10-2020

LiS: Lightweight Signature Schemes for continuous message authentication in cyber-physical systems

Zheng YANG

Chenglu JIN

Yangguang TIAN

Junyu LAI

Jianying ZHOU

LiS: Lightweight Signature Schemes for Continuous Message Authentication in Cyber-Physical Systems

Zheng Yang
Singapore University of Technology and Design
zheng_yang@sutd.edu.sg

Chenglu Jin
New York University
chenglu.jin@nyu.edu

Yangguang Tian
Singapore Management University
ygtian@smu.edu.sg

Junyu Lai
University of Electronic Science and Technology of China
laijy@uestc.edu.cn

Jianying Zhou
Singapore University of Technology and Design
jianying_zhou@sutd.edu.sg

ABSTRACT

Cyber-Physical Systems (CPS) provide the foundation of our critical infrastructures, which form the basis of emerging and future smart services and improve our quality of life in many areas. In such CPS, sensor data is transmitted over the network to the controller, which will make real-time control decisions according to the received sensor data. Due to the existence of spoofing attacks (more specifically to CPS, false data injection attacks), one has to protect the authenticity and integrity of the transmitted data. For example, a digital signature can be used to solve this issue. However, the resource-constrained field devices like sensors cannot afford conventional signature computation. Thus, we have to seek for an efficient signature mechanism that can support the fast and continuous message authentication in CPS, while being easy to compute on the devices.

To this end, we introduce two Lightweight Signature schemes (LiS), which are suitable for continuous message authentication commonly seen in cyber-physical systems. In our constructions, we exploit the efficient hash collision generation property of a chameleon hash function to transform a chameleon hash function into signature schemes. In our schemes, the signature of a message m is the randomness r associated with m in a chameleon hash function, such that they can lead to a hash collision with a given message randomness pair (m', r'). Thus, the task of a signer is to generate the collision using the private key of the underlying chameleon hash function, and a verifier can verify the signature by checking the hash collision with a known message and randomness pair.

We also specifically instantiate the chameleon hash function in such a way that it leads to a fast signing procedure and an optimal storage requirement on the signer side. The optimized signing algorithms are very efficient. Namely, our first scheme requires only three additions and two multiplications, and only one additional hash is needed in the second scheme to resist adaptive chosen message attacks. In addition, the size of the signing key in our schemes is a small constant-sized bit string, which well fits CPS applications.

CCS CONCEPTS

• **Security and privacy → Digital signatures**; *Key management*; *Distributed systems security*.

KEYWORDS

Digital Signature; Chameleon Hash; Continuous Message Authentication; Bloom Filter

ACM Reference Format:
Zheng Yang, Chenglu Jin, Yangguang Tian, Junyu Lai, and Jianying Zhou. 2020. LiS: Lightweight Signature Schemes for Continuous Message Authentication in Cyber-Physical Systems. In *15th ACM Asia Conference on Computer and Communications Security (ASIA CCS'20), October 5–9, 2020, Taipei, Taiwan.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3320269.3372195

1 INTRODUCTION

Cyber-Physical Systems (CPS) integrate various cyber computation, physical devices, and networking technology to control physical processes through data exchange in real-time. The emerging CPS is expected to encompass every aspect of our lives, which generates a paradigm shift towards a hyper-connected society. Up to now, CPS devices are widely used in navigation systems, smart grid, smart city application domains. However, due to the limitation in both computation and storage, many modern cryptographic schemes cannot run on CPS devices, which results in serious security and privacy issues [1–3].

Thus, it is notoriously important to ensure that the data that comes from CPS devices has not been changed by attackers [4, 5]. A natural solution to protect the authenticity and integrity of a message is to use a message authentication code (MAC). It allows the verifier (who possess a shared key with the message sender) to detect any change to the message content [6]. However, MAC has a potential security risk: when the key stored on the verifiers gets leaked, all the future message authentication codes will possibly be compromised. Notice that, in more physically isolated CPS, like nuclear plants and manufacturing systems, the verifiers (servers), are actually more vulnerable to cyber attacks than CPS devices. This is because they are connected to enterprise networks or even the Internet. For example, the famous Stuxnet worm compromised the engineering workstations first in Iranian nuclear plants before it got a footprint in the controller systems [1].

Another standard solution is to use cryptographic digital signatures, which allow receivers to verify the origin of a message using a 'public' key so that it can be secure in a verifier breach. Although there are many online efficient (computationally secure) digital signature schemes, such as ElGamal signature [7] and Schnorr signature [8, 9], they require expensive cryptographic operations like modular exponentiation which is too complex for resource-constrained CPS devices.

To make a digital signature scheme deployable on constrained devices, Even *et al.* proposed an online/offline signature paradigm [10]. The idea is to use a trusted and powerful server to pre-compute some expensive operations in an offline phase, so the signer does not need to compute complex operations, but the signer needs to securely store a large number of private intermediate values D generated by the server, and the size of D has a linear relation with the number of signatures to be signed [10].

It is suggested in [11–13] that the offline phase for computing D can be carried out either during the device manufacturing process or by the device itself as a background computation. However, none of these two ways of generating D is perfect in CPS. The first solution requires a large amount of storage overhead on the device (e.g., 97MB for seven-day usage with a message rate at 1 second per message [12, Schnorr]), but it is generally infeasible on CPS devices. Although some sorts of replenishment of D might be possible, it may interrupt the normal operation and communication of the CPS devices, which need to keep sending data measured in real-time. The second solution demands a lot of computational power and idle time on the device side. However, the CPS devices keep generating data at a fast pace, so there is no enough idle time for it to compute these operations in the background. For example, in an automatic identification system used on ships [14], each time slot for sending a message is just 26.66 milliseconds, which is too short for an exponentiation operation on an embedded device.

In this work, we are specifically motivated to design signature schemes that can be *efficiently* used in CPS for *continuous message authentication*. In particular, we introduce signature schemes that are optimized for the signer, in terms of both computation overhead and storage overhead, and the signer only needs to store a constant-sized signing key which does not need to be replenished for continuous and uninterrupted message authentication.

Our Work. Our construction leverages on a chameleon hash function [15] and a pre-computation strategy to shift the 'burdens' of computation and storage from a signer to verifiers and possibly a trusted third-party server. It is worth noting that the verifiers in CPS are servers that have enough computation power and have storage. Recall that, given a message and randomness pair (m', r'), a chameleon hash function CHF allows one to use its secret key sk to efficiently compute a collision r for a message $m \neq m'$ such that $\mathsf{CHF}(m, r) = \mathsf{CHF}(m', r')$. The general idea is to utilize a trusted server to take as inputs a set of dummy message/randomness pair (m'_i, r'_i) (for $1 \leq i \leq \ell$) and pre-compute the chameleon hash values which will be used as a part of the verification key vk, where ℓ is the maximum number of signatures can be verified by vk. For the online signature generation, the signer only needs to compute the collision r_i as the signature of m_i based on a used dummy random r'_i.

In order to further optimize the signing algorithm for the resource-constrained signers, we propose to fix all dummy message m'_i as a constant M, and use a universal hash function UHF to chain up all dummy randomness, i.e., $r'_i = \mathsf{UHF}(k, r'_{i-1})$ for $1 \leq i \leq \ell$, and r_0 is chosen randomly, where k is a random hash key of UHF. As a result, the signer only needs to store a few hundreds of bits $(sk, sk \cdot M, r'_0, k)$ for signing. Also, we applied some arithmetic tricks in our instantiation of the chameleon hash function [15], which successfully saves one big-number modular division in the signing algorithm. After our extensive optimization, only *three modular additions and two modular multiplications* are needed to generate one signature in our first protocol LiS_1. One additional hash computation is required in the second protocol LiS_2.

However, in the above naïve construction, the size of the verification key is dominated by the pre-computed hash values. If the vk can support the verification of many signatures (i.e., ℓ is large), then vk may get very large too. To reduce the size of vk, we propose to use a Bloom filter [16] to compress vk. For certain applications that can tolerate small errors, such as Globe Positioning System (GPS), we can even allow a relatively large false-positive rate for the Bloom filter.

Nevertheless, the verification key of our schemes will still be used up after ℓ signatures. We further develop two verification key replenishment solutions to enable unlimited signing capability. Notice that the chameleon hash keys (both secret and public key) do not need to be changed after a verification key update, so *the signer can keep signing messages without being interrupted.* In an update of vk, the verifiers need to get a new set of authenticated chameleon hash values for future verification. Our first verification key replenishment approach is to outsource the computation of the new chameleon hash values to a trusted server, which periodically computes the chameleon hash values based on the dummy message M and the dummy randomness r' and publishes them to a bulletin (where verifiers have access) together with a signature of the server. Moreover, the second verification key replenishment solution can be used when all the verifiers are honest (and not controlled by attackers). For instance, in industrial control systems, the verifier can be a server located in an isolated control center, which is trusted by all signer (i.e., the client devices). In this case, one can let the verifier have the dummy message M and dummy randomness r', so it can generate the verification key on the fly during the verification procedure. We specifically customized a verification algorithm for this scenario. Notice that even if the dummy message/randomness pairs (M, r'_i) are leaked to an adversary, he still cannot forge a valid signature without knowing the secret signing key.

Contributions. We made the following contributions in our paper:

(1) We propose two lightweight signature schemes LiS with an optimized signing algorithm in terms of both computation and storage. Our first protocol LiS_1 is designed to sign small messages and to provide security guarantee against weak chosen message attacks, so it is suitable for the applications

(such as GPS), which have an untamperable data source. We extend our first protocol to the second one LiS_2 by adding a hash function, which is modeled as a random oracle, to not only sign large messages, but also achieve adaptive security.

(2) We instantiate and optimize the proposed algorithms. As a result, the signing algorithm of LiS_1 requires three additions and two multiplications, and one more hash is needed in LiS_2.

(3) We propose two verification key replenishment solutions based on different assumptions of the verifiers. New verification algorithms are designed for honest verifiers.

(4) We show the security results of our schemes with formal security proofs. LiS_1 and LiS_2 are proved in the standard model and the random oracle model, respectively.

(5) We implement our signature schemes and evaluate the performance experimentally. Our results show that they are efficient for continuous message authentication in CPS.

Organization. We introduce necessary preliminaries that will be used frequently in the rest of the paper in Section 2. Section 3 presents a signature framework with two concrete signature schemes based on chameleon hash functions. The security of the proposed schemes is analyzed in Section 4. We show instantiations and optimizations in Section 5. Section 6 presents two verification key replenishment solutions. Some candidate applications of our schemes in cyber-psychical systems are discussed in Section 7. Performance analysis and evaluation results are presented in Section 8. In Section 9, we review the literature related to our work. The paper concludes in Section 10.

2 PRELIMINARIES

Here we briefly review the notions and cryptographic primitives that our constructions rely on.

We denote with κ the security parameter, $\emptyset$ an empty string, and with $[n] = \{1, \ldots, n\} \subset \mathbb{N}$ the set of integers between 1 and n. If X is a set, then $x \xleftarrow{\$} X$ denotes the action of sampling a uniformly random element from X. If X is a probabilistic algorithm, then $x \xleftarrow{\$} X$ denotes that X is run with fresh random coins and returns x. Let $\|$ be an operation to concatenate two strings, $|\cdot|$ be an operation to get the bit-length of a variable, and # be an operation to get the number of elements in a set.

2.1 Universal Hash Functions

A universal hash function (UH) family [17]: $\mathcal{K}_{\mathsf{UH}} \times \mathcal{M}_{\mathsf{UH}} \rightarrow \mathcal{R}_{\mathsf{UH}}$, refers to a family of hash functions which guarantees a low number of collisions in expectation even, where $\mathcal{K}_{\mathsf{UH}}$, $\mathcal{M}_{\mathsf{UH}}$ and $\mathcal{R}_{\mathsf{UH}}$ be key, message and output space of UH, respectively. These spaces are determined by the security parameter κ.

Definition 2.1. We say that a set of hash functions UH is universal hash function family if: i) we uniformly choose a hash function $\mathsf{UHF} \in \mathsf{UH}$ by sampling a random hash key $k \xleftarrow{\$} \mathcal{K}_{\mathsf{UH}}$, ii) and $\forall (x, y) \in \mathcal{M}_{\mathsf{UH}}$ we have the probability $\Pr[\mathsf{UHF}(k, x) = \mathsf{UHF}(k, y)] \leq \frac{1}{\#\mathcal{M}_{\mathsf{UH}}}$.

2.2 Chameleon Hash Functions

An important cryptographic primitive that we will use is chameleon hash function [15]. A chameleon hash function $\mathsf{CH}(pk, \cdot, \cdot) : \mathcal{PK}_{\mathsf{CH}} \times \mathcal{M}_{\mathsf{CH}} \times \mathcal{R}_{\mathsf{CH}} \rightarrow \mathcal{Y}_{\mathsf{CH}}$ is associated with a pair of public key $pk \in \mathcal{PK}_{\mathsf{CH}}$ and private key $sk \in \mathcal{SK}_{\mathsf{CH}}$, where $(\mathcal{PK}_{\mathsf{CH}}, \mathcal{SK}_{\mathsf{CH}})$ are public and private key spaces, respectively, $\mathcal{M}_{\mathsf{CH}}$ is the message space, $\mathcal{R}_{\mathsf{CH}}$ is the randomness space and $\mathcal{Y}_{\mathsf{CH}}$ is the output space. These public/secret key pairs are generated by a PPT algorithm $(pk, sk) \xleftarrow{\$} \mathsf{CHKGen}(1^\kappa)$. If the key is clear from the context, we will write $\mathsf{CH}(m, r)$ for $\mathsf{CH}(pk, m, r)$.

A hash value generated by $\mathsf{CH}(m, r)$ on input a message m and a random string r satisfies the following properties:

- Collision resistance. There is no efficient algorithm that on input the public key pk can output two pairs (m_1, r_1) and (m_2, r_2) such that $m_1 \neq m_2$ and $\mathsf{CH}(m_1, r_1) = \mathsf{CH}(m_2, r_2)$, except with negligible probability in the security parameter κ.
- Trapdoor collisions. There exists an efficient deterministic algorithm CHColl that on input the secret key sk, and $(r, m, m') \in \mathcal{R}_{\mathsf{CH}} \times \mathcal{M}_{\mathsf{CH}} \times \mathcal{M}_{\mathsf{CH}}$, outputs a value $r' \in \mathcal{R}_{\mathsf{CH}}$ such that $\mathsf{CH}(pk, m, r) = \mathsf{CH}(pk, m', r')$.
- Uniformity. For an arbitrary public key pk output by CHKGen, all messages $m \in \mathcal{M}_{\mathsf{CH}}$ generate equally distributed hash values $\mathsf{CH}(m, r)$ when drawing $r \xleftarrow{\$} \mathcal{R}_{\mathsf{CH}}$ uniformly at random. This property ensures that a third party is unable to examine the value hash from deducing any information about the hashed message.

Given an adversary $\mathcal{A}$ and a chameleon function CHF, the CH security game $G^{\mathsf{CH}}_{\mathcal{A},\mathsf{CHF}}(\kappa)$ is defined in Figure 1.

```
Proc.Init():                          Proc.Finalize($m$, $m'$, $r$, $r'$):
$(sk, pk) \xleftarrow{\$} \mathsf{CHKGen}(1^\kappa)$     If $(m, m') \in \mathcal{M}_{\mathsf{CH}} \wedge$
OUTPUT $pk$                               $(r, r') \in \mathcal{R}_{\mathsf{CH}} \wedge m \neq m' \wedge$
                                          $\mathsf{CHF}(m, r) = \mathsf{CHF}(m', r')$
                                          OUTPUT 1
                                      ELSE OUTPUT 0
```

Figure 1: Procedures used to define security for CH.

Definition 2.2. We denote with $\mathsf{Adv}^{\mathsf{CH}}_{\mathcal{A},\mathsf{CHF}}(\kappa) := \Pr[G^{\mathsf{CH}}_{\mathcal{A},\mathsf{CHF}}(\kappa) = 1]$ the advantage of a PPT adversary $\mathcal{A}$ in breaking the security of the chameleon hash function CHF under the security parameter κ. We say CHF is secure if no PPT adversary has non-negligible advantage $\mathsf{Adv}^{\mathsf{CH}}_{\mathcal{A},\mathsf{CHF}}(\kappa)$.

The security of chameleon hash functions can be based on standard computational hardness assumptions like the discrete logarithm assumption or the factoring assumption.

2.3 Digital Signature Schemes

We define a digital signature scheme SIG with three probabilistic polynomial time (PPT) algorithms (KGen, Sign, Verify). We assume that a signature scheme is associated with public and secret key spaces $\{\mathcal{PK}_{\mathsf{SIG}}, \mathcal{SK}_{\mathsf{SIG}}\}$, message space $\mathcal{M}_{\mathsf{SIG}}$, and signature space $\mathcal{S}_{\mathsf{SIG}}$ in the security parameter κ. We denote the bit-length of the space $\mathcal{R}_s$ by ℓ_s which is determined by κ. The algorithms of SIG are defined as follows:

- KGen$(1^\kappa, \ell, \mathsf{aux})$: This algorithm takes as input the security parameter 1^κ, the maximum number ℓ of signature that SIG can generate, and an auxiliary input aux, and generates the secret key sk and the verification key vk.
- Sign(sk, m): This is the signing algorithm that generates a signature $\sigma \in \mathcal{S}_{\mathsf{SIG}}$ for a message $m \in \mathcal{M}_{\mathsf{SIG}}$ with the signing key sk.
- Verify(vk, m, σ): This is the verification algorithm that takes as input a verification key vk, a message m and a signature σ, outputs 1 if σ is a valid signature for m under vk, and 0 otherwise.

Here we consider both weak and adaptive chosen message attacks against SIG. The resilience of weak chosen message attacks is enough for specific applications that the signer may have reliable message sources (for most of running time), e.g., sensors deployed in an isolated environment, or the satellites that send GPS signals. Besides, the weaker security notion allows us to achieve high performance which is crucial for the resource-constrained devices in CPS. Whereas, the adaptive security is needed in a more complicated environment with sophisticated adversaries.

We define a security game $G^{\mathsf{SIG}}_{\mathcal{A},\mathsf{S}}(\kappa, \ell)$ that is played between an adversary $\mathcal{A}$ and a challenger based on a signature scheme S, the security parameter κ, and the number ℓ that bounds the signatures that the adversary can obtain. In the security game, $\mathcal{A}$ may ask the procedures defined in Figure 2. Concretely, the adversary $\mathcal{A}$ proceeds with the game by sequentially calling the procedures Proc.Init, Proc.SQuery and Proc.Finalize. We stress that we can obtain the adaptive security game by removing those boxed steps or elements. Namely, the security notions that we defined are also known as *strongly existential unforgeable against weak chosen message attacks* (SEUF-wCMA) and *strongly existential unforgeable against adaptive chosen message attacks* (SEUF-CMA), respectively.

```
Proc.Init($\boxed{M}$) :
$(sk, vk) \xleftarrow{\$} \mathsf{S.KGen}(1^\kappa, \ell, \boxed{M})$:
$cnt := 0$
OUTPUT $vk$

Proc.SQuery($m$) :
$\boxed{\text{IF } m \notin M}$
    $\boxed{m' \xleftarrow{\$} \mathcal{M}_{\mathsf{SIG}}}$
    $\boxed{m := m'}$
IF $cnt \geq \ell$ OUTPUT$\perp$
$\sigma := \mathsf{S.Sign}(sk_{\mathsf{id}_C}, m)$
$cnt := cnt + 1$
APPEND $(m, \sigma) \rightarrow$ QD
OUTPUT$(m, \sigma)$

Proc.Finalize($m^*, \sigma^*$) :
If $\mathsf{S.Verify}(vk, m^*, \sigma^*) = 1$
$\wedge (m^*, \sigma^*) \notin$ QD
    OUTPUT 1
ELSE OUTPUT 0
```

Figure 2: Procedures used to define security for SIG.

Definition 2.3. Let $\mathsf{Adv}^{\mathsf{SIG}}_{\mathcal{A},\mathsf{S}}(\kappa, \ell) := \left|\Pr[G^{\mathsf{SIG}}_{\mathcal{A},\mathsf{S}}(\kappa, \ell) = 1] - \frac{1}{2}\right|$ be the advantage of a PPT adversary $\mathcal{A}$ in breaking the security of a signature scheme S under the security parameter κ. We say S is secure if no PPT adversary has non-negligible advantage $\mathsf{Adv}^{\mathsf{SIG}}_{\mathcal{A},\mathsf{S}}(\kappa, \ell)$.

2.4 Bloom filter

Bloom filter [16] is a probabilistic data structure that provides space-efficient storage of a set and that can efficiently test whether an element is a member of the set. The probabilistic property of BF may lead to false positive matches, but not false negatives. The more elements are in the BF, the higher chance to get a false positive match insertion. To reduce its false positive rate, we follow the approach of [18], i.e., a BF with $1.44\epsilon N$ bits for a set with size N has a false positive rate (FPR) of $2^{-\epsilon}$.

We review the algorithms of a Bloom filter as follows:

- Init(N, ϵ): On input, a set size N, the initialization algorithm initiates the Bloom filter of bit length $1.44\epsilon N$.
- Insert(m): Element insertion algorithm takes an element m as input, and inserts m into BF.
- Check(m): Element check algorithm returns 1 if an element m is in BF, and 0 otherwise.
- Pos(m): Position update algorithm computes positions to be changed for element m in BF.

3 LIS: LIGHTWEIGHT SIGNATURE SCHEMES FROM CHAMELEON HASH

In this section, we propose a family of lightweight signature schemes called as LiS = (LiS$_1$, LiS$_2$) from chameleon hash functions, Bloom filters, and universal hash functions.

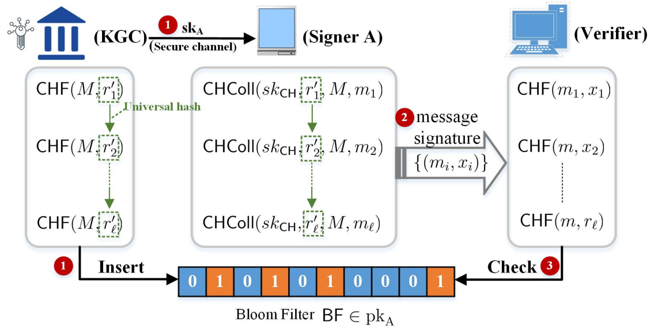

Figure 3: Overview of LiS. KGC stands for *key generation center*.

Design Rational. The system overview of LiS is shown in Figure 3. Our primary design goal is to enable the resource-constrained signer device to authenticate its message in an extremely cheap way, and the scheme should be resilient to verifier breaches. To this end, we leverage on a chameleon hash function CHF and a pre-computation strategy. In a *naïve solution*, one can simply compute a verify point $t = \mathsf{CHF}(m', r')$ based on a dummy message m' and a randomness r', and send the t to the verifier for verification. To authenticate a message m online, the signer can compute a collision $x := \mathsf{CHColl}(r', m', m)$ as the signature for m. The signer can repeat the above procedure polynomial times to sign multiple messages.

However, the above naïve solution is insecure since an adversary can recover the secret key sk_{CH} of the chameleon hash function given a pair of collided information (two signature/message pairs in the above example). To fix the problem of the above naïve solution, we should only use each pre-computed verify point t once. Therefore, in our solution, we pre-compute verify points of a polynomial number ℓ based on a set of dummy random values $\{r'_i\}_{i\in[\ell]}$ and a single dummy message M, such that $t_i := \mathsf{CHF}(M, r'_i)$. To avoid store these randomnesses at the signer, we generate them using a universal hash function UHF, such that $r'_i := \mathsf{UHF}(k, r'_{i-1})$, where k is the hash key of UHF. So that the signer only needs to store the keys sk_{CH}, M, r'_0, and k. For online message authentication, the signer only needs to recover the corresponding dummy random values and compute the collisions, which is very efficient, i.e., only two multiplication and three additions are required, according to our instantiation and optimization in Section 5. The verification key comprising $\{t_i\}$ might be costly. To reduce the storage cost of the verifier, one could use a Bloom filter to compress these verify points t_i. This would save over 100x of storage costs. Notice that the verification key stored at the verifier can be public so it can resist a verifier breach.

Based on the above idea, we designed two signature schemes LiS_1 and LiS_2, respectively. LiS_1 is weakly secure and suitable for small messages with a few hundred bits (determined by $\mathcal{M}_{\mathsf{CH}}$) but is the most efficient one. The second protocol LiS_2 can provide adaptive security, but one more cryptographic hash function is required.

3.1 Weakly Secure Signature Scheme LiS_1

Description of LiS_1. This scheme relies on a universal hash function UHF, a chameleon hash function CHF, and a Bloom filter BF, which are defined in Section 2. The algorithms of the proposed scheme LiS_1 are shown in Figure 4. Basically, LiS_1 consists of three functions which are briefly illustrated as follows:

- *Initialization*: A signer id_C first runs the key generation algorithm of the chameleon hash function $(sk_{\mathsf{CH}}, pk_{\mathsf{CH}}) \overset{\$}{\leftarrow} \mathsf{CHKGen}(1^\kappa)$ to generate a pair of secret/public key, and samples a random key $k \overset{\$}{\leftarrow} \mathcal{K}_{\mathsf{UH}}$ for the universal hash function UHF, a random message $M \overset{\$}{\leftarrow} \mathcal{M}_{\mathsf{CH}}$, and an initial random value r'_0. Through the parameter aux, id_C should be able to parse the 'false positive parameter' ϵ of the Bloom filter from it. A Bloom filter instance BF is initialized by $\mathsf{BF.Init}(\ell, \epsilon)$. For $i \in [\ell]$, id_C generates ℓ dummy random values such that $r'_i := \mathsf{UHF}(k, r'_{i-1})$, and the verify points $t_i := \mathsf{CHF}(M, r'_i)$ for future use. Meanwhile, id_C inserts those verify points into the Bloom filter $\mathsf{BF.Insert}(t_i)$.[1] A random variable $r' := r'_1$ which is used for generating the next signature. Note that if any $r'_i = r'_j$ for $i \neq j$ then id_C re-run the key generation algorithm. Eventually, the secret key and the verification key of id_C are $sk_{\mathsf{id}_\mathsf{C}} := (sk_{\mathsf{CH}}, k, r', M)$ and $vk_{\mathsf{id}_\mathsf{C}} := (\mathsf{BF}, pk_{\mathsf{CH}})$. The verification key $vk_{\mathsf{id}_\mathsf{C}} := (\mathsf{BF}, pk_{\mathsf{CH}})$ will be sent to potential verifier(s), and the secret key $sk_{\mathsf{id}_\mathsf{C}}$ will be stored privately by the signer id_C. To authenticate the first verification key, the signer can either transmit it to the designated verifier via a secure channel (that is isolated from adversaries) or ask a trustworthy third party to sign it digitally.
- *Signing*: Upon obtaining a message m that requires authentication, id_C first retrieves the stored secret key $sk_{\mathsf{id}_\mathsf{C}} := (sk_{\mathsf{CH}}, k, r', M)$. Thanks to the trapdoor collisions property of CHF, id_C can compute the signature x for m as $x := \mathsf{CHColl}(sk_{\mathsf{CH}}, r', M, m)$. Then, id_C can send m together with the signature x to the verifier. After this , id_C updates the dummy randomness r' to the next one as $r' := \mathsf{UHF}(k, r')$. In fact, the update of the dummy randomness can be done at any time before the next message authentication is carried out, so its performance overhead can possibly be hidden in the background.
- *Verification:* Upon receiving a message m and its signature x from the signer id_C, the verifier id_S verifies it by checking that whether the resultant hash value $t = \mathsf{CHF}(m, x)$ is in the Bloom filter, i.e., $\mathsf{BF.Check}(t)$.

Correctness. Since the Bloom filter does not have any false negative, for every $x := \mathsf{LiS}_1.\mathsf{Sign}(sk_{\mathsf{id}_\mathsf{C}}, m)$, it must have that $\mathsf{BF.Check}(\mathsf{CHF}(m, x)) = 1$ since $\mathsf{CHF}(m, x) = \mathsf{CHF}(m'_i, t_i)$ which is inserted into BF during initialization.

Remark 1. *To obtain better online efficiency, the signer could compute the universal hash operations offline (or during its idle time). Of course, the signer can also pre-compute and cache many such universal hash values as online/offline signature schemes [11, 12]. Then the signer only needs to run* CHColl *in the online signing phase, and therefore the signing algorithm could be approximately 2x faster.*

3.2 Adaptively Secure Signature Scheme LiS_2

In this subsection, we introduce a signature scheme LiS_2 that can resist adaptively chosen message attacks. LiS_2 is basically derived from LiS_1 by using an additional cryptographic hash function $h_1 : \{0,1\}^* \to \mathcal{M}_{\mathsf{CH}}$ which will be modeled as a random oracle. Hence LiS_2 can be used to authenticate a message with an arbitrary size, unlike LiS_1, which is constrained by the size of $\mathcal{M}_{\mathsf{CH}}$. We let $\mathcal{R}_{\mathsf{CH}} = \{0,1\}^{\ell_r}$ be the randomness used in this construction. Besides, the universal hash function is replaced with another hash function $h_2 : \{0,1\}^* \to \mathcal{R}_{\mathsf{CH}}$.[2]

Here we prefer to use an OAEP alike approach [19] for achieving adaptive security (i.e., binding a randomness to each message using h) so that it can have an optimal signing efficiency. Although there might be an alternative generic transformation from a weakly secure signature to an adaptively secure signature such as [20], it is less computationally efficient than ours.

Description of LiS_2. The algorithms of the proposed scheme LiS_2 are shown in Figure 5. LiS_2 consists of three functions which are briefly illustrated as follows:

- *Initialization*: The signer id_C first runs the key generation algorithm of the chameleon hash function $(sk_{\mathsf{CH}}, pk_{\mathsf{CH}}) \overset{\$}{\leftarrow} \mathsf{CHKGen}(1^\kappa)$ to generate a pair of secret/public key, and samples a random key $k \overset{\$}{\leftarrow} \mathcal{R}_{\mathsf{CH}}$, a random message $M \overset{\$}{\leftarrow}$

[1] Notice that, even if false positives happen in the initialization phase, they would not affect the correctness and security of our signature schemes.

[2] One of the reasons for using h_2 is that it could help a party (e.g., verifier shown in Figure 7) to quickly compute the non-sequentially generated randomness.

```
**LiS₁.KGen**$(1^\kappa, \ell, \mathsf{aux})$:
$(sk_{\mathsf{CH}}, pk_{\mathsf{CH}}) \xleftarrow{\$} \mathsf{CHKGen}(1^\kappa)$
$k \xleftarrow{\$} \mathcal{K}_{\mathsf{UH}}; \epsilon \leftarrow \mathsf{aux}$
$\mathsf{BF.Init}(\ell, \epsilon)$
$M \xleftarrow{\$} \mathcal{M}_{\mathsf{CH}}; r'_0 \xleftarrow{\$} \mathcal{R}_{\mathsf{UH}}$
For $i \in [\ell]$:
    $r'_i := \mathsf{UHF}(k, r'_{i-1})$
    $t_i := \mathsf{CHF}(M, r'_i)$
    $\mathsf{BF.Insert}(t_i)$
$r' := r'_1; sk_{\mathsf{id_C}} := (sk_{\mathsf{CH}}, k, r', M)$
$vk_{\mathsf{id_C}} := (\mathsf{BF}, pk_{\mathsf{CH}})$
Return $(sk_{\mathsf{id_C}}, vk_{\mathsf{id_C}})$

**LiS₁.Sign**$(sk_{\mathsf{id_C}}, m)$:
$x := \mathsf{CHColl}(sk_{\mathsf{CH}}, r', M, m)$
$r' := \mathsf{UHF}(k, r')$
Return $x$

**LiS₁.Verify**$(vk_{\mathsf{id_C}}, m, x)$:
$vr := 0$
$t := \mathsf{CHF}(m, x)$
$vr := \mathsf{BF.Check}(t)$
Return $vr$
```

Figure 4: Algorithms of LiS_1.

$\mathcal{M}_{\mathsf{CH}}$. $\mathsf{id_C}$ would additionally initializes two cryptographic hash functions $h_1 : \{0,1\}^* \to \mathcal{M}_{\mathsf{CH}}$ and $h_2 : \{0,1\}^* \to \mathcal{R}_{\mathsf{CH}}$. A Bloom filter instance BF is initialized by $\mathsf{BF.Init}(\ell, \epsilon)$. For $i \in [\ell]$, $\mathsf{id_C}$ generates ℓ dummy random values such that $r'_i := h_2(k||i)$, and the verify points $t_i := \mathsf{CHF}(M, r'_i)$ for the future use. Meanwhile, $\mathsf{id_C}$ inserts those verify points into the Bloom filter $\mathsf{BF.Insert}(t_i)$. $\mathsf{id_C}$ initializes a counter $cnt = 0$ to count the number of generated signatures. As a final outcome of the procedure, the secret key and the public verification key of $\mathsf{id_C}$ are $sk_{\mathsf{id_C}} := (sk_{\mathsf{CH}}, k, M, cnt)$ and $vk_{\mathsf{id_C}} := (\mathsf{BF}, pk_{\mathsf{CH}})$.

- *Signing*: To authenticate a message m, $\mathsf{id_C}$ first samples a random value $N \xleftarrow{\$} \mathcal{R}_2$, and computes $y := h(m||N)$ and $r'_{cnt} := h(k||cnt)$. After this, it updates the counter $cnt = cnt + 1$. Then the signer $\mathsf{id_C}$ generates the signature x for y as $x := \mathsf{CHColl}(sk_{\mathsf{CH}}, r'_i, M, y)$, and sends the tuple (m, N, x) to the verifier.
- *Verification:* Upon receiving (m, N, x), the verifier $\mathsf{id_S}$ verifies it by checking that whether the resultant hash value $t = \mathsf{CHF}(h_1(m||N), x)$ is in the Bloom filter.

Correctness. The correctness of LiS_2 is implied by LiS_1. The newly added random value N and the hash operations do not change the authentication property.

4 SECURITY ANALYSIS

In this section, we present the security results of our proposed schemes with formal proofs in the standard model.

4.1 Security Analysis of LiS_1

Theorem 4.1. *We assume the chameleon hash function* CHF *and the universal hash function* UHF *are secure as defined in Section 2. Then* LiS_1 *with given parameters* κ *and* ℓ *is secure against selective chosen message attacks with advantage*

$$\mathsf{Adv}^{\mathsf{SIG}}_{\mathcal{A},\mathsf{LiS}_1}(\kappa, \ell) \le \mathsf{Adv}^{\mathsf{CH}}_{\mathcal{A},\mathsf{CHF}}(\kappa) + 2^{-\epsilon}$$

.

The full proof of Theorem 4.1 is presented in Appendix B. In Table 1, we summarize sequence of games to present the main ideas of the proof.

Table 1: Sequence of games for LiS_1

Game	Description & Modification
0	Real experiment following original algorithms
1	Replace each output of the universal hash function UHF with a uniform random value
2	Randomly generate a signature x_i for each message m_i instead of running CHColl. The output of CHColl is statistically close to a uniform random value
3	Reduce the security to that of the chameleon hash function CHF
4	Reduce the security to the false positive error of Bloom Filter BF

4.2 Security Analysis of LiS_2

Theorem 4.2. *We assume the chameleon hash function* CHF *is secure as defined in Section 2, and the hash function h is modeled as a random oracle. Then* LiS_2 *with given parameters* κ *and* ℓ *is secure against adaptive chosen message attacks with advantage*

$$\mathsf{Adv}^{\mathsf{SIG}}_{\mathcal{A},\mathsf{LiS}_2}(\kappa, \ell) \le \mathsf{Adv}^{\mathsf{CH}}_{\mathcal{A},\mathsf{CHF}}(\kappa) + \frac{\ell^2}{2^{\ell_r}} + 2^{-\epsilon}$$

.

The proof of this theorem is similar to that of Theorem 4.1. Here we mainly explain why the random oracle h and nonce N can work to provide adaptive security. Recall that, to reduce the security of LiS_1 to that of chameleon hash in the proof of Theorem 4.1, we generated ℓ dummy random messages and random values $\{(m'_i, r'_i)\}_{i \in [\ell]}$. The key point in the reduction is how to use the random oracle to map a message m^* chosen by the adversary to the pre-sampled m'_i. By design, we attach each message m^*_i with a random value N_i so that the string $m^*_i||N_i$ is unique unless a collision happens with a collision probability $\ell^2/2^{\ell_r}$. Due to this fact, we can establish a unique connection between each string $m^*_i||N_i$ and m'_i as $h(m^*_i||N_i) = m'_i$. Hence we can reduce the security of LiS_2 to that of chameleon hash function with a similar proof strategy of that of LiS_1.

```
$\mathsf{LiS_1.KGen}(1^\kappa, \ell, \mathsf{aux})$:
$(sk_{\mathsf{CH}}, pk_{\mathsf{CH}}) \xleftarrow{\$} \mathsf{CHKGen}(1^\kappa)$
$k \xleftarrow{\$} \mathcal{R}_{\mathsf{CH}}; \epsilon \leftarrow \mathsf{aux}$
$\mathsf{BF.Init}(\ell, \epsilon)$
$M \xleftarrow{\$} \mathcal{M}_{\mathsf{CH}}$
For $i \in [\ell]$:
    $r'_i := h_2(k||i)$
    $t_i := \mathsf{CHF}(M, r'_i)$
    $\mathsf{BF.Insert}(t_i)$
$cnt := 1$;
$sk_{\mathsf{id_C}} := (sk_{\mathsf{CH}}, k, M)$
$vk_{\mathsf{id_C}} := (\mathsf{BF}, pk_{\mathsf{CH}})$
Return $(sk_{\mathsf{id_C}}, vk_{\mathsf{id_C}})$
```

```
$\mathsf{LiS_2.Sign}(sk_{\mathsf{id_C}}, m)$:
$N \xleftarrow{\$} \mathcal{R}_2$
$y := h_1(m||N)$
$x := \mathsf{CHColl}(sk_{\mathsf{CH}}, r'_{cnt}, M, y)$
$cnt := cnt + 1$
$r'_{cnt} := h_2(k||cnt)$
Return $x, N$
```

```
$\mathsf{LiS_2.Verify}(vk_{\mathsf{id_C}}, (m, N), x)$:
$vr := 0$
$t := \mathsf{CHF}(h_1(m||N), x)$
$vr := \mathsf{BF.Check}(t)$
Return $vr$
```

Figure 5: Algorithms of $\mathsf{LiS_2}$**.**

5 INSTANTIATIONS AND OPTIMIZATIONS

In this section, we show the concrete instantiations and optimizations of our generic building blocks, i.e., chameleon hash function CHF, universal hash function UHF, and the hash functions h_1 and h_2. Let p and q be two large prime numbers, such that $p = u \cdot q + 1$ where u is a small integer. We particularly have that $\mathcal{K}_{\mathsf{CH}} = \mathcal{M}_{\mathsf{CH}} = \mathcal{R}_{\mathsf{CH}} = \mathcal{K}_{\mathsf{UH}} = \mathcal{M}_{\mathsf{UH}} = \mathcal{S}_{\mathsf{SIG}} = \mathbb{Z}_q$, and $\mathcal{Y}_{\mathsf{CH}} = \mathbb{Z}_p$.

The Hash Functions h_1 and h_2. Our choice for both hash functions h_1 and h_2 is the standardized cryptographic hash function SHA2 [22]. Since the range of h_1 is identical to that of h_2, we can instantiate h_1 and h_2 by the same functions.

Universal Hash Function UHF. We instantiate UHF by Multiply-modular scheme proposed in [17]. The key $k = (k_0, k_1)$ of UHF consists of two group elements $k_0 \xleftarrow{\$} \mathbb{Z}_q^*$ and $k_1 \xleftarrow{\$} \mathbb{Z}_q^*$. Given a message m, the hash function evaluates the hash value $y := \mathsf{UHF}(k, m) = k_0 \cdot m + k_1 \pmod q$. Some optimizations can be adopted by following [23].

Instantiation of Chameleon Hash Function. We review the original discrete logarithm based chameleon hash function [15] in Appendix A. To have a better performance in the collision algorithm, we slightly modify the hash evaluation algorithm, and we describe our modified version as follows:

- $\mathsf{CHKGen}(1^\kappa)$: The key generation algorithm samples random group generator g of order q in $\mathbb{Z}_p^*$ and a secret key $sk_{\mathsf{CH}} \xleftarrow{\$} \mathbb{Z}_q^*$, and computes the public key $pk_{\mathsf{CH}} := g^{sk_{\mathsf{CH}}} \pmod p$.
- $\mathsf{CHF}(pk_{\mathsf{CH}}, m, r)$: The evaluation algorithm takes as input a public key $pk_{\mathsf{CH}} \in \mathbb{Z}_p^*$, a message $m \in \mathbb{Z}_q^*$ and a randomness $r \in \mathbb{Z}_q^*$, and outputs a hash value $y := g^r pk_{\mathsf{CH}}^m \pmod p$. In contrast to the algorithm in [15], we just switch the places of m and r, and this change is only conceptual.
- $\mathsf{CHColl}(sk_{\mathsf{CH}}, r', M, m)$: An efficient deterministic collision algorithm CHColl takes as input the secret key sk_{CH}, and $(r', M, m) \in \mathbb{Z}_q^*$, outputs a value $x := M \cdot sk_{\mathsf{CH}} + r' - m \cdot sk_{\mathsf{CH}} \pmod q$.

We stress that the signer $\mathsf{id_C}$ can pre-compute $M \cdot sk_{\mathsf{CH}}$ and store it instead of M. Due to the modification in this scenario, we have the following major performance optimization:

- We reduce one big-number division (comparing to [15]) due to our modification on the chameleon hash evaluation algorithm. The improvement is significant for a resource-constrained device since the cost of a big-number modular division is close to a hash operation.

Lemma 5.1. *The modified chameleon hash function* CHF *is secure if the discrete logarithm problem is hard relative to* $\mathbb{Z}_p$.

Proof. Comparing with the algorithm in [15], we only switched the places of the message m and the randomness r. If the adversary can output two message/randomness pairs (m, r) and (m', r') that lead to the same hash value, then we can use them to solve the discrete logarithm problem, i.e., $sk_{\mathsf{CH}} := \frac{r-r'}{m'-m} \pmod q$.

6 VERIFICATION KEY REPLENISHMENT

Obviously, one limitation of the pre-computation strategy is that the pre-computed verification keys will be used up eventually. To overcome this limitation and support unlimited message authentication, we need to design a mechanism to re-initialize the verification key. We stress that it is sufficient to only refresh the verification key without modifying the secret/public key pair $(pk_{\mathsf{CH}}, sk_{\mathsf{CH}})$, so the services running on the signer will not be interrupted at all. The first naïve solution is to let the signer initialize a new Bloom filter instance BF′ with ℓ chameleon hash values which are generated based on the initial seed r'_ℓ and M as in $\mathsf{LiS_1}$.KGen. To this end, it has to send BF′ to the verifiers together with a signature that can be verified by the current verification key. However, this solution requires the signer to run the expensive key generation algorithm, and thus, it is not favorable in practice.

To free the signer from updating the verification key, we develop two solutions as follows.

Server-aided Replenishment (SAR): The signer can outsource the re-initialization job of the new Bloom filter instance BF' to a trusted server (which is not the verifier). Then, the outsourcing server who knows the dummy randomness/message pair $(r', pk_{\mathsf{CH}}^{M})$ and the key k of the universal hash function can compute those chameleon hash values for the signer without any interaction. The signer does not need to get involved in the verification key update, and it can keep using its signing key to sign future messages continuously. The outsourcing server (the key generation center) only needs to periodically publish a new BF' together with the server's signature to a public bulletin, which can be downloaded by the public. Nothing needs to be changed on the signer side. Hence, the signer and the verifier can run in parallel as long as the replenishment of the verification key is in time before the old verification key becomes invalid. The system of SAR is depicted in Figure 6.

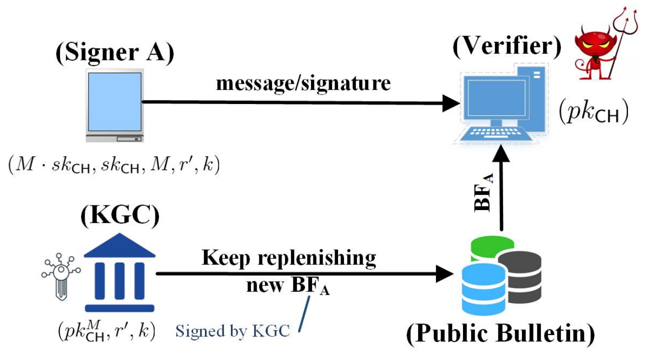
(Signer A)
message/signature
(Verifier)
(pk_{CH})
$(M \cdot sk_{\mathsf{CH}}, sk_{\mathsf{CH}}, M, r', k)$
$\mathsf{BF_A}$
(KGC)
Keep replenishing new $\mathsf{BF_A}$
$(pk_{\mathsf{CH}}^{M}, r', k)$
Signed by KGC
(Public Bulletin)

Figure 6: System Overview of Server-aided Replenishment (SAR). Verifiers can be malicious. The singer and the trusted key generation center (KGC) do not need any interactions during the replenishment procedure.

Verifier Self-replenishment (VSR): If the verifier is trustworthy (not controlled or compromised by an adversary), then we can allow the verifier to possess $(pk_{\mathsf{CH}}^{M}, r', k)$ for signature verification. In this way, the verifier can replenish its own verification keys regularly. In particular, in a cyber-physical system, it is common that the messages are sent on a regular basis with a fixed time period. Thus, we can actually exploit this fact and develop a simplified verification algorithm. In the model, we can consider all messages are associated with a monotonically increasing time-stamp, and we simplify the verification algorithm (to reduce the storage cost) to enable the verifier to have a small and constant storage cost. Besides, we use T_m to denote a time-stamp that is in the message m, and let T_l be the time when the last valid signature is received. Δ_s stands for the fixed time slot between two consecutive messages sent from the signer. We show the system overview and the modified algorithm of VSR in Figure 7 and 8. Note that the Bloom filter is not needed in both Setup and Verify algorithms anymore, and thus the size of the verification key does not depend on ℓ and becomes a constant. Furthermore, to modify LiS_2, we require the signer to include the counter cnt as part of the message, and it computes $h_1(m||N||cnt)$ in the Sign algorithm. Since r' or k needs to be kept secret, the modified algorithms will not be able to provide public verifiability, and it can only be verified by a group of trusted verifiers.

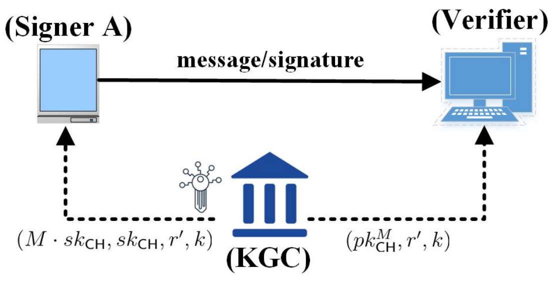
(Signer A)
message/signature
(Verifier)
$(M \cdot sk_{\mathsf{CH}}, sk_{\mathsf{CH}}, r', k)$
(KGC)
$(pk_{\mathsf{CH}}^{M}, r', k)$

Figure 7: Overview of Verifier Self-replenishment (VSR). Verifiers should be trusted. KGC only needs to send the tuple $(M \cdot sk_{\mathsf{CH}}, sk_{\mathsf{CH}}, r', k)$ to the signer and the tuple $(pk_{\mathsf{CH}}^{M}, r', k)$ to verifier only once respectively after key generation. $r' = \emptyset$ in LiS_2.

The modified algorithm well fits a cyber-physical system scenario (e.g., smart grid and manufacturing systems) where the verifier needs to continuously monitor the status (and data) of the signer (e.g., a sensor), and the verifiers are only a few pre-known and trusted machines. We stress that in this scenario, ℓ can be considered as the maximum number of signature failures (including signature loss and signature verification fails) that the verifier can tolerate between the last valid time T_l and the current time T_c. For a real-time monitoring system in a CPS, ℓ should be small. Note that we modified the KGen algorithm to let the KGC and the verifier store pk_{CH}^{M} instead of pk_{CH} for both security and efficiency reasons. This change can hide the value of M from the adversary, and therefore, an adversary who compromised the KGC or the verifier cannot extract the secret key sk_{CH} with knowing M.

Remarks. The first replenishment solution SAR is more appealing and practical than the naïve solution since it does not need to interact with the signer for replenishment. For example, a maritime transport company can periodically replenish the verification keys for ships in the sea every day. The second replenishment solution VSR can be used when the signature schemes are deployed within a factory or enterprise, which has trustworthy verifiers and does not need public verifiability.

Based on the above replenishment scenarios, the message authentication power of the signer in our signature schemes can be unlimited. Besides, due to this replenishment property, we can use a smaller ℓ to reduce the size of the verification key.

7 APPLICATIONS

LiS can be widely used in cyber-physical systems. It enables resource-constrained signers to (continuously) authenticate messages. In the following, we just name a few examples of suitable applications.

Satellite Navigation Systems. Satellite-dependent positioning systems, such as GPS and GNSS, can provide navigation and time synchronization features, which have a significant impact on daily life. The users of such systems just periodically (e.g., every 1 second) receive signals and navigation messages from the satellites, without transmitting any data back to the satellites. Such a widely used and convenient system becomes an attractive target of various attackers. For example, by launching a spoofing attack [24, 25], attackers can effectively coerce GPS/GNSS receivers into concluding false

```
$\mathsf{LiS}_1^{\mathsf{VSR}}$.Verify($vk_{\mathrm{id_C}}, m, x$):
$(k, pk_{\mathrm{CH}}^M, T_l) = vk_{\mathrm{id_C}}$
$r' \leftarrow$ private storage
If $T_m < T_l$: OUTPUT 0
$r'_0 := r'$; $vr := 0$; $t := \mathsf{CHF}(m, x)$
$\ell := \lfloor \frac{T_m - T_l}{\Delta_s} \rfloor$
For $i \in [\ell]$:
    $r'_i := \mathsf{UHF}(k, r'_{i-1})$
$t_\ell := pk_{\mathrm{CH}}^M \cdot g^{r'_\ell}$;
If $t_\ell = t$:
    $vr := 1$; $r' := r'_\ell$; $T_l := T_m$
OUTPUT $vr$
```

```
$\mathsf{LiS}_2^{\mathsf{VSR}}$.Verify($vk_{\mathrm{id_C}}, (m, N, cnt), x$):
$(pk_{\mathrm{CH}}^M, T_l) = vk_{\mathrm{id_C}}$
$k \leftarrow$ private storage
If $T_m < T_l$ : OUTPUT 0
$r'_0 := r'$; $vr := 0$; $t := \mathsf{CHF}(h_1(m||N||cnt), x)$
$r'_{cnt} := h_2(k||cnt)$
$t_{cnt} := pk_{\mathrm{CH}}^M \cdot g^{r'_{cnt}}$;
If $t_{cnt} = t$:
    $vr := 1$; $T_l := T_m$
OUTPUT $vr$
```

Figure 8: The Modified Verification Algorithms.

location and navigation solutions. To prevent spoofing attacks, LiS_1 can be used to authenticate GPS/GNSS signals and messages. In this application, the verification key of the satellite can be refreshed using server-aided replenishment (SAR). For example, the satellite control center on the ground can periodically (say every week) publish a new Bloom Filter instance for the GPS/GNSS devices on the ground to verify the signatures.

Maritime Systems. One of the most important systems used on ships is the automatic identification system (AIS). It can show some information about vessels such as unique identification, position, course, and speed. AIS is intended to assist watch-standing officers on a vessel and allow maritime authorities to track and monitor vessel movements. The AIS information provided by those vessels is the primary method of collision avoidance for water transports. In practice, we need a very accurate position of a ship when it is in busy waters and harbors. Our LiS_2 can be a solution to broadcast authenticated AIS information to different verifiers, such as other vessels, lighthouses, or buoys, at a fast speed as required in the standard (i.e., $< 27ms$).

Critical Infrastructures. Our daily life may become a dreadful mess when we do not have various critical infrastructures like smart grids, water plants, and transportation systems. At the core of these systems, usually Programmable Logic Controllers (PLCs) are used to control the physical processes directly. To make sure that the PLCs are running correctly and are controlling the processes, supervisory control and data acquisition (SCADA) systems in a control center monitor the processes remotely in real-time. It is, therefore, crucial for SCADA to verify the authenticity of the data obtained from PLCs. One can integrate $\mathsf{LiS}_1/\mathsf{LiS}_2$ into the firmware of PLCs and authenticate each data sent to the SCADA. Depending on whether the inputs of the signer can be adaptively affected by the adversaries, one can choose to use LiS_1 or LiS_2. Apparently, $\mathsf{LiS}_1/\mathsf{LiS}_2$ can be applied to sensors as well, because, as its nature, a sensor continuously reports its measurement to a server. In this case, having a lightweight signature integrated will prove the authenticity of the sensor data to the server so that this can prevent sensor data injection attacks via digital channels [5, 26]. In the meantime, with our VSR replenishment scheme, the verification

Table 2: The Runtime of Sign and Verify

	Sign	Verify
$\mathsf{LiS}_1^{\mathsf{SAR}}$	$7.32\mu s$	0.59 ms
$\mathsf{LiS}_1^{\mathsf{VSR}}$	$7.32\mu s$	Figure 9
$\mathsf{LiS}_2^{\mathsf{SAR}}$	$11.06\mu s$	0.59 ms
$\mathsf{LiS}_2^{\mathsf{VSR}}$	$11.06\mu s$	0.35 ms

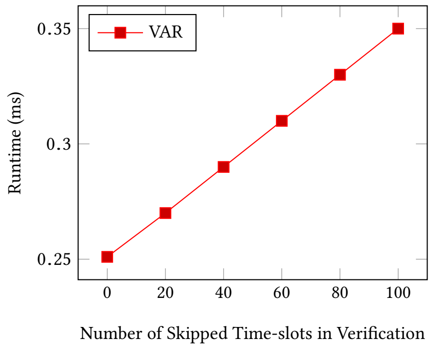

Figure 9: Runtime of LiS_1.Verify.

keys of PLCs can be easily replenished by a trusted SCADAs (or other local servers) themselves.

8 COMPARISON AND BENCHMARK

Comparison. Here we compare the security features and performance between our signature schemes and some related works, including Schnorr [9], Yao and Zhao's Γ-1 and Γ-2 [12], and SEMECS [13]. Let 'CMAu' and 'PV' denote continuous message authentication and public verifiability, respectively. And let 'SKR' denote signer key replenishment (i.e., whether it needs to replenish signing key), and 'VKR' denote verifier key replenishment. Note that our concrete signature schemes LiS_1 and LiS_2 can have further variants with different verification key replenishment solutions. So we use

Table 3: Comparison

	Properties					Key Size		KGen	Computation Cost		Signature Size
	Security	SKR	VKR	CMAu	PV	Signer	Verifier		Signer	Verifier	
Schnorr [9]	SEUF-CMA	×	×	√	√	$1\|\mathbb{Z}_q\|$	$1\|\mathbb{Z}_p\|$	1E	1A+1M+1E+1H	2E+1H	$1\|\mathbb{Z}_q\|+1\|\mathbb{Z}_p\|$
Γ-1 [12]	SEUF-CMA	√	×	×	√	$2\ell \cdot \|\mathbb{Z}_p\|$	$(\ell+1) \cdot \|\mathbb{Z}_p\|$	$\ell \cdot (1E+1M+1H)$	1M+1A+1H	1E+1M+1H	$1\|\mathbb{Z}_q\|+1\|\mathbb{Z}_p\|$
Γ-2 [12]	SEUF-CMA	√	×	×	√	$3\ell \cdot \|\mathbb{Z}_p\|$	$(2\ell+1) \cdot \|\mathbb{Z}_p\|$	$\ell \cdot (1E+1M+1H)$	1M+1A+1H	1E+1M+1H	$1\|\mathbb{Z}_q\|+1\|\mathbb{Z}_p\|$
SEMECS [13]	SEUF-CMA	×	√	×	√	$\|\mathbb{Z}_q\|$	$(2\ell+1)\|\mathbb{Z}_q\|$	$\ell(4H+1E)$	1A+1M+3H	2E+3H	$2\|\mathbb{Z}_q\|$
$\mathsf{LiS}_1^{\mathrm{SAR}}$	SEUF-wCMA	×	√	√	√	$5\|\mathbb{Z}_q\|$	$1.44\ell \cdot \epsilon+1\|\mathbb{Z}_p\|$	ℓ(2E+1BF+1U)+1E	3A+2M	2E+1BF	$1\|\mathbb{Z}_q\|$
$\mathsf{LiS}_1^{\mathrm{VSR}}$	SEUF-wCMA	×	×	√	×	$5\|\mathbb{Z}_p\|$	$1\|\mathbb{Z}_p\|+3\|\mathbb{Z}_q\|$	3E+1U	3A+2M + ℓ·U	2E+ 1H	$1\|\mathbb{Z}_q\|$
$\mathsf{LiS}_2^{\mathrm{SAR}}$	SEUF-CMA	×	√	√	√	$4\|\mathbb{Z}_q\|$	$1.44\ell \cdot \epsilon+1\|\mathbb{Z}_p\|$	ℓ(2E+1BF+1H)	3A+2M+2H	2E+1H+1BF	$1\|\mathbb{Z}_q\| + \ell_r$
$\mathsf{LiS}_2^{\mathrm{VSR}}$	SEUF-CMA	×	×	√	×	$4\|\mathbb{Z}_p\|$	$1\|\mathbb{Z}_p\|+3\|\mathbb{Z}_q\|$	3E	3A+2M+2H	2E+2H	$1\|\mathbb{Z}_q\| + \ell_r$

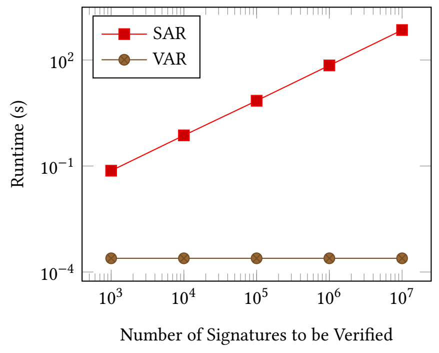

Figure 10: Runtime of KGen.

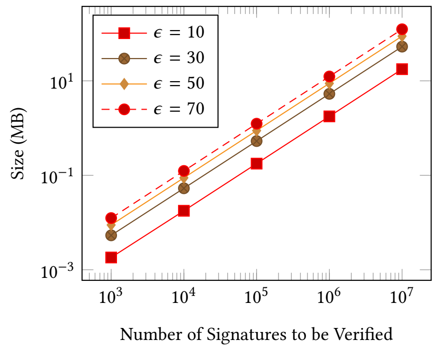

Figure 11: Size of Verification Key of LiS **with SAR.**

the superscripts 'SAR' and 'VSR' to indicate them, respectively. We use 'A', 'M', 'E', 'I', 'U', and 'H' to denote addition, scalar multiplication, exponentiation, inversion, universal hash function, and hash function, respectively. 'SEUF-(w)CMA' stands for strong existential unforgeability (weak) chosen message attacks.

The comparison is presented in Table 3. Comparing with Yao and Zhao's online/offline signatures, our schemes have a much shorter key size for either the signer or the verifier. Besides, our schemes do not need to replenish the signing keys. Although our signing algorithm is slightly more expensive than the online/offline signatures, our signature schemes still have a practical performance. Also, compared with Schnorr's signature, our signing algorithm is more efficient as Schnorr's signature requires modular exponentiation, which does not fit any resource-constrained devices in CPS [27]. Although our schemes' the key size of the signer is slightly longer than that of SEMECS [13], we outperform SEMECS with respect to the other performance metrics.

Our $\mathsf{LiS}_1^{\mathrm{VSR}}$ and $\mathsf{LiS}_2^{\mathrm{VSR}}$ have near optimal storage costs (implied by the optimal key size) for both signer and verifier so that they well fit cyber-physical systems (such as smart grids) which do not require public verifiability.

Implementations and Evaluations. In this section, we show the practicality of our proposed family of lightweight signature schemes $\mathsf{LiS} = (\mathsf{LiS}_1, \mathsf{LiS}_2)$. All benchmark results reported in Table 2 were obtained on an Intel Core i7-4770K from the server side, and a Raspberry Pi 3 from the client side. The operating system of the server is Ubuntu 16.04, which runs in a VMWare virtual machine, and only one core of the CPU is used. Note that we set $|p| = 1024$ and $|q| = 320$, and use SHA2 [22] to implement the hash function h in LiS_2. The implementation of SHA2 is taken from the highly optimized MIRACL library [28]. We benchmarked the Bloom filter with different error parameters (i.e., $10^{-3}, 10^{-6}, 10^{-9}, 10^{-12}$), and they resulted in similar performance. Hence, we fix the parameter 10^{-9} (as an example) to show the computational cost of our signature schemes.

The performance of the Sign and Verify algorithms of LiS_1 are presented in Table 2 and Figure 9. Notice that in LiS_1^{VSR}, if the signer does not send signatures in a few time slots, then the verifier will have to fast forward the universal hash function to skip these time slots and generate the correct dummy randomness for next signature verification (see Figure 8 for details). The size of hash input is 1KB for testing[3]. We stress that hashing 1KB only takes 0.32 microseconds (μs) on the server (which is much faster than other operations such as modular exponentiation), so we will omit it while calculating the performance on the server side. The performances of the KGen algorithms are shown in Figure 10. Also, the size of verification keys is shown in Figure 11 with various parameters.

From our benchmark results, we can see that the Sign algorithm is very efficient for the embedded devices since it only takes 7.32 or 11.06 milliseconds. And the Verify algorithm is practical as well, such that it will not slow the processes in a CPS. On the one hand, the cost of KGen and the size of the verification key for SAR are linear with the number of signatures to be verified. For 10 million

[3]The cost of the hash function is linear with the size of the hashing message. Here we just compute the hash value of 1KB as a reference.

signatures, the verification key is about 10 MB, which is acceptable for most of the verifiers (even if it is a modern smartphone). On the other hand, it takes about four months to generate 10 million signatures for authentication if we assume the signature generation time intervals are 1s. Similarly, it will also take four months for the verifier to consume all the signatures in the Bloom filter, so we have more than enough time to replenish the next Bloom filter in practice. In the case of VSR, the verification key is just a constant with dozens of bytes.

9 RELATED WORK

Broadcast Authentication. A research topic that is related to our work is broadcast authentication (BA), which is widely used to authenticate broadcast messages from resource-constrained devices. One of the main motivations of BA is to authenticate the timely delivery of messages, so most of the existing BA protocols are time-constrained. A typical example of BA is the standardized protocol called Timed Efficient Stream Loss-Tolerant Authentication (TESLA) [29] for broadcast authentication in wireless networks. TESLA leverages on a symmetric message authentication code (MAC) mechanism for authentication and a time-based chain structure to pre-compute the secrets of MAC disclosed later. Whereas, the message and its corresponding MAC is sent to the receiver immediately upon creation, while the secret key is dispatched after a pre-determined interval of time. However, a drawback of TESLA and its variants [30–32] is the usage of symmetric key cryptographic technique (i.e., MAC) so that the corruption of either key share would affect the security of the whole cryptosystem. To overcome such a drawback of TESLA, researchers [33–35] appeal to lightweight public key based cryptographic building blocks (such as (one-time) digital signature) to construct BA protocols. Recently, Afianti et al. [36] proposed a BA scheme which mixed many cryptographic methods including signature, MAC, encryption, and a new dynamic cipher puzzle scheme. However, the above public key based solutions need either ecliptic-curve multiplications or many cryptographic hash operations that are too heavy for resource-constrained devices. We refer the reader to [37] for more BA protocols. In this work, we focus on designing a lightweight signature scheme that has an optimized lightweight signing procedure.

Online/Offline Signature. To enable the digital signature on a resource-constrained device, the online/offline signature scheme was invented [10]. The idea of such a scheme is to run those expensive public-key cryptographic operations (such as the exponentiation) in a signature scheme at the offline phase and pre-store the intermediate private data on the device, so that in the online phase the signer can sign a message in very fast speed with a few cheap arithmetic operations. Hence, the online/offline signatures are suitable for many applications where the signer (e.g., sensors or RFID) has very limited computational resources. In 2013, Yao and Zhao proposed a variant of Fiat-Shamir paradigm [38] called Γ-transformation that can transform Fiat-Shamir style signature schemes (such as Schnorr [9]) into efficient online/offline signature schemes. The security of their scheme is further studied in [39]. Moreover, there are also some variants [40–42] which are proposed in the identity-based setting. However, a shortcoming of online/offline signatures is that they require the signer to pre-store a non-trivial amount of intermediate private data, and this data needs to be replenished when it is used out. Thus, it does not fit cyber-physical systems that have a high demand for continuous message authentication without interruptions. In contrast, we are interested in the lightweight signature schemes that have a small constant-sized storage cost for the signer without any need for replenishment.

In 2019, Yavuz and Ozmen [13] proposed a lightweight signature scheme called SEMECS, which is adapted from the Schnorr signature. To facilitate the signature generation, SEMECS particularly chain up the exponents of verification keys, such that $r_j := H(r_{j-1})$, where the j-th verification key is computed as g^{r_j} and g is the group generator of a cyclic abelian group, so that it can pre-compute all verification keys in advance. To sign a message m_j, the signer only needs to compute the randomness of the Schnorr signature by using one multiplication and one subtraction, i.e., $s_j := r_j - m_j \cdot sk$, where sk is the signing key. However, the SEMECS does not consider the verification key replenishment problem like our schemes. Hence, it is not suitable for continuous message authentication. Note that it is not allowed to outsource the verification key replenishment procedure to a third party or the verifier by handing over the seed r_{j-1}. Once the attackers get r_j and r_{j-1}, and the corresponding messages (m_{j-1}, m_j), it can trivially extract the signing key. Our scheme does not have this problem since we only chain up the randomness.

Message Authentication using Chameleon Hash. Chameleon hash functions (CHF) [15] are not only probabilistic (randomized) collision-resistant but also adaptable with a trapdoor (i.e., one can efficiently generate hash collisions based on the given trapdoor). CHF is widely used as a building block in various cryptographic primitives and protocols. In 2010, Mohassel[43] proposed a general construction for transforming any chameleon hash function to a strongly unforgeable one-time signature scheme. But, unlike ours, it did not build a regular (multiple-times) signature scheme for continuous message authentication in cyber-physical systems. CHF is also used in many real-world application such as the vehicular communications [44], verifiable data streaming [45], and rewritable block-chains [46]. However, all these constructions require the (message) sender to compute the hash value of the CHF, which involves two expensive exponentiation operations. In contrast, we use the chameleon hash function in a different way by leveraging the collision generation function to generate the signatures. Therefore our signing procedure is much more efficient.

10 CONCLUSIONS

In this work, we proposed a signature framework called LiS, which consists of two concrete signature schemes based on chameleon hash functions. The proposed signature schemes have very fast signing algorithms and require a small constant storage cost on the signing device, so they are particularly suitable for cyber-physical systems with continuous message authentication services. We also implemented our schemes on embedded devices to show the practicality of our schemes. In contrast to prior work (especially online/offline signatures), our schemes have much smaller storage cost, and the signing key does not need to be replenished during its whole lifespan. In addition, we discuss the potential applications of our signature schemes in specific types of cyber-physical systems.

An open problem remaining is how to modify LiS further to provide public verifiability without using SAR. One may check whether the state-of-the-art cryptographic technique: indistinguishability obfuscators (IO) [47, 48] can work in this scenario. For instance, one may apply IO to obfuscate our modified verification algorithms to hide the dummy randomness r'.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their helpful comments. This research is supported by A*STAR under its RIE2020 Advanced Manufacturing and Engineering (AME) Industry Alignment Fund - Pre Positioning (IAF-PP) Award A19D6a0053. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of A*STAR. This was partially supported by the Natural Science Foundation of China under Grant No. 61872051 and 61402085. Chenglu Jin's work was done when he was with the University of Connecticut and was supported by AFOSR MURI under award number FA9550-14-1-0351.

REFERENCES

[1] Nicolas Falliere, Liam O Murchu, and Eric Chien. 2011. W32. stuxnet dossier. *White paper, Symantec Corp., Security Response* 5, 6 (2011), 29.
[2] Eduard Kovacs. [n.d.]. Attackers Alter Water Treatment Systems in Utility Hack: Report. https://www.securityweek.com/attackers-alter-water-treatment-systems-utility-hack-report.
[3] Chuadhry Mujeeb Ahmed, Jianying Zhou, Aditya Mathur, Rizwan Qadeer, Reza Rahaeimehr, Carlos Murguia, and Justin Ruths. 2018. NoisePrint: Attack Detection Using Sensor and Process Noise Fingerprint in Cyber Physical Systems. In AsiaCCS. Springer, 483–497.
[4] Nils Ole Tippenhauer, Christina Pöpper, Kasper Bonne Rasmussen, and Srdjan Capkun. 2011. On the requirements for successful GPS spoofing attacks. In CCS. ACM, 75–86.
[5] Yao Liu, Peng Ning, and Michael K Reiter. 2011. False data injection attacks against state estimation in electric power grids. *ACM Transactions on Information and System Security (TISSEC)* 14, 1 (2011), 13.
[6] Marten van Dijk, Chenglu Jin, Hoda Maleki, Phuong Ha Nguyen, and Reza Rahaeimehr. 2018. Weak-Unforgeable Tags for Secure Supply Chain Management. In FC. Springer, 80-98.
[7] Taher A El Gamal. 1984. *Cryptography and logarithms over finite fields.* Ph.D. Dissertation. UMI.
[8] Nils Fleischhacker, Tibor Jager, and Dominique Schröder. 2019. On Tight Security Proofs for Schnorr Signatures. *J. Cryptology* 32, 2 (2019), 566–599.
[9] Claus-Peter Schnorr. 1989. Efficient Identification and Signatures for Smart Cards. In *CRYPTO*. Springer, 239–252.
[10] Shimon Even, Oded Goldreich, and Silvio Micali. 1996. On-Line/Off-Line Digital Signatures. *J. Cryptology* 9, 1 (1996), 35–67.
[11] Adi Shamir and Yael Tauman. 2001. Improved Online/Offline Signature Schemes. In *CRYPTO*. Springer, 355–367.
[12] Andrew Chi-Chih Yao and Yunlei Zhao. 2013. Online/Offline Signatures for Low-Power Devices. *IEEE Trans. Information Forensics and Security* 8, 2 (2013), 283–294.
[13] A. A. Yavuz and M. O. Ozmen. 2019. Ultra Lightweight Multiple-time Digital Signature for the Internet of Things Devices. *IEEE Transactions on Services Computing* (2019), 1–1. https://doi.org/10.1109/TSC.2019.2928303
[14] [n.d.]. Automatic identification system. https://en.wikipedia.org/wiki/\Automatic_identification_system. Accessed: 2019-07-25.
[15] Hugo Krawczyk and Tal Rabin. 2000. Chameleon Signatures. In *NDSS*. The Internet Society.
[16] Burton H Bloom. 1970. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM* 13, 7 (1970), 422-426.
[17] Larry Carter and Mark N. Wegman. 1977. Universal Classes of Hash Functions (Extended Abstract). In *STOC*. ACM, 106–112.
[18] Anna Pagh, Rasmus Pagh, and S. Srinivasa Rao. 2005. An optimal Bloom filter replacement. In *SODA*. SIAM, 823–829.
[19] Mihir Bellare and Phillip Rogaway. 1994. Optimal Asymmetric Encryption. In *EUROCRYPT*. Springer, 92–111.
[20] Dan Boneh and Xavier Boyen. 2004. Short Signatures Without Random Oracles. In *EUROCRYPT*. Springer, 56–73.
[21] Victor Shoup. 2004. Sequences of games: a tool for taming complexity in security proofs. *IACR Cryptology ePrint Archive* 2004 (2004), 332.
[22] PUB FIPS. 2012. 180-4. *Secure hash standard (SHS),âĂİ March* (2012).
[23] Martin Dietzfelbinger, Torben Hagerup, Jyrki Katajainen, and Martti Penttonen. 1997. A Reliable Randomized Algorithm for the Closest-Pair Problem. *J. Algorithms* 25, 1 (1997), 19–51.
[24] A. Jafarnia-Jahromi, A. Broumandan, J. Nielsen, and G. Lachapelle. 2012. GPS vulnerability to spoofing threats and a review of Antispoofing techniques. *Int J Navigation and Observation* 2012 (2012).
[25] Eric Horton and Prakash Ranganathan. 2018. Development of a GPS spoofing apparatus to attack a DJI Matrice 100 Quadcopter. *The Journal of Global Positioning Systems* 16, 1 (03 Jul 2018), 9.
[26] Gyorgy Dan and Henrik Sandberg. 2010. Stealth attacks and protection schemes for state estimators in power systems. In *SmartGridComm*. IEEE, 214–219.
[27] Emiliano Sisinni, Abusayeed Saifullah, Song Han, Ulf Jennehag, and Mikael Gidlund. 2018. Industrial internet of things: Challenges, opportunities, and directions. *IEEE Transactions on Industrial Informatics* 14, 11 (2018), 4724–4734.
[28] Miracl 2018. MIRACL Cryptographic Library. https://bit.ly/2MltKVG
[29] Adrian Perrig, Ran Canetti, Dawn Song, Professor Doug Tygar, and Bob Briscoe. 2005. Timed Efficient Stream Loss-Tolerant Authentication (TESLA): Multicast Source Authentication Transform Introduction. RFC 4082. https://doi.org/10.17487/RFC4082
[30] Adrian Perrig, Ran Canetti, Dawn Xiaodong Song, and J. D. Tygar. 2001. Efficient and Secure Source Authentication for Multicast. In *NDSS*. The Internet Society.
[31] Adrian Perrig, Robert Szewczyk, J. D. Tygar, Victor Wen, and David E. Culler. 2002. SPINS: Security Protocols for Sensor Networks. *Wireless Networks* 8, 5 (2002), 521–534.
[32] Donggang Liu and Peng Ning. 2004. Multilevel μTESLA: Broadcast authentication for distributed sensor networks. *ACM Trans. Embedded Comput. Syst.* 3, 4 (2004), 800–836.
[33] Adrian Perrig. 2001. The BiBa one-time signature and broadcast authentication protocol. In *CCS*. ACM, 28–37.
[34] Shang-Ming Chang, Shiuhpyng Shieh, Warren W. Lin, and Chih-Ming Hsieh. 2006. An efficient broadcast authentication scheme in wireless sensor networks. In *AsiaCCS*. ACM, 311–320.
[35] Jaeheung Lee, Seokhyun Kim, Yookun Cho, Yoojin Chung, and Yongsu Park. 2012. HORSIC: An efficient one-time signature scheme for wireless sensor networks. *Inf. Process. Lett.* 112, 20 (2012), 783–787.
[36] Farah Afianti, Wirawan, and Titiek Suryani. 2018. Dynamic Cipher Puzzle for Efficient Broadcast Authentication in Wireless Sensor Networks. *Sensors* 18, 11 (2018), 4021.
[37] Kanika Grover and Alvin S. Lim. 2015. A survey of broadcast authentication schemes for wireless networks. *Ad Hoc Networks* 24 (2015), 288–316.
[38] Amos Fiat and Adi Shamir. 1986. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *CRYPTO*. Springer, 186–194.
[39] Özgür Dagdelen, David Galindo, Pascal Véron, Sidi Mohamed El Yousfi Alaoui, and Pierre-Louis Cayrel. 2016. Extended security arguments for signature schemes. *Des. Codes Cryptogr.* 78, 2 (2016), 441–461.
[40] Joseph K. Liu, Joonsang Baek, Jianying Zhou, and Jun Wen Wong. 2010. Efficient online/offline identity-based signature for wireless sensor network. *Int. J. Inf. Sec.* 9, 4 (2010), 287–296.
[41] Ya Gao, Peng Zeng, Kim-Kwang Raymond Choo, and Fu Song. 2016. An Improved Online/Offline Identity-Based Signature Scheme for WSNs. *I. J. Network Security* 18, 6 (2016), 1143–1151.
[42] Chandrashekhar Meshram, Chun-Ta Li, and Sarita Gajbhiye Meshram. 2019. An efficient online/offline ID-based short signature procedure using extended chaotic maps. *Soft Comput.* 23, 3 (2019), 747–753.
[43] Payman Mohassel. 2010. One-Time Signatures and Chameleon Hash Functions. In *SAC*. Springer, 302–319.
[44] Song Guo, Deze Zeng, and Yang Xiang. 2014. Chameleon Hashing for Secure and Privacy-Preserving Vehicular Communications. *IEEE Trans. Parallel Distrib. Syst.* 25, 11 (2014), 2794–2803.
[45] Johannes Krupp, Dominique Schröder, Mark Simkin, Dario Fiore, Giuseppe Ateniese, and Stefan Nürnberger. 2016. Nearly Optimal Verifiable Data Streaming. In *PKC*. Springer, 417–445.
[46] David Derler, Kai Samelin, Daniel Slamanig, and Christoph Striecks. 2019. Fine-Grained and Controlled Rewriting in Blockchains: Chameleon-Hashing Gone Attribute-Based. In *NDSS*. The Internet Society.
[47] Nir Bitansky and Vinod Vaikuntanathan. 2018. Indistinguishability Obfuscation from Functional Encryption. *J. ACM* 65, 6 (2018), 39:1–39:37.
[48] Shweta Agrawal. 2019. Indistinguishability Obfuscation Without Multilinear Maps: New Methods for Bootstrapping and Instantiation. In *EUROCRYPT (1)*. Springer, 191–225.
[49] Sven Schäge. 2011. Tight Proofs for Signature Schemes without Random Oracles. In *EUROCRYPT*. Springer, 189–206.

A THE ORIGINAL DISCRETE LOGARITHM BASED CHAMELEON

In the following, we briefly review the discrete logarithm based chameleon hash function of [15].

- $\mathsf{CHKGen}(1^\kappa)$: The key generation algorithm samples random group generator $g \xleftarrow{\$} \mathbb{Z}_q$ and a secret key $sk_{\mathsf{CH}} \xleftarrow{\$} \mathbb{Z}_q$, and computes the public key $pk_{\mathsf{CH}} := g^{sk_{\mathsf{CH}}} \pmod p$.
- $\mathsf{CHF}(pk_{\mathsf{CH}}, m, r)$: The evaluation algorithm takes as input a public key $pk_{\mathsf{CH}} \in \mathbb{Z}_p^*$, a message $m \in \mathbb{Z}_q^*$ and a randomness $r \in \mathbb{Z}_q^*$, and outputs a hash value $y := g^m pk_{\mathsf{CH}}^r \pmod p$.
- $\mathsf{CHColl}(sk_{\mathsf{CH}}, r', m, m')$: An efficient deterministic collision algorithm CHColl takes as input the secret key sk_{CH}, and $(r', m', m) \in \mathbb{Z}_q^*$, outputs a value $r' := \frac{m + r \cdot sk_{\mathsf{CH}} - m'}{sk_{\mathsf{CH}}} \pmod q$.

B PROOF OF THEOREM 4.1

Let BK_i denote an event that there exists an adversary $\mathcal{A}$ wins in Game i. Note that BK_0 is the advantage of $\mathcal{A}$ breaking the scheme in the real game. In the following, we will change the games starting from the real game until the last game in which the advantage of $\mathcal{A}$ is zero.

Game 0. This game equals to the real SEUF-wCMA security experiment of the digital signature scheme. Meanwhile, all queries are answered honestly according to the specification of LiS_1. Thus, we have that

$$\Pr[\mathsf{BK}_0] = \mathsf{Adv}_0 = \mathsf{Adv}^{\mathsf{SIG}}_{\mathcal{A},\mathsf{LiS}_1}(\kappa, \ell).$$

Game 1. This game proceeds like before but the challenger uses the ℓ uniformly distributed dummy randomness $\{r'_i\}_{i\in[\ell]}$ to generate the verification key without using the universal hash function UHF. This modification actually does not change the distribution of the dummy randomness. Recall that the seed of UHF is a random value r_0, and the hash keys of UHF are random in our design as well. Thus, the outputs of UHF are distributed uniformly in terms of Definition 2.1. Hence, we can just randomly choose those dummy randomnesses $\{r'_i\}_{i\in[\ell]}$. Thus, we have that

$$\Pr[\mathsf{BK}_1] = \Pr[\mathsf{BK}_0].$$

Game 2. This game proceeds like before, but the challenger changes the game as follows. It chooses the randomness $x_i \xleftarrow{\$} \mathcal{R}_{\mathsf{CH}}$ for each message $m_i \in \overline{M}$ (submitted by $\mathcal{A}$) as the signature instead of running the collision generation function CHColl. The verification key is generated using the real signature and message pairs, i.e., $\{(x_i, m_i)\}_{i\in[\ell]}$. Note that $x_i := M \cdot sk_{\mathsf{CH}} + r'_i - m_i \cdot sk_{\mathsf{CH}} \pmod q$. We can rewrite x_i as $x_i := r'_i + \tilde{m}_i \pmod q$ where $\tilde{m}_i = M \cdot sk_{\mathsf{CH}} - m_i \cdot sk_{\mathsf{CH}}$. Since each m_i is unique, so is $\tilde{m}_i$. We claim that each x_i is statistically close to a uniform random value with distance 0. The proof of this claim could follow the proof of the combining functions of [49, Lemma 1]. Thus, we have that

$$\Pr[\mathsf{BK}_2] = \Pr[\mathsf{BK}_1].$$

Game 3. In this game, the challenger C proceeds exactly like the previous game but adds an abort rule. Namely, C aborts if the adversary submits a tuple (m^*, x^*) which leads to a collision to one of those hash values recorded in BF, i.e., $\mathsf{CHF}(m^*, x^*) = \mathsf{CHF}(M, r'_i)$ for some r'_i. If this case occurs with a non-negligible probability $\mathsf{Adv}^{\mathsf{CH}}_{\mathcal{A},\mathsf{CHF}}(\kappa)$, then we can construct an efficient algorithm $\mathcal{F}$ by using $\mathcal{A}$ to break the security of the chameleon hash function.

Specifically, $\mathcal{F}$ could simulate the signature game for $\mathcal{A}$ while receiving a challenge public key pk^*_{CH} from the chameleon hash challenger. However, without knowing the secret key sk^*_{CH}, $\mathcal{F}$ cannot compute the collisions online as done in LiS_1. Instead, $\mathcal{F}$ uses the real signatures randomly chosen as the previous game to generate the verification key rather than using dummy message/randomness pairs. This change is possible because of the modifications in the previous games.

Due to the security of chameleon hash function, we have that

$$\Pr[\mathsf{BK}_2] \leq \Pr[\mathsf{BK}_3] + \mathsf{Adv}^{\mathsf{CH}}_{\mathcal{A},\mathsf{CHF}}(\kappa).$$

Game 4. In this game, C proceeds as before, but aborts if the adversary $\mathcal{A}$ submits a tuple (m^*, x^*) such that $\mathsf{Check}(\mathsf{CHF}(m^*, x^*)) = 1$ and (m^*, x^*) has not been queried by the adversary before, i.e., $\mathcal{A}$ finds a false positive error of BF. By applying the false positive probability of BF, we have that

$$\Pr[\mathsf{BK}_3] \leq \Pr[\mathsf{BK}_4] + 2^{-\epsilon}.$$

In this game, if C does not abort then the Proc.Finalize query would always return 0. Thus, the advantage of $\mathcal{A}$ in this game is zero, i.e., $\Pr[\mathsf{BK}_4] = 0$.

Putting the probabilities altogether in the above games, we have the following probability:

$$\mathsf{Adv}_0 \leq \mathsf{Adv}^{\mathsf{CH}}_{\mathcal{A},\mathsf{CHF}}(\kappa) + 2^{-\epsilon}$$

, which is the result of Theorem 4.1.