins the *backward-unpredictability* game if $\beta^{\#}$ is a non-trivial (valid) PRSUF output to the query $\alpha^{\#}$ that was not part of the q_x queries. This unpredictability implies that an adversary with access to the PRSUF will not be able to predict a valid response of the PRSUF *before the reset* happened. Accordingly, a PRSUF is *backward (or forward) unpredictable*, when there is no PPT adversary $\mathcal{A}$ that can win the game with significant success probability.

Definition 2. *A PRSUF is said to be a (q_x, q_y, ε)-secure backward and forward unpredictable PUF if there is no PPT adversary $\mathcal{A}$ who makes at most q_x queries in Phase 1 and at most q_y queries in Phase II, is able to win the above backward and forward unpredictability game with probability greater than ε.*

B. Security Analysis

Using the security model that was previously mentioned, we present a security analysis of the suggested PRSUF architecture in this section.

Theorem 1: *The proposed reconfigurable construction can ensure forward and backward unpredictability, if the underlying $PhysicalFunction(PUF)$ is a (Φ, ϵ)-unpredictable PUF and h is a secure one-way collision-resistant hash functions.*

Proof. To prove this theorem, we use the above *backward and forward unpredictability game*, where an adversary $\mathcal{A}$ is allowed to access of the PRSUF attached with the device and to obtain a set of CRPs from that. Assume that $\mathcal{A} = (\mathcal{A}_{\S}, \mathcal{A}_{\dagger})$ breaks the *backward-and forward unpredictability* of the PRSUF with non-negligible probability. We now construct an adversary $\mathfrak{B}$ that breaks the unpredictability of the underlying physical PRSUF with the same success probability as $\mathcal{A}$. $\mathfrak{B}$ selects an arbitrary state ς of the PRSUF then passes it to $\mathcal{A}_{\S}$ and executes a black-box simulation of the challenger $\mathcal{C}$ of the *backward and forward unpredictability game* (shown in Theorem 1). Now, for a challenge α_j received from $\mathcal{A}_{\S}$, $\mathfrak{B}$ queries the PRSUF and stores (α_j, β_j) in a log file $\mathcal{F}$ and forwards β_j to $\mathcal{A}_{\S}$. At some point, $\mathcal{A}_{\S}$ stops and outputs some log file $\mathcal{F}^*$. After that, $\mathfrak{B}$ changes the PRSUF state to ς^* for resetting the configuration of the PRSUF. Next, $\mathfrak{B}$ initializes $\mathcal{A}_{\dagger}$ with state ς^* and log file $\mathcal{F}^*$ and continues to simulate $\mathcal{C}$. Now, when $\mathcal{A}_{\dagger}$ sends a challenge α_j, $\mathfrak{B}$ queries the PRSUF and stores (α_j, β_j) in a log file $\mathcal{F}$ and forwards β_j to $\mathcal{A}_{\dagger}$. Finally, $\mathcal{A}_{\dagger}$ stops and outputs a CRP $(\alpha^{\#}, \beta^{\#})$ of the PRSUF. Since $\mathfrak{B}$ has never queried $\alpha^{\#}$ to the PRSUF, this contradicts the unpredictability property of the PRSUF. Hence, the success probability of $\mathfrak{B}$ is similar as $\mathcal{A}$. As mentioned before, the security of the proposed scheme is based on the unpredictability property of the PRSUF, where an adversary should not be able to predict any PUF response for a given challenge. Therefore, no adversary can differentiate the encoded PRSUF outputs such as $X = \beta_i^{2\dagger} \oplus k_i$ and $\beta^*_{i+1} = \beta_{i+1} \oplus k_i$ from a randomly chosen string. Hence, our proposed authentication scheme is secure against under unpredictability game. ■

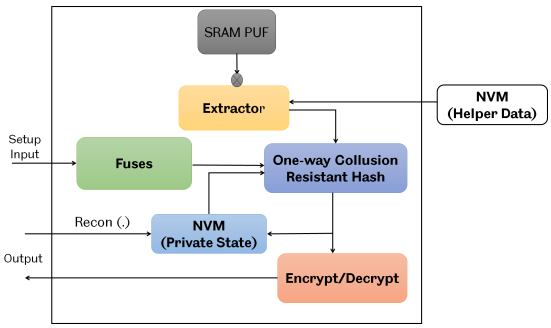

Fig. 4. Example of the proposed PRSUF architecture.

TABLE II
IMPLEMENTATION COST ESTIMATE FOR PRSUF IN GATE EQUIVALENT (GE)

Component	Cost Estimate	Comments
PUF	14,743 GE	SRAM-PUF
Private NVM	1,367 GE	256-bit 2T MTP NVM
Public NVM	-	EEPROM
FE	7,456 GE	BCH Encoding
Control logic	4,635 GE	MUXing, I/O, etc.
Hashing	23,780 GE	SHA-256
Encryption/Decryption	12,315 GE	128-bit AES-CBC mode
Fuses	156 GE	2T OTP anti-fuses

V. IMPLEMENTATION DETAILS AND COST EVALUATION

In order to estimate the number of resources required for the implementation of the proposed PRSUF framework, we consider an example design architecture of the proposed PRSUF (as shown in Fig. 4). Based on our design, the following components are required:

- **PUF:** SRAM memory (2 MB)
- **Fuzzy Extractor:** Golay(24, 12, 8) code (for constructing the device's ID D_x)
- **Public NVM:** Assumed to be an electrically erasable programmable read-only memory (EEPROM), external to the SoC embedding the PRSUF, for storing helper data (2 MB). The size of the public-NVM needs to be the same as that of the SRAM.
- **Private NVM:** 256 bits 2T multi-time programmable (MTP) private NVM (also known as logic NVM) to store the state.
- **Hashing Functionality:** We use the collision-resistant SHA-256 algorithm.

- **Encryption/Decryption:** AES-CBC 128-bit has been used for the required encryption/decryption in PRSUF.
- **Fuses:** Here, we use one-time programmable fuses to manage (disable or enable based on the requirement) the data and control paths associated with enrollment.

Table II shows the estimated cost of implementing the proposed PRSUF construction, where, for each component of the PRSUF, an area estimation in Gate Equivalent (GE) has been considered. From Table II, we can see that as compared to a conventional PUF implementation (such as SRAM PUF, which is still vulnerable to downgrading attacks), PRSUF incurs some additional resource costs (in terms of NVMs, hashing, etc.), which is acceptable for protection against *downgrading attacks*. In this regard, the proposed PRSUF framework is reconfigured with the new version of the firmware, where the result of each reconfiguration process is an unpredictable secret key K. Updating the key on a regular basis also provides protection against any older version of the firmware from being run on the device. Furthermore, if we assume that the attacker has knowledge of a previously valid state, it will still be difficult for an adversary to write any arbitrary data to the NVM due to the presence of a hash function in the NVM write path. However, this mechanism does not prevent an attacker from attempting to use a previously known state S_{i-1} in order to write S_i into the NVM. The proposed PRSUF construction can prevent such attacks by disabling the external NVM write path after enrollment (as shown in Section V). Besides, the proposed PRSUF framework can guarantee security against *cloning attacks*, which involves extracting the secret from a targeted smart metering device and copying it to form a clone. In the case of PRSUF, the secret state is stored in the private NVM and the underlying PUF is unclonable. Hence, it will be difficult for an attacker to obtain the secret state and also to replicate the PUF responses in the cloned system. Finally, since the smart-meter uses a different temporary id (TID) in each session that helps to preserve privacy of the system.

VI. Conclusion

In this paper, we presented a reconfigurable firmware updating framework for smart meters that base its security on a combination of the physical properties of a PUF and secret state information stored in a private NVM. From the analyses, it can be argued that the proposed construction is secure against some of the imperative attacks such as man-in-middle, cloning, and downgrading attacks. Although the proposed construction requires a little more additional resources as compared to traditional PUFs, we believe that this cost is acceptable for the sake of security and new functionalities.

References

[1] E. Oughton et al., "Stochastic counterfactual risk analysis for the vulnerability assessment of cyber-physical attacks on electricity distribution infrastructure networks," Risk Analysis, vol. 39, no. 9, pp.2012-2031, 2019.

[2] Q. Yang et al., "Advanced power electronic conversion and control system for universal and flexible power management" *IEEE Trans. Smart Grid*, vol. 2, pp. 231-243, Jun. 2011.

[3] P. McDaniel and S. McLaughlin, "Security and privacy challenges in the smart grid", *IEEE Security Privacy*, vol. 7, no. 3, pp. 75-77.

[4] A. Spadafora, "Smart-metering Devices Still Major Target for Cyberattacks."

[5] L. Ilascu, "When their firmware is vulnerable, its up to you to protect your smart devices," Accessed: 5 May 2019.

[6] P. Gope, and B. Sikdar, "A Comparative Study of Design Paradigms for PUF-based Security Protocols for IoT Devices: Current Progress, Challenges and Future Expectation," *IEEE Computer Magazine,* DOI: 10.1109/MC.2021.3067462, 2021.

[7] P. Gope, and B. Sikdar, "A Privacy-Aware Reconfigurable Authenticated Key Exchange Scheme for Secure Communication in Smart Grids," *IEEE Transactions on Smart Grid*, DOI: 10.1109/TSG.2021.3106105, 2021.

[8] Y. Dodis et al., " Fuzzy extractors: How to generate strong kesy from biometrics and other noisy data." In Advances in Cryptology - *EUROCRYPT* '2004, Lecture Notes in Computer Science. Springer-Verlag, Berlin Germany, 2004.

[9] C. B¨osch, J. Guajardo, A.-R. Sadeghi, J. Shokrollahi, and P. Tuyls. Efficient helper data key extractor on fpgas. *In Proceedings of CHES*, pages 181–197, 2008.

[10] S. Cheng, et al., "Traffic-Aware Patchin for Cyber Security in Mobile IoT," in *IEEE Communications Magazine*, vol. 55, no. 7, pp. 29-35, July 2017.

[11] P. Gope and B. Sikdar, "Lightweight and Privacy-Preserving Two-Factor Authentication Scheme for IoT Devices," in *IEEE Internet of Things Journal*, vol. 6, no. 1, pp. 580-589, January 2019.

[12] X. Li et. al., "A Robust ECC-Based Provable Secure Authentication Protocol With Privacy Preserving for Industrial Internet of Things," in *IEEE Trans. Ind. Informat.*, vol. 14, no. 8, pp. 3599-3609, August 2018.

[13] M. Aman et al., "HAtt: Hybrid Remote Attestation for the Internet of Things with High Availability," in *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 7220-7233, August 2020.

[14] J. Kong, et al. "PUFatt: Embedded platform attestation based on novel processor-based PUFs," in *Proc. ACM/EDAC/IEEE Design Automation Conference (DAC)*, San Francisco, CA, USA, 2014, pp. 1-6.

[15] L. Katzir and I. Schwartzman, "Secure firmware updates for smart grid devices," in *Innovative Smart Grid Technologies (ISGT Europe)*, pp. 1-5., 2011.

[16] Y.-j. Kim et al., "A remote firmware upgrade method of nan and han devices to support amis energy services," *in International Conference on Hybrid Information Technology*, pp. 303-310, 2017

[17] S. Cheng et al., "Traffic-Aware Patching for Cyber Security in Mobile IoT," in *IEEE Commun. Magazine*, vol. 55, no. 7, pp. 29-35, July 2017.

[18] Q. D. Ngo, et al. "A survey of IoT malware and detection methods based on static features," in *ICT Express*, vol. 6, no. 4, pp.280-286, 2020.

[19] N. Guizani and A. Ghafoor, "A Network Function Virtualization System for Detecting Malware in Large IoT Based Networks," in *IEEE J. Sel. Areas Commun.*, vol. 38, no. 6, pp. 1218-1228, June 2020.

[20] M. Vojnovic and A. J. Ganesh, "On the race of worms, alerts, and patches," in *IEEE/ACM Trans. Netw.*, vol. 16, no. 5, pp. 1066-1079, October 2008.

[21] J. Simmins, "Remote meter firmware update," in *American Electric Power*, Use Case Document, 2011.

[22] S. Tonyal et al., "An Attribute & Network Coding-based Secure Multicast Protocol for Firmware Updates in Smart Grid AMI Networks," in *Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 97-102, 2018.

[23] A. Qasem et al., "Automatic vulnerability detection in embedded devices and firmware: survey and layered taxonomies," *ACM Computing Surveys,* vol. 54, no. 2, pp. 1-42, 2021.

[24] C. Konstantinou and M. Maniatakos, "Impact of firmware modification attacks on power systems field devices," *Proc. IEEE SmartGridComm,* pp. 283-288, Miami, FL, November 2015.

[25] A. Khattak, et al., "Smart meter security: Vulnerabilities, threat impacts, and countermeasures," *Proc. International Conference on Ubiquitous Information Management and Communication,* pp. 554-562, Phuket, Thailand, January 2019.

[26] P. Gope, et al., "QR-PUF: Design and Implementation of A RFID-based Secure Inpatient Management System Using XOR-Arbiter-PUF and QR-Code," *IEEE Transactions on Network Science and Engineering*, DOI:10.1109/TNSE.2022.3186478, 2022.