Optimizing Lightweight Intermittent Message Authentication for Programmable Logic Controller

Jiaqi Yang[1], Jun Xian Chia[2], Xijie Ba[1], Jianying Zhou[2], and Zheng Yang[1(✉)]

[1] Southwest University, Chongqing, China
{blsmcat,bxj32309713}@email.swu.edu.cn, youngzheng@swu.edu.cn
[2] Singapore University of Technology and Design, Singapore, Singapore
junxian_chia@mymail.sutd.edu.sg, jianying_zhou@sutd.edu.sg

Abstract. Programmable Logic Controllers (PLCs) are essential for Cyber-Physical Systems (CPS) but lack a software solution for lightweight message authentication to ensure the authenticity and integrity of data. A state-of-the-art lightweight signature, called LiS, is recently proposed for continuous message authentication in CPS, based on a chameleon hash function and a universal hash function (UHF) chain. Meanwhile, the signer and verifier should synchronize the randomness in the UHF chain for message authentication. To deal with the intermittent message authentication, Yang *et al.* proposed a lightweight authentication scheme LARP to replace the UHF chain with a UHF tree, in which each layer is divided by a series of time slots. The signer can quickly skip some randomness rather than linearly update the randomness as in LiS. However, the computational overhead required for synchronization might be expensive in LARP when the signer is suspended for a long time. Additionally, the overhead fluctuates greatly with the growth of interruption time. In this paper, we first propose an optimized UHF tree to reduce the overhead and mitigate its growth fluctuation. In particular, we implement our optimized scheme on an Allen Bradley ControlLogix 5571, leveraging fast modular reduction of pseudo-Mersenne prime to achieve efficient modulo operation. Our results demonstrate, for the first time, the feasibility and efficiency of running a lightweight signature scheme on the PLC.

Keywords: Intermittent Message Authentication · Digital Signature · Programmable Logic Controllers · Universal Hash Function

1 Introduction

Cyber-Physical Systems (CPS) are widely used across various domains, integrating advanced sensing, computing, and communication technologies with automatic control to manage physical processes. However, this also raises security and privacy issues [11,21]. CPS has become a target for numerous attacks, such as the

© ICST Institute for Computer Sciences, Social Informatics and Telecommunications Engineering 2025
Published by Springer Nature Switzerland AG 2025. All Rights Reserved
H. Duan et al. (Eds.): SecureComm 2023, LNICST 567, pp. 225–243, 2025.
https://doi.org/10.1007/978-3-031-64948-6_12

Stuxnet attack against an Iranian uranium enrichment plant in late 2009 [16]. In this attack, the attacker targeted commercial programmable logic controllers (PLCs) and carried out deception and disclosure attacks, resulting in significant damage to a large number of centrifuges. Despite the urgent need for security and privacy in CPS, many cryptographic schemes are not feasible due to the limitations of resource-constrained devices commonly found in CPS.

In particular, when it comes to PLCs, which serve as the core for CPS, there is a lack of implementations of lightweight cryptography. While existing work such as AV and Duka [10] have implemented TinyJAMBU [23], one of the finalists in the NIST competition for lightweight cryptography [6], and Yang *et al.* [24] proposed and implemented PLCrypto, a lightweight symmetric cryptographic library, no implementations of asymmetric cryptography on PLCs have been reported. However, a lightweight signature scheme is critical for CPS. In CPS, sensor data is sent to the PLC, which then forwards it to the Supervisory Control and Data Acquisition (SCADA) system. As such, it is critical to ensure the authenticity and integrity of the data originating from the PLC.

To fill the gap in the lightweight message authentication scheme on PLC, we optimized the lightweight intermittent message authentication proposed by Yang *et al.* and evaluated it on PLC. In their work [26], they proposed a state-of-the-art lightweight signature scheme, named LiS, suitable for continuous message authentication for CPS. The scheme utilizes the chameleon hash function (CHF) [15], reducing the cost for the signer. Its ability to enable continuous message authentication relies on constantly updated randomness r, generated with a universal hash function chain, i.e., $r = \mathsf{UHF}(k, r)$, where k is key of UHF. Using r, the verify points vr are computed by CHF and all verify points are compressed and stored in bloom filters as verification keys. However, the UHF chain poses a challenge to implementing intermittent message authentication. Note that signing and verification can only be successful if the signer and verifier have the same r. Using the UHF chain results in a significant number of UHF operations that the signer must execute these operations to synchronize with the verifier's r at the beginning of the next run, which is not feasible for resource-constrained devices like PLC. To address this limitation, in LARP [25], they replace the UHF chain with a three-layered UHF tree with a different time slot of each layer, significantly reducing the computational overhead required to synchronize r, thus supporting intermittent message authentication.

In this work, we analyze the problems with their scheme in the context of PLC characteristics. First, in light of the Tag Manipulation Attack (TMA) [24], the PLC confronts a challenge in guaranteeing the integrity of variables between scan cycles. Consequently, synchronization needs to commence from the initial hard-coded randomness, unlike LARP where synchronization is based on the most recent randomness. Therefore, the synchronization overhead of the PLC is determined by the signing time rather than the interruption time interval, unlike in LARP. Second, in LARP, if the signer interrupts for too long, the computational overhead required for synchronization may become unaffordable, even with the reduced number of UHF operations provided by the UHF tree. Likewise, on the PLC, if the signing time is far from the initial time, the computational overhead

will be very expensive. Note that we only consider the case on the PLC in the sequel. To address this issue, we propose to add a cache layer on top of the UHF tree to constrain the number of UHF operations required for synchronization. Specifically, we cache the first randomness of the first layer in the UHF tree every 24 h. For example, once the signing time is longer than 24 h, the first 24 h of UHF operation can be avoided when synchronizing. Third, we observe that the growth of computational overhead exhibits significant fluctuations as the interruption time increases, a problem also noted in [25]. We attribute this to the way the UHF tree is divided based on time and the number of UHF operations varies greatly between the different layers, especially in the last layer. To minimize both the computational overhead and the magnitude of fluctuations (In the sequel, it is also referred to as fluctuations), we optimize the UHF tree in two parts, delimited by the verification key layer (5 min). We assume that the ratio of time slots of adjacent layers in the UHF tree is the same, dividing the time slots uniformly in this way yields the smallest fluctuations. Therefore, optimizing the number of layers is the key to optimizing the UHF tree. As the number of layers increases, on the one hand, the number of UHF operations per layer will be less and the fluctuation will be smaller, but on the other hand, the number of UHF operations required to switch branches will be larger and the overhead will be higher. Therefore, we also build the cost models to determine the optimal number of layers and the ratio to minimize the computational overhead and its fluctuations. We finally build an eleven-layered UHF tree, which in the worst case only needs to execute 32 UHF operations and has smaller fluctuations in computational overhead.

We also implement the optimized scheme based on the symmetric cryptographic library PLCrypto [24] on a commercial PLC but encountered two challenges in the implementation. First, the efficiency of the existing modulo operation for big-integer is low [24]. To overcome this challenge, we leverage the properties of pseudo-Mersenne primes to enable fast modular reduction on PLC. Second, our implementations must prevent TMA. We employ various implementation strategies, such as hard coding [24]. As far as we know, we are the first to implement a lightweight signature scheme on a PLC.

Contribution. In this paper, we make the following significant contributions:

- We add a cache layer to the UHF tree to cache the randomness r in the first layer of the UHF tree every 24 h to prevent the computational overhead from growing indefinitely.
- We propose an optimized UHF tree by dividing the time slot of layers in almost the same ratio and constructing cost models for the computational overhead concerning the number of layers. This reduces the computational overhead required for synchronization and ensures a smoother growth of overhead.
- We evaluate our optimized scheme on a commercial Allen Bradley PLC. We also implement the fast modular reduction algorithm which is admitted by the pseudo-Mersenne prime, allowing it to perform public key cryptography and compensating for the shortcomings of PLCrypto.

Organization. The paper is organized as follows. Section 2 and Sect. 3 provide an introduction to the preliminaries and background of PLC. In Sect. 4, we describe the threat model. We review LiS and the UHF tree proposed in LARP in Sect. 5. Section 6 presents our optimization techniques, and Sect. 7 provides our key implementation on PLC. We discuss the performance of our optimized scheme on PLC in Sect. 8. Finally, we conclude the paper in Sect. 9.

2 Preliminaries

In this section, we introduce the notations and preliminaries that may utilize. For more details about the security definitions, please refer to [14].

Notations. We set κ as the security parameter, and use $[n] = 1, \ldots, n \subset \mathbb{N}$ to denote a set of integers ranging from 1 to n. We use $x \xleftarrow{\$} S$ to denote the action of sampling a uniformly random element selected from a set S. Additionally, we use the symbol $\|$ to represent the operation of concatenating two strings and $|\cdot|$ to denote the bit-length of a variable.

Table 1 presents the list of notations that we use. To facilitate the understanding, some notations referenced from LiS [26] and LARP [25].

Table 1. The Description of Notations

Notation	Description
$\mathsf{sk}_{\mathsf{PLC}}, \mathsf{pk}_{\mathsf{PLC}}$	The private key and public key of PLC
$\mathsf{hk}i$	The universal hash function keys of the i-th layer and each of them composes of two sub-keys that is $\mathsf{hk}i = (\mathsf{hk}i_0, \mathsf{hk}i_1)$
r_i	The constantly refreshing secret parameter of the i-th UHF layer
M	A secret parameter for chameleon hash function
T_t	The lifetime of the scheme
T_i	The time slot of the i-th layer of UHF tree
T_c	The current time of signer running signature generation. We assume that the start time is 0

Universal Hash Function (UHF) family [9] is a family of hash functions with a low number of collisions, denoted as $\mathsf{UH} : \mathcal{K} \times \mathsf{A} \rightarrow \mathsf{O}$. Here, $\mathcal{K}$, A, and O are the key, message, and output spaces of UH, respectively, and they are determined by the security parameter κ. To instantiate UHF, we choose a large prime number q and set $\mathcal{K} = \mathsf{A} = \mathsf{O} = \mathsf{Z}_q^*$. Given a message m, the hash value x is generated through $\mathsf{x} = \mathsf{UHF}(\mathsf{hk}, m) = \mathsf{hk}_0 \cdot m + \mathsf{hk}_1 \pmod q$ where $\mathsf{hk} = (\mathsf{hk}_0, \mathsf{hk}_1)$ and they are randomly chosen from Z_q^*.

Chameleon Hash Function (CHF)**.** [15] A CHF consists of three algorithms.

- $\mathsf{CHKGen}(1^\kappa)$: This algorithm samples a secret key $\mathsf{sk_{CH}} \xleftarrow{\$} \mathsf{Z}_q^*$ where q is a large prime and computes the public key $\mathsf{pk_{CH}} = g^{\mathsf{sk_{CH}}} \pmod p$ where $p = u \cdot q + 1$, u is a small integer and g is a random group generator of order q in Z_q^*.
- $\mathsf{CHF}(\mathsf{pk_{CH}}, m, r)$: This algorithm outputs a hash value $y = g^r \cdot (\mathsf{pk_{CH}})^m \pmod p$, where $\mathsf{pk_{CH}}$ is the public key, $m \in \mathsf{Z}_q^*$ is a message, and $r \in \mathsf{Z}_q^*$ is a randomness.
- $\mathsf{CHColl}(\mathsf{sk_{CH}}, r, \mathsf{M}, m)$: This algorithm outputs collision $r' = \mathsf{M} \cdot \mathsf{sk_{CH}} + r - m \cdot \mathsf{sk_{CH}}$ such that $\mathsf{CHF}(\mathsf{pk_{CH}}, \mathsf{M}, r) = \mathsf{CHF}(\mathsf{pk_{CH}}, m, r')$ where $(r, \mathsf{M}, m) \in \mathsf{Z}_q^*$.

Digital Signature Schemes. We define a digital signature scheme Sig, which is based on [12], consists of three PPT algorithms such that:

- $\mathsf{Gen}(1^\kappa)$: This is the key generation algorithm that generates a secret key sk and a public key pk, with 1^κ as the input security parameter.
- $\mathsf{Sign}(sk, m)$: This is the signing algorithm that outputs a signature s using the secret key sk and a message $m \in M$, where M denotes the message space.
- $\mathsf{Verify}(pk, m, \sigma)$: This is the verification algorithm that outputs 1 if the input signature s is valid, and 0 otherwise.

Bloom Filter (BF). [8,19] is a probabilistic data structure that provides space efficient storage of a set and that can efficiently test whether an element m is a member of the set S. A BF consists of three algorithms as follows.

- Init: This algorithm initiates the BF of bit length $1.44\epsilon\mathsf{N}$, where N denote the size of BF. And it has a false positive rate (FPR) of $2^{-\epsilon}$ [20].
- Insert: This algorithm inserts the element m into BF
- Check: This algorithm $\mathsf{Check}(m)$ returns 1 if the element m is in BF, and 0 otherwise.

Pseudo-Mersenne Prime. If a prime q is of the form $2^m - c$, where $c \ll 2^m$, then it is referred to as a pseudo-Mersenne prime [13]. These pseudo-Mersenne primes have a fast modular reduction similar to Mersenne primes, making them useful in public key cryptography. For instance, the Curve25519 elliptic curve cryptography scheme uses the pseudo-Mersenne prime $2^{255} - 19$ [7], and the Bitcoin protocol employs $2^{256} - 2^{23} - 977$ [18].

To overcome the low efficiency of modulo operation on PLC [24], we chose pseudo-Mersenne primes and implemented the fast modular reduction. This approach allowed us to implement public key cryptography on PLC for the first time while ensuring high efficiency.

3 The Background of PLC

Programmable Logic Controllers (PLCs) are embedded devices that are dedicated to controlling industrial equipment such as actuators. They are highly

reliable and consist of several different modules, including serial or Ethernet communication and discrete/analog I/O modules.

Scan Cycle. In a PLC, all control logic (i.e., control program) is executed periodically. This periodic process is called a scan cycle. During a scan cycle, sensor data is transmitted to the PLC through the I/O and communication modules. After the control logic in the PLC is executed, the output signal is sent to the actuator, and the data is sent to SCADA.

Structured Text (ST). The standard IEC-61131-3 [22] defines several PLC programming languages, including Structured Text (ST), Ladder Logic (LL), and Function Block Diagrams (FBD). Among these, ST is considered more suitable for data processing and is more in line with the syntax and programming habits of high-level languages such as C. Therefore, we chose ST as the programming language for our implementation on an Allen-Bradley PLC. However, due to the characteristics of the ST version of Allen-Bradley, some functions are not provided [5]. For example, it does not support shift/rotate, which is essential for cryptographic algorithms, leading to difficulties in our implementation. On the other hand, ST provides bit accessibility, allowing direct access to every bit of a tag (i.e., a variable), which reduces the overhead of certain special operations. For example, the shift can be implemented by bit access.

User-Defined Data Type (UDT) [4]**.** UDT is a custom data type that simplifies the code by allowing programmers to reference the UDT instead of multiple individual data elements. UDT can also improve code readability and maintainability by making it more organized and easier to understand. They can be exported for other programmers to use and modify. In our implementation, we define two UDTs to represent big-integers and pseudo-Mersenne primes.

Add-On Instruction (AOI) [3]**.** AOI is a reusable instruction that encapsulates a specific function or algorithm. It can be used to implement complex algorithms or group-related instructions, making the program more modular and easier to maintain. AOI can also be exported for use and modification. In our implementation, we use AOIs to execute all big-integer operations.

4 Threat Model

In this section, we describe the authentication threat to CPS and an attack against PLC known as the Tag Manipulation Attack (TMA).

Forgery. In Fig. 1, we present the system model and threat model. In this system model, the plant manages all equipment as a fully trusted authority. In addition to sending control signals to the actuators, the PLC also transmits sensor data to the SCADA, which is responsible for collecting operational data from all the PLCs to adjust control parameters.

We assume that attackers can gain control of communications at the plant. The threats come from network attackers attempting to send forged and fake data to the SCADA system. Specifically, attackers may eavesdrop and intercept

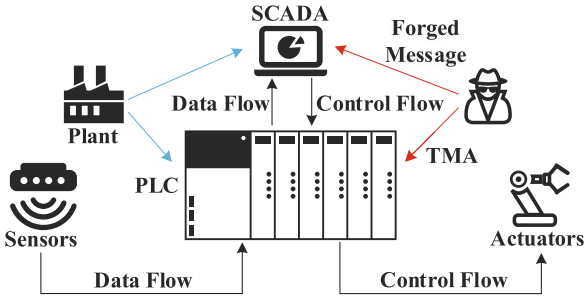


Fig. 1. System Model of CPS

communication to maliciously modify or simulate messages sent by legitimate PLCs to the SCADA system.

Tag Manipulation Attack (TMA). In [24], Yang *et al.* described a potential attack against PLCs known as the TMA. The attack assumes that a network attacker can communicate with the PLC to download/upload control logics and read/write tags using the basic supervisory function provided by the PLC [2]. This can result in the leakage of secrets or modification of parameters. Therefore, to prevent TMA, we adopt the strategy proposed in [24] in our implementation.

Note that due to TMA, the PLC cannot ensure the integrity of variables between scan cycles, making it impossible to retain the randomness used in the last signature generation and the time of the last signature. Consequently, when synchronizing the randomness, the PLC must begin with the initial randomness.

5 Review

In LARP, to avoid a single point of failure due to the learning of the signing key by the semi-honest authorization authority (AA), the randomness used for verify points is generated by computing the secret exponentiation between the AA and enrolment authority (EA). Specifically, LARP splits the first sub-key hk_0 of UHF into two secret shares ($\mathsf{hk}_0 \cdot \alpha$ and $1/\alpha$), and distribute them to the AA and EA respectively, where α is a random value. Thus, When AA computes a verify point, it first sends g^r to EA which will compute a response $\mathsf{U} = g^{r \cdot 1/\alpha \pmod q}$ for AA. Then AA can refresh g^r as $g^r = (\mathsf{U})^{\mathsf{hk}_0 \cdot \alpha} \cdot g^{\mathsf{hk3}_1}$ which is used to compute a verify point. Therefore, for compatibility with LARP, we retain modular arithmetic in the computation of UHF. Additionally, since LARP is based on LiS, we will start our review with LiS.

In LiS [26],[1] the verify points $vp = \mathsf{CHF}(\mathsf{M}, r)$ are pre-generated by the key generation center (KGC) and compressed into Bloom filters which are used as

[1] In the sequel, we mainly focus on demonstrating our optimization ideas based on the first construction of LiS in [26] which is relatively simple. Nevertheless, our new technique can be applied to both versions of LiS. And our optimized scheme is also compatible with the operation of the server and verifier in LARP.

verification keys. To obtain a signature s for a message m, the signer computes $\mathsf{s} = \mathsf{CHColl}(\mathsf{sk}_{\mathsf{CH}}, r, \mathsf{M}, m)$[2] where M is a dummy message and $\mathsf{sk}_{\mathsf{CH}}$ is the private key of the CHF. During verification, the verifier computes $vr = \mathsf{CHF}(m, \mathsf{s})$ and accepts the signature by checking whether vr is in the Bloom filter. We can observe that the signer and verifier must have the same randomness r for a successful signature. However, in LiS, r is generated by the UHF chain, which means that if the signer stops running, it will need to execute many UHF operations to synchronize r with the verifier upon its next restart. This results in LiS not being able to implement intermittent message authentication.

To solve this problem, LARP [25] proposed the three-layered UHF tree instead of the UHF chain. Here, we quote the structure diagram in LARP as shown in Fig. 2. In the three-layered UHF tree, each layer is divided into different time slots with distinct lengths: $\mathsf{T}_1 = 1$ h, $\mathsf{T}_2 = 5$ min, and $\mathsf{T}_3 = 1$ s. Note that LARP assumes that 5 min in the second layer denotes the validity period of a verification key (i.e., a Bloom filter is instantiated every 5 min), and 1 s in the third layer denotes the maximum runtime for generating a signature. Similar to LARP, our optimized scheme also considers a verification key's validity period of 5 min and a maximum runtime of 1 s for generating a signature.

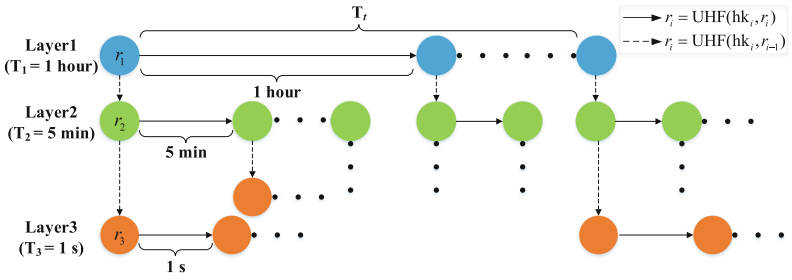


Fig. 2. Three-layered UHF Tree Diagram

With the UHF tree, LARP achieves intermittent message authentication. It only needs to execute a few UHF operations to synchronize r with the verifier. For example, if the signer stops for 2 h 5 min 5 s, it only needs to execute 10 UHF operations (2 UHF operations in the first layer, 1 UHF operation in the second layer, 5 UHF operations in the third layer, and 2 UHF operations for switching branches).

Existing Problems. We notice that for PLCs, there are two problems with the three-layered UHF tree for randomness synchronization. First, When the time gap between the signing time and the initial time is substantial (e.g., several weeks), the signer may not be able to afford the high computational overhead.

[2] Note that, in LiS_2 [26], the message m is hashed together with a random value R, where R is chosen at random from a randomness space $\mathcal{R}$. That is, $m' = \mathsf{H}(m||\mathsf{R})$, Where $m \in \{0,1\}^*$ and $\mathsf{R} \in \mathcal{R}$.

Second, the computational overhead exhibits significant fluctuations in growth. In other words, there exists a significant disparity under different T_c. If T_c is not a multiple of 5 min, a significant number of UHF operations are required in the third layer, whereas if it is a multiple of 1 h, only a few UHF operations are required in the first layer. Specifically, a maximum of 299 ($\frac{5\,\text{min}}{1\text{s}} - 1$) UHF operations are required in the third layer, compared to a maximum of 11 ($\frac{1\text{hr}}{5\,\text{min}} - 1$) operations in the second layer and 23 ($\frac{24\text{hr}}{1\text{hr}} - 1$) operations in the first layer.

6 Optimized Scheme

In this section, we present several optimizations aimed at solving existing computational overhead problems.

Key Idea. To address the first problem, our main idea is to cache the randomness when T_c reaches a certain time. This eliminates an amount of UHF operations. For the second problem, as mentioned above, we find a very large gap in the number of all UHF operations per layer. As the layers of the UHF tree are divided based on time, this problem is inevitable and can only be mitigated as much as possible. We notice that there is a significant difference in the number of UHF operations between adjacent layers (which is determined by the ratio of time slots), which leads to substantial fluctuations. Therefore, if the ratio of time slots of layers (i.e., $\mathrm{T}_i/\mathrm{T}_{i-1}$) differs little or even the same the fluctuations will be smoother. Additionally, the number of layers in the UHF tree affects the number of UHF operations. To minimize the computational overhead and its fluctuations, we construct the cost models concerning the number of layers. Considering that a verification key remains valid for 5 min, we utilize the layer with a time slot of 5 min as a boundary to partition the UHF tree into upper and lower parts for optimization.

6.1 Optimization

Synchronization when $\mathsf{T}_c \geq 24\mathsf{hr}$. As shown in Fig. 3, we add a cache layer on top of the UHF tree as layer 0. The time slot of the layer is set to 24 h, which means that the first randomness r_1 in the first layer can be obtained directly from the cache every 24 h. For an instance with a lifetime $T_t = 1$ year, only about 10KB of storage is required (for 224-bit randomness).

In this way, when $\mathsf{T}_c \geq 24\mathsf{hr}$, it can be bounded to less than 24 h by using the latest r_1 obtained from the cache, thus eliminating a large number of UHF operations. For instance, suppose $\mathsf{T}_c = 30 \cdot 24\mathsf{hr}$, 30 UHF operations were required before optimization to get the latest r_1, and just from the cache after optimization.

Optimization of UHF Tree. In the previous optimization, we introduced a cache layer to prevent unbounded growth of computational overhead. However, the substantial number of UHF operations in each layer (particularly the last

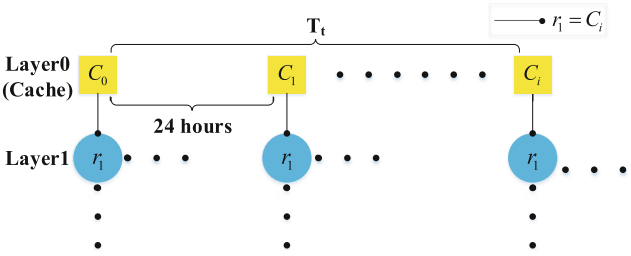


Fig. 3. UHF Tree with a Cache Layer

layer with 299 UHF operations), combined with the considerable variation in the number of operations between layers, results in significant computational overhead and its corresponding fluctuations. The number of UHF operations in each layer depends on the ratio of time slots of the adjacent layer. Thus, when the ratios of the time slots of all adjacent layers are nearly identical and small, the fluctuations are smaller.

Additionally, for compatibility with LARP, we keep the 5-minute validity of the verification key unchanged. Therefore, we partition the UHF tree into two parts, delineated by the layer with a 5-minute time slot (corresponding to the second layer in LARP). Specifically, the first part ranges from 5 min to 24 h (including 5 min, with n denoting the number of layers). The second part ranges from 1 s to 5 min (excluding 5 min, with m denoting the number of layers).

For the first part, we assume that the ratio of time slots between adjacent layers is the same. Thus each layer has the same number of UHF operations (i.e., the ratio minus 1). However, achieving smoother growth in computational overhead is not that simple. It is related to the number of layers n. If n is larger, the ratio is smaller and the number of UHF operations per layer will also be smaller, resulting in smoother growth of computational overhead. However, a larger n also leads to a higher number of UHF operations for switching branches, resulting in higher computational overhead. Hence, we construct a cost model regarding the number of layers n.

First, we calculate the ratio as $\left(\frac{86400}{300}\right)^{\frac{1}{n}} = 288^{\frac{1}{n}}$, where 86400 denotes 24 h and 300 denotes 5 min. Second, n layers will require at most $n-1$ UHF operations for switching branches.

Then we can express the total number of UHF operations required in the worst case as the following function:

$$\begin{aligned} f(n) &= (288^{\frac{1}{n}} - 1) \cdot n + (n-1) \\ &= n \cdot 288^{\frac{1}{n}} - 1 \end{aligned} \tag{1}$$

where the first term is the number of UHF operations needed in the layers, and the second term is the number of UHF operations needed for switching branches.

To determine the minimum cost we need to find the minimum value of the function $f(n)$. This can be accomplished by taking the derivative of the func-

tion and setting it equal to zero, which yields the value of n. After taking the derivative of $f(n)$, we obtain the expression for $f'(n)$ given by:

$$f'(n) = 288^{\frac{1}{n}} \cdot (1 - \frac{\ln^{288}}{n}) \tag{2}$$

We set $f'(n)$ to 0, resulting in $n = 5.663$. Since n must be an integer, we compare the values of $f(5)$ and $f(6)$ using equation (1). We find that $f(6) < f(5)$. Therefore, we chose $n = 6$, which yields a time slot ratio of 2 or 3 between adjacent layers, as indicated by $288^{\frac{1}{6}} = 2.57$. Since the UHF operations for each layer must be integers, we set the ratio of time slot ratio between the first layer and the 24 h to 4 to make the ratio of each time slot an integer.

As shown in Fig. 4, the first part consists of six layers, with time slots allocated as follows: $\mathsf{T}_1 = 6$ h, $\mathsf{T}_2 = 2$ h, $\mathsf{T}_3 = 40$ min, $\mathsf{T}_4 = 20$ min, $\mathsf{T}_5 = 10$ min, and $\mathsf{T}_6 = 5$ min. The average ratio of time slot is 2.67 which is close to $288^{\frac{1}{6}} = 2.57$. Additionally, the first randomness r_1 of the first layer every 24 h is obtained from the cache.

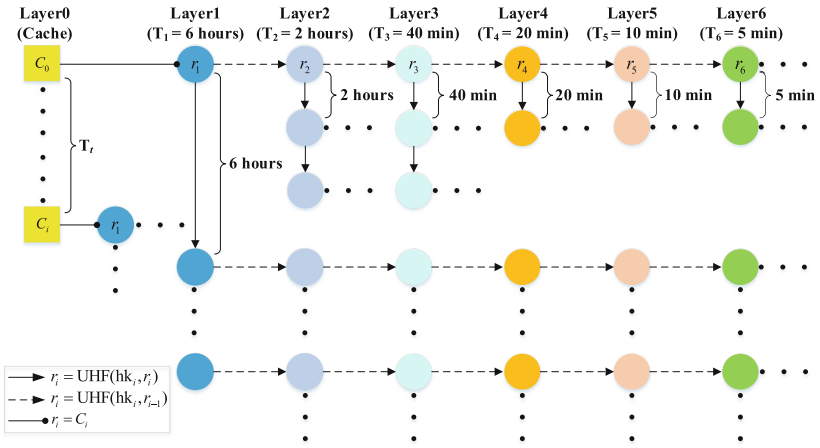


Fig. 4. The First Part of the UHF Tree Diagram

For the second part, we apply the same optimization approach as we did for the first part. First, we formulate the UHF operations using a function of the number of layers m:

$$\begin{aligned} f(m) &= (300^{\frac{1}{m}} - 1) \cdot m + m \\ &= m \cdot 300^{\frac{1}{m}} \end{aligned} \tag{3}$$

The only alteration from the function in the first part is that the number of switching branches in the second part is m instead of $m - 1$ (as the 5-minute layer is absent in the second part). After taking the derivative of $f(m)$, we obtain

the expression for $f'(m)$ given by:

$$f'(m) = 300^{\frac{1}{m}} \cdot (1 - \frac{\ln^{300}}{m}) \tag{4}$$

We set $f'(m)$ to 0, resulting in $m = 5.704$. Even though $f(6) < f(5)$, we opt for $m = 5$ to prevent the layer division of the UHF tree from leading to a non-integer number of UHF operations when $m = 6$. Similarly, to ensure divisibility by 300 (5 min), we set the ratio of time slots between the fourth and fifth layers, and between the fifth and sixth layers, to be 5.

As shown in Fig. 5, the second part consists of five layers, with time slots allocated as follows: $\mathsf{T}_7 = 1$ min, $\mathsf{T}_8 = 12$ s, $\mathsf{T}_9 = 4$ s, $\mathsf{T}_{10} = 2$ s, and $\mathsf{T}_{11} = 1s$. The average ratio of time slot is 3.4 which is close to $300^{\frac{1}{5}} = 3.12$.

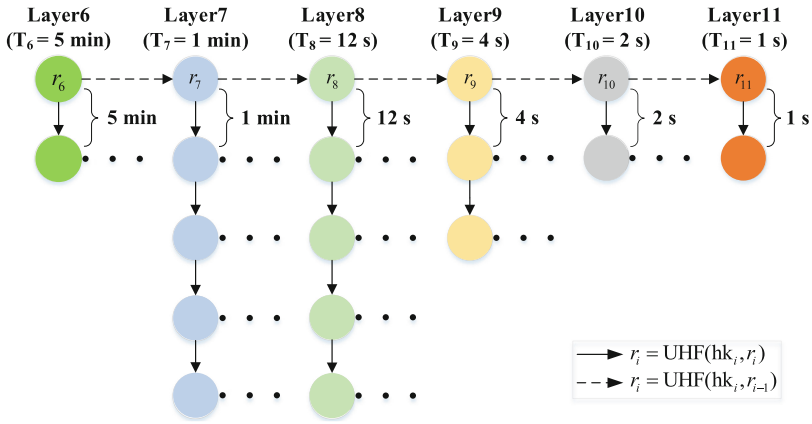


Fig. 5. The Second Part of the UHF Tree Diagram

Finally, by combining the optimization for both parts, we construct an optimized eleven-layered UHF tree. Since the number of UHF operations in each layer is small and the ratios almost equal, the computational overhead grows more smoothly as T_c grows. Our UHF tree also has smaller computational overhead, the worst case (T_c = 23hr 59min 59 s) requires only 32 UHF operations. In comparison, the three-layered UHF tree in LARP requires 335 operations.

6.2 Optimized Scheme

As shown in Fig. 6, we assume that the plant is fully trusted and acts as the key generation center (KGC). The PLC acts as the signer and the SCADA acts as the verifier. In the following, we will elaborate optimized lightweight intermittent message authentication scheme.

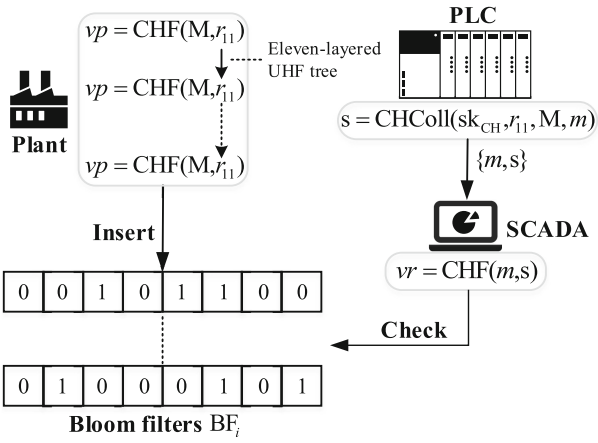


Fig. 6. Overview of Optimized Scheme

- *Initialization*: First, the plant runs a key generation function of CHF to generate a pair of secret/public keys: $(\mathsf{sk_{CH}}, \mathsf{pk_{CH}}) \leftarrow \mathsf{CHKGen}(1^\kappa)$. Next, a random message $\mathsf{M} \xleftarrow{\$} \mathsf{Z}_q^*$ and all the random keys $\mathsf{hk} \xleftarrow{\$} \mathcal{K}$ are sampled. Then, a set of bloom filter instances BF_i are initialized, where a BF is generated every 5 min. The verification points $vp = \mathsf{CHF}(\mathsf{pk_{CH}}, \mathsf{M}, r_{11})$ are inserted into the Bloom filters: $\mathsf{BF}_i.\mathsf{Insert}(vp)$. The randomness r_{11} is updated with the eleven-layered UHF tree. Finally, the secret key $sk_{\mathsf{PLC}} = (\mathsf{sk_{CH}}, \mathsf{M})$ and the public verification key $pk_{\mathsf{PLC}} = (\mathsf{BF}_i, \mathsf{pk_{CH}})$ are returned.
- *Synchronization*: Once the PLC runs signature generation, it needs to call Algorithm 1 to synchronize the randomness r_{11} with the verifier.
- *Signing*: PLC generates the signature $\mathsf{s} = \mathsf{CHColl}(\mathsf{sk_{CH}}, r_{11}, \mathsf{M}, m)$ for message m and sends the tuple (m, s) to the verifier.
- *Verification*: To verify (m, s), the SCADA checks that whether the $vr = \mathsf{CHF}(\mathsf{pk_{CH}}, m, \mathsf{s})$ is in the current Bloom filter.

6.3 Security Analysis

The security of our optimized scheme relies on the cryptographic primitives and LiS which has been proven to be secure. In this subsection, we provide an informal analysis of how our optimized scheme can resist the threat described in Sect. 4. Interested readers can refer to the formal proofs presented in [26]. Moreover, we describe the tricks we use against TMA which are proposed in [24].

Resistance to Forgery. Since the attacker does not know the private key $\mathsf{sk_{CH}}$ and it cannot get the randomness from the intercepted data, he cannot forge the signature or tamper with the message. If a network attacker forges the data (m, s) and sends it to the SCADA, the SCADA will compute $vr = \mathsf{CHF}(\mathsf{pk_{CH}}, m, \mathsf{s})$ and

Algorithm 1. Synchronization

Input: r_i $(1 \leq i \leq n)$ where n is the number of layer, T_c
Output: Updated r_i $(1 \leq i \leq \mathsf{n})$

if $\mathsf{T}_c \geq 24\mathsf{hr}$ **then**
  $i := \left\lfloor \frac{\mathsf{T}_c}{24\mathsf{hr}} \right\rfloor$
  $r_1 :=$ Cache$[i]$ // First r_1 of every 24 hours
  for $i := 2$ to n **do**
    $r_i := \mathsf{hk}i_0 \cdot r_{i-1} + \mathsf{hk}i_1 \pmod q$
  $\mathsf{T}_c := \mathsf{T}_c - i \cdot 24\mathsf{hr}$
tmp $:= 0$
for $i := 1$ to $n - 1$ **do**
  $\mathsf{T}_c := \mathsf{T}_c -$ tmp
  $\mathsf{n}_i := \left\lfloor \frac{\mathsf{T}_c}{\mathsf{T}_i} \right\rfloor$ // The number of UHF operations in each layer
  tmp $:= \mathsf{n}_i \cdot \mathsf{T}_i$
Flag := **false** // If Flag is true, next layer should be update
for $i := 1$ to n **do**
  if Flag = **true then**
    $r_i := \mathsf{hk}i_0 \cdot r_{i-1} + \mathsf{hk}i_1 \pmod q$ // Switch to the next layer
  for $j := 1$ to n_i **do**
    $r_i := \mathsf{hk}i_0 \cdot r_i + \mathsf{hk}i_1 \pmod q$ // Update r_i in the i-th UHF layer
  if $\mathsf{n}_i \neq 0$ **then**
    Flag := **true**

discover that vr is not in the Bloom filter. Therefore, our optimized scheme can against authentication threat.

Tricks Against TMA. To resist TMA, we use some tricks from [24] in our implementation. First, to prevent attackers from downloading/uploading control logic, we use the physical switch to switch the PLC mode to run mode, which can only be operated by authorized PLC administrators. Additionally, we set the task to be periodical with a higher priority than the communication task, to prevent interruptions by the communication task and avoid attacks within a scan cycle. Second, to protect the confidentiality of secret constants between scan cycles, we use the Hard-coding with Runtime Loading (HC-RL) strategy. Specifically, we dynamically load the concrete values of these tags of secret constants at the beginning of a scan cycle, and then erase the tags at the end. This ensures that the secret constants are only in memory during the time they are required.

7 Implementation

Platform. Our implementation involves synchronizing and signing on the Allen Bradley ControlLogix 5571, while verification is carried out on a desktop PC with an Intel(R) Core(TM) i7-10700 CPU 2.90 GHz.

Libraries. Our implementation on PLC is based on PLCrypto [24], with the incorporation of a modular reduction of pseudo-Mersenne prime. We also re-implement the big-integer operations as Add-On instructions to facilitate calls. For verification on the PC, we employ the cryptographic library Miracl [1].

User-Defined Data Type. Since the data type in ST does not support big-integers, we introduce a UDT to represent positive big-integers. It consists of 256-dimensional unsigned 32-bit integers, where only 15 bits are used per dimension to support the multiplication. Therefore, a big-integer can represent a maximum of 3940 bits and supports up to two 1920-bit positive big-integer multiplications. Additionally, we define another UDT for the pseudo-Mersenne prime, which comprises three big integers val, m, and c, where $val = 2^m - c$.

Add-On Instructions. In PLCrypto, big-integer operations are implemented in separate routines, which can make the code that requires a large number of big-integer operations complex and difficult to maintain. This is because every big-integer operation must be re-implemented. To address this issue, we re-implemented all big-integer operations as AOIs, including PLCrypto's original code and modular reduction of pseudo-Mersenne prime.

Modular Reduction. We refer to the modular reduction described in [17] as shown in Algorithm 2. This modular reduction requires only a small number of shifts, additions, and multiplications, rather than costly big-integer division. Although ST does not have built-in bit-wise shift instructions, shifts can be quickly implemented due to supporting bit accessibility. Moreover, mod 2^n can be obtained by setting the high bits to 0, which further simplifies the computation.

Algorithm 2. Modular Reduction

Input: a positive integer x, a pseudo-Mersenne modulus $q = 2^n - c$
Output: $r = x \mod m$

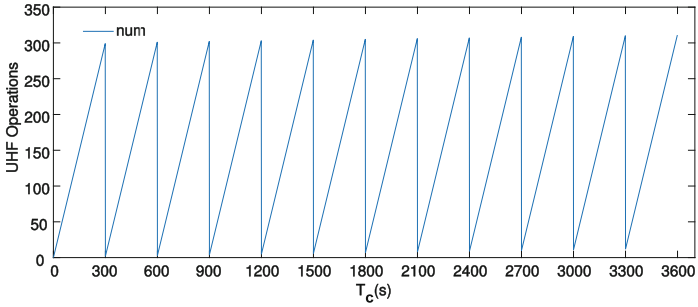
```
1: while x ≥ 2^n do
2:     low := x mod 2^n // Take the lower n bits of x
3:     high := x >> n // Take the higher bits of x
4:     x := low + high · c
5: if x < q then
6:     r := x
7: else
8:     r := x − q
```

8 Performance Evaluation

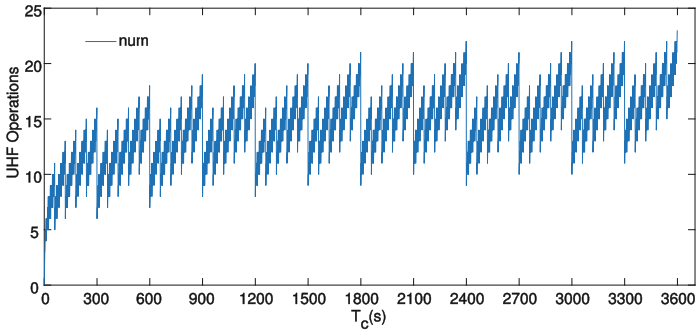
In this section, we present a computational overhead evaluation of our optimized scheme under three parameters (i.e., $|q|$ =224, 256, and 384 bits) which are pseudo-Mersenne primes. We use the optimized scheme to denote our optimization and the original scheme to denote the scheme of Yang *et al.* Note

that we are the first to implement a lightweight signature scheme on a PLC. Consequently, our evaluation focused solely on our scheme without comparative analysis against other schemes.

Comparison. Figure 7 presents a comparison of the computational overhead between our optimized scheme and the original scheme. We use the number of UHF operations to denote the computational overhead. Note that, due to the high density of the line, we set $\mathsf{T}_c < 1\mathsf{hr}$. As shown in Fig. 7 (a), the UHF operations of the original scheme exhibit significant fluctuations, ranging from approximately 0 to 300 times. Conversely, in Fig. 7 (b), we present our optimized scheme, in which the fluctuation amplitude is confined to approximately 0 to 25 UHF operations. Consequently, our optimized scheme effectively reduces the extent of fluctuation, even though there is an inevitable rise in frequency due to the increase in the number of layers. Additionally, for $\mathsf{T}_c \geq 24\mathsf{hr}$, our optimized scheme simply obtains randomness from the cache layer instead of continuing to grow with fluctuations, as the original scheme does.



(a) Original Scheme



(b) Optimized Scheme

Fig. 7. The Number of UHF Operations.

Runtimes of Synchronization. In Table 2, we present the cost of synchronization (i.e., Algorithm 1) for different values of T_c. We observe that for $\mathsf{T}_c < 24\mathsf{hr}$, synchronization can be achieved around 500 ms, except for $|q| = 384$ bits, which is close to 1 s in the worst case.

Table 2. Runtimes of Synchronization

T_c	$\|q\| = 224$ bits	$\|q\| = 256$ bits	$\|q\| = 384$ bits
5 min	70.432 ms	95.823 ms	172.740 ms
36 min	123.594 ms	175.770 ms	316.498 ms
1 hr 57min	170.870 ms	239.372 ms	439.427 ms
2 hr	117.348 ms	163.719 ms	297.630 ms
23 hr 59min 59 s	362.752 ms	508.530 ms	931.431 ms

Runtimes of Signature and UHF. In Table 3, we show the cost of signature and UHF operation. Notably, the signature is within 10 ms for $|q| = 224$ bits. Furthermore, we observe that even with the smallest parameter we chose (i.e., $|q| = 224$ bits), one UHF operation takes about 11 ms. This suggests that the original scheme is not suitable for PLCs, as a single synchronization may take several seconds to complete.

Table 3. Runtimes of Signature, Verification, and UHF

Scheme	$\|q\| = 224$ bits	$\|q\| = 256$ bits	$\|q\| = 384$ bits
UHF	11.052 ms	15.608 ms	28.772 ms
Signature	9.951 ms	12.393 ms	23.880 ms

Runtimes of Modular Reduction. In Table 4, we also conduct an evaluation of the modular reduction under three parameters q which are pseudo-Mersenne primes. Our results demonstrate that the fast modular reduction designed for pseudo-Mersenne primes achieves excellent performance on PLCs. This suggests the possibility of implementing public key cryptography, particularly ECC, on PLCs.

Table 4. Runtimes of Modular Reduction

$\|q\|$	Runtime
224 bits	5.466 ms
256 bits	7.748 ms
384 bits	14.450 ms

9 Conclusion

In this work, we proposed several optimization techniques for achieving more efficient lightweight intermittent message authentication for PLC based on LiS and the UHF tree of LARP. Our optimized scheme can handle prolonged interruption of the signer, mitigate growth fluctuations of computational overhead, and significantly reduce it. We also implemented the fast modular reduction of pseudo-Mersenne prime on Allen Bradley ControlLogix 5571, and our optimized scheme's efficiency has been demonstrated by its implementation on PLC.

However, there are still two open questions that require further exploration. First, it is worth investigating whether there is a better synchronization method that does not rely on the UHF tree. Although our optimized scheme has greatly mitigated computational overhead fluctuations, they still exist to some extent. Second, there is room for further speedup of big-integer operations for PLCs by utilizing more efficient algorithms. We encourage researchers to optimize big-integer operations and to implement public key cryptography for PLCs.

Acknowledgements. This work is supported by the Natural Science Foundation of China (Grant No. 62372386), the Natural Science Foundation of Chongqing (Grant No. CSTB2022NSCQ-MSX0437), and the Fundamental Research Funds for the Central Universities (Grant No. SWU-KR22003). This research is also partly supported by the National Research Foundation, Singapore, under its National Satellite of Excellence Programme "Design Science and Technology for Secure Critical Infrastructure: Phase II". Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore.

References

1. Miracl (2018). https://github.com/miracl
2. Allen Bradley: Logix5000 controllers. Rockwell Automation (2009)
3. Allen Bradley: Logix 5000 Controllers Add On Instructions. Rockwell Automation (2018)
4. Allen Bradley: Logix 5000 Controllers Design Considerations. Rockwell Automation (2018)
5. Allen Bradley: Logix 5000 Controllers Structured Text. Rockwell Automation (2018)
6. Bassham, L., Çalık, Ç., McKay, K., Turan, M.S.: Submission requirements and evaluation criteria for the lightweight cryptography standardization process. In: NIST (2018)
7. Bernstein, D.J.: Curve25519: new Diffie-Hellman speed records. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T. (eds.) PKC 2006. LNCS, vol. 3958, pp. 207–228. Springer, Heidelberg (2006). https://doi.org/10.1007/11745853_14
8. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Commun. ACM **13**(7), 422–426 (1970)
9. Carter, L., Wegman, M.N.: Universal classes of hash functions. J. Comput. Syst. Sci. **18**(2), 143–154 (1979)

10. Duka, A.V.: Software implementation and benchmarking of TinyJAMBU on programmable logic controllers. In: Moldovan, L., Gligor, A. (eds.) The 16th International Conference Interdisciplinarity in Engineering, Inter-Eng 2022, LNNS, vol. 605, pp 889–899. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-22375-4_73
11. Giraldo, J., Sarkar, E., Cárdenas, A.A., Maniatakos, M., Kantarcioglu, M.: Security and privacy in cyber-physical systems: a survey of surveys. IEEE Des. Test **34**(4), 7–17 (2017)
12. Goldwasser, S., Micali, S., Rivest, R.L.: A digital signature scheme secure against adaptive chosen-message attacks. SIAM J. Comput. **17**(2), 281–308 (1988)
13. Gura, N., Patel, A., Wander, A., Eberle, H., Shantz, S.C.: Comparing elliptic curve cryptography and RSA on 8-bit CPUs. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 119–132. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-28632-5_9
14. Katz, J., Lindell, Y.: Introduction to Modern Cryptography. Chapman and Hall/CRC Press, Boca Raton (2007)
15. Krawczyk, H., Rabin, T.: Chameleon hashing and signatures. IACR Cryptol. ePrint Arch. p. 10 (1998)
16. Langner, R.: Stuxnet: dissecting a cyberwarfare weapon. IEEE Secur. Priv. **9**(3), 49–51 (2011)
17. Menezes, A., van Oorschot, P.C., Vanstone, S.A.: Handbook of Applied Cryptography. CRC Press, Boca Raton (1996)
18. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. Decentralized Business Review, p. 21260 (2008)
19. Naor, M., Yogev, E.: Bloom filters in adversarial environments. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. LNCS, vol. 9216, pp. 565–584. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48000-7_28
20. Pagh, A., Pagh, R., Rao, S.S.: An optimal bloom filter replacement. In: SODA, pp. 823–829. SIAM (2005)
21. Song, H., Fink, G.A., Jeschke, S.: Security and Privacy in Cyber-Physical Systems: Foundations, Principles, and Applications. Wiley, Hoboken (2017)
22. Tiegelkamp, M., John, K.H.: IEC 61131-3: Programming industrial automation systems, vol. 166. Springer, Berlin (2010). https://doi.org/10.1007/978-3-662-07847-1
23. Wu, H., Huang, T.: Tinyjambu: a family of lightweight authenticated encryption algorithms (version 2). In: NIST (2021)
24. Yang, Z., Bao, Z., Jin, C., Liu, Z., Zhou, J.: PLCrypto: a symmetric cryptographic library for programmable logic controllers. IACR Trans. Symmetric Cryptol. **2021**(3), 170–217 (2021)
25. Yang, Z., et al.: LARP: a lightweight auto-refreshing pseudonym protocol for V2X. In: SACMAT, pp. 49–60. ACM (2022)
26. Yang, Z., Jin, C., Tian, Y., Lai, J., Zhou, J.: Lis: lightweight signature schemes for continuous message authentication in cyber-physical systems. In: AsiaCCS, pp. 719–731. ACM (2020)