Optimizing Proof of Aliveness in Cyber-Physical Systems

Zheng Yang, Chenglu Jin, Xuelian Cao, Marten van Dijk, *Fellow, IEEE,* and Jianying Zhou

Abstract—At ACSAC 2019, we introduced a new cryptographic primitive called proof of aliveness (PoA), allowing us to remotely and automatically track the running status (aliveness) of devices in the fields in cyber-physical systems. We proposed to use a one-way function (OWF) chain structure to build an efficient proof of aliveness, such that the prover sends every node on the OWF chain in a reverse order periodically, and it can be verified by a remote verifier with the possession of the tail node (last node) of the OWF chain. However, the practicality of this initial construction is limited by the finite number of nodes on an OWF chain. We enhance our first PoA construction by linking multiple OWF chains together using a pseudo-random generator chain in our second PoA scheme. This enhancement allows us to integrate one-time signature (OTS) schemes into the structure of the second construction to realize the auto-replenishment of the aliveness proofs. This implies that securely an initialized PoA instance can be used forever without interruption for reinitialization. In this work, our primary motivation is to further improve our secondary PoA and auto-replenishment schemes. Instead of storing the tail nodes of multiple OWF chains on the verifier side, we use a Bloom Filter to compress them. This saves 4.7 times the storage cost compared to our previous version at ACSAC 2019. Moreover, the OTS-based auto-replenishment solution cannot be applied to our first scheme solely based on OWFs, and it is not so efficient despite its standard model security. To overcome these limitations, we design a new auto-replenishment scheme from a hash-based commitment under the random oracle model in this work, which is much faster and can be used by both PoA schemes. Additionally, we implement and evaluate our PoA constructions on Raspberry Pis to demonstrate their performance. Considering the implementation on a storage/memory-constrained device, we particularly study the strategies for efficiently generating proofs.

Index Terms—Proof of Aliveness, One-Way Functions, One-Time Passwords, Auto Replenishment, Cyber-Physical Systems, CPS Security, Authentication

1 Introduction

Availability or aliveness are critical to cyber-physical systems (CPS), especially to the so-called critical infrastructures, which our daily life heavily relies on. For instance, hundreds of thousands of people had to stay in the dark for hours when the power grids were shut down by cyber attackers [2]. This kind of denial-of-service attack affects users and customers immediately, while another type of attacker may intend to disable certain services and hide their malicious behaviors from being detected. Triton, the most murderous malware in the world, attacked one petrol plant in Saudi Arabia and disabled its safety instrumentation systems (SIS) [3], [4]. If this malware was not discovered, the plant could keep running without the protection of its SIS and explode at some point. The latest news shows that Triton is still spreading and hit its second victim in 2019 [5], and nobody can tell how many plants are infected unknowingly. Thus, in the adversarial setting, knowing that a particular service is actually running is even more important than just running the service. To this end, we introduce a concept called "**Proof of Aliveness**" (PoA), which allows a device in the field to prove to the control center that it is still "alive" and running. More application scenarios of PoA are discussed in Section 8.

Obviously, the most straightforward construction of PoA is to send a heartbeat signal from the devices to the control center every few seconds. However, an attacker in the network can easily intercept the heartbeat signal and keep replaying it to fool the control center. Recent studies have highlighted that a more sophisticated attacker can even simulate the normal behavior of a compromised device and send the simulated data to deceive the control center while the actual device is not running at all [6]. Hence, it becomes imperative for the devices to send a proof message that can prove its authenticity. Meanwhile, the proof should be different every time to prevent replay attacks. For example, the prover can periodically send the current time T with its digital signature $Sig(T)$ or message authentication code $MAC(T)$ to a verifier as proof of aliveness. However, the devices in CPS are resource-constrained and consequently

- *Z. Yang is with Southwest University, Chongqing, China, 400715. E-mail: youngzheng@swu.edu.cn*
- *C. Jin is with Centrum Wiskunde & Informatica, Amsterdam, Netherlands, 1098 XG. E-mail: chenglu.jin@cwi.nl*
- *X. Cao is with Southwest University, Chongqing, China, 400715. E-mail: xueliancao7@email.swu.edu.cn*
- *M. van Dijk is with Centrum Wiskunde & Informatica, Amsterdam, Netherlands, 1098 XG, Vrije Universiteit, Amsterdam, Netherelands, 1081 HV, and University of Connecticut, Storrs, United States, CT 06269. E-mail: marten.van.dijk@cwi.nl*
- *J. Zhou is with the iTrust, Singapore University of Technology and Design, 8 Somapah Rd, Singapore, 487372. E-mail: jianying_zhou@sutd.edu.sg*

Chenglu Jin and Zheng Yang contribute equally and share the first authorship. Xuelian Cao is the corresponding author. This is an extended version of our paper published at ACSAC 2019 [1].

This work is supported by the Natural Science Foundation of China (Grant No. 62372386) and the Natural Science Foundation of Chongqing (Grant No. CSTB2022NSCQ-MSX0437). Jianying Zhou is 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.

cannot afford the computational cost of asymmetric key cryptography. Moreover, a MAC-based PoA scheme would require a verifier to possess a shared secret key with a client, making it vulnerable to a single point of failure. That is, if a verifier holds many shared keys from various clients, compromising the verifier would compromise the security of the entire protocol.

To construct a more efficient proof of aliveness (PoA) scheme resilient to verifier compromise, one can run lightweight authentication protocols between the device/client and the control center/server repeatedly [7], [8]. Among existing authentication schemes, we notice that time-based one-time passwords (TOTP) schemes [9], [10] share many similarities with proof of aliveness protocol, such as single side authentication, single communication pass, and the efficiency requirement. TOTP protocols are widely used as a second authentication factor to enhance the security of mobile user authentication, e.g., in Duo [11] and Google Authenticator [12]. During authentication, the prover/client sends a password generated based on a secret and the current time. After that, the password is sent to the verifier/server for verification using the present time and some pre-shared data. A recent advance and the state-of-the-art in the research of TOTP is a technique called T/Key published at CCS 2017 [10], which converts a hash chain-based authentication method [13], [14] into a time-based one-time password, such that the verifier does not need to store any secrets. It also implies that if we can construct a PoA construction similar to T/Key, then the PoA will be *publicly verifiable*. This technique removes a single point of failure in the system, such that an attacker cannot disable the verification services of the control center by just compromising a fraction of the servers.

Although these authentication-akin cryptographic primitives could be adapted to realize PoA, the security definition of PoA itself remains an open question. That is, we still need to formalize the unique (and essential) security features of PoA. For example, it is notable that a PoA scheme may primarily require the capacity to authenticate time-specific information without the need for session-aware entity authentication or the authentication of arbitrary messages (unlike other authentication schemes). Additionally, the procedures of aliveness attestation should continuously run through the whole life span of the client devices. These peculiarities of PoA motivate us to build new constructions that are efficient enough for resource-constrained devices and provably secure under a PoA security model.

Our Work. In this paper, we first introduce a formal security model for Proof of Aliveness. In the model, we assume adversaries can actively intercept and tamper with aliveness proofs, and take the effects of multiple clients into account, as it is common to use a single authentication server for multiple clients in practice.

In addition, we propose two PoA constructions. The first one is based on an N-node one-way function (OWF) chain using a random seed as its head node x_0, i.e., the i-th node in the chain is $x_i = f(x_{i-1})$ for $1 \leq i \leq N$, where $f(\cdot)$ is a one-way function. By revealing the nodes in the chain in reverse order (i.e., $x_{N-1}, x_{N-2}, \ldots, x_0$) as proofs at a constant pace, such as 30 seconds per node, the verifier with the possession of the tail node (x_N) of the chain can verify the correctness of each password x_i by taking it as input and repeatedly evaluating the OWF $f(\cdot)$ for $N - i$ times, and comparing with the stored tail node. This construction is similar to T/Key [10], but our OWF chain construction can be proved secure without random oracles. Although this construction is secure and very efficient in computation, an open question is how to enable it to run continuously for a long time, like a decade, in CPS. Note that the one-way function chain has only a finite length. Therefore, the proofs will be used up eventually at some point. Then, the PoA has to stop running and reinitialize itself, which is not preferable in CPS. Moreover, to facilitate the proof generation in this scheme, the client could cache a few internal nodes as the checkpoints during the setup phase, like the T/Key [10], so that the client can generate a proof from the nearest checkpoint rather than from the head node every time.

To overcome the limitations of the first construction, we extend its idea and link multiple one-way function chains together by a chain constructed by a pseudo-random generator in [1]. Most importantly, this multi-chain PoA construction can achieve auto-replenishment of proofs, which is crucial for its real-world deployment. Essentially, we intentionally reserve the last nodes of every OWF chain for constructing a one-time signature scheme to sign the tail nodes of new OWF chains, which will be used after the current chain is used up completely. In this way, a man-in-the-middle attacker cannot tamper with the commitment of a new chain structure. Therefore, the new chain is successfully bootstrapped by the current chain. In particular, our second scheme only needs to store one seed of PRG, and the head node of each OWF chain can be computed on the fly. However, one limitation of the previous multi-chain PoA construction is the large number of verify-points. To reduce the storage cost of the prover and the verifier, we realize that it is possible to use a Bloom filter to compress those verify-points. Consequently, the prover only needs to commit the Bloom filter of the new protocol instance, and the verifier stores the Bloom filter and caches the most recent verify-point for verification. It is sufficient to check whether the proof can result in either the currently cached verify-point or a verify-point stored in the Bloom filter.

Although our OTS-based auto-replenishment scheme is effective and secure in the standard model, it is not very efficient and cannot be applied in our first PoA scheme. Therefore, we develop a new auto-replenishment scheme from hash-based commitment. This new solution can be used by both of the PoA schemes. The idea is that the client could use an unopened proof x_i to commit the tail nodes of a new PoA protocol instance when the proof x_{i-1} is sent. When x_i is opened at time i, the verifier can verify the commitment received before. As the commitment may be lost, the client could commit the new protocol instance a few times, depending on the system setting.

Contributions. We make the following contributions in our paper:

1) [1]: We introduced a new cryptographic notion called "Proof of Aliveness" (PoA), which allows a remote device/prover to prove its "aliveness" to a server/verifier over an open network.

2) [1]: Our first PoA protocol Π_{OWF} has a single chain-based structure as the hash chain-based authentication scheme in [13], but the hash function is replaced with a more general cryptographic building block, i.e., one-way function. This led to a more general security analysis result.
3) [1]: To improve our first protocol, we proposed to leverage our first protocol as a building block to construct our second PoA protocol $\Pi_{\mathsf{OWF}}^{\mathsf{PRG}}$ that has multiple OWF chains (which are the protocol instances of Π_{OWF}). Meanwhile, we make use of a pseudo-random generator to quickly access the head nodes of those OWF chains on the fly. This allows auto-replenishment of proofs, so we can never run out of proofs.
4) **[This work]**: We further improve the storage cost of the prover and the verifier in $\Pi_{\mathsf{OWF}}^{\mathsf{PRG}}$ by leveraging a Bloom filter which is used to compress all the verify-points. This improvement could roughly save 4.7 times the storage cost for both parties. The new protocol will be called as $\Pi_{\mathsf{OWF}}^{\mathsf{PRG}}$-BF.
5) **[This work]**: We propose a new auto-replenishment scheme that is based on a hash-based commitment scheme in the random oracle model and can be used by all of our PoA schemes. The new scheme is very efficient as only one additional hash computation is required during each proof generation.
6) [1]: We showed a new modular approach to prove the security of our proposed PoA protocols in the standard model, which also significantly simplifies the security analysis.
7) **[This work]**: We study the performance constraints and define a cost model for storage-constrained devices, in contrast to [1], which only considers storage-sufficient devices. The cost model can be used to determine the number of sub-chains in our second construction.
8) **[This work]**: Last but not least, on an embedded device, Raspberry Pi, we evaluate the performance of our new PoA construction with the instantiations of the OWF and PRG in the random oracle model and the standard model respectively. We benchmark the performance of protocols with more fine-grained parameters compared to [1]. In particular, new experiments are carried out to evaluate the performance of our new auto-replenishment scheme and the proof generation time for storage-constrained devices. The results show that the proof generation time is in the order of microseconds, so they are sufficiently efficient to be deployed in real-world devices.

Organization. Related work is reviewed in Section 2. We introduce the preliminaries in Section 3. The notion of Proof of Aliveness protocols is presented in Section 4. Sections 5 and 6 present two PoA protocols. In Section 7, we propose a new auto-replenishment scheme. Section 8 presents a few applications of PoA. Implementation and evaluation results are presented in Section 9. We discuss some relevant topics in Section 10, and conclude in Section 11.

2 Related Work

One-time Password. The use of a one-way function (OWF) for generating a one-time password can be dated back to the seminal work by Lamport [13]. Lamport's scheme initiates the idea of chaining the one-time password based on OWF, i.e., the i-th password is computed with the OWF using the $(i\text{-}1)$-th password as input. However, the security of Lamport's scheme is not formally analyzed in [13], i.e., the security model, the security assumptions of OWF, and the formal security proof are not presented in [13]. Although the security of Lamport's scheme is implicitly analyzed in [15] (as a building block of a one-time signature) based on a security assumption *quasi-inverting*, the concrete hardness of quasi-inverting was not presented in [15]. Namely, we still do not have a complete security analysis of Lamport's scheme with concrete adversary's advantage in breaking it under an appropriate security model (especially in a multi-client setting). We notice that the quasi-inverting assumption is not enough for analyzing the passwords' collision probability in a multi-client setting. Alternatively, we will present a new approach for proving our OWF-chain-based PoA schemes in the standard model.

Also, there are some existing variants of the Lamport's scheme. Haller proposed the S/KEY standard [14] by specifically instantiating the OWF with a cryptographic hash function. However, Chen and Mitchell [16] pointed out that S/KEY is vulnerable to eavesdropping/replay attacks. Besides, the key length suggested by S/KEY, i.e., 64 bits, is not enough to resist state-of-the-art inverting attacks against hash functions. To overcome the drawbacks of S/KEY, Kogan *et al.* recently proposed a hash chain-based second-factor authentication token called T/Key [10] which particularly integrated the time constraint into the use of the passwords in the hash chain, so that it is also categorized as a time-based one-time password (TOTP) protocol. However, the security of T/Key is proved in the random oracle model, while a real random oracle may not be feasible in practice. It is still an open question on how to remove the random oracle assumption from T/Key. The hash chain is a finite chain, so the usage of T/Key is limited. How to reinitialize a new hash chain over an untrusted communication channel is another open problem for T/Key.

Auto-replenishment. In [17], Goyal proposed to leverage a one-time signature (OTS) scheme to reinitialize the protocol instance when the head node of the secret chain is reached. Goyal also showed an improved scheme that uses (additional) passwords in the chain as the signature secrets to sign the new protocol instance. Zhang et al. [18] later proposed a self-updating hash chain that combines the one-time signature scheme with the hash chain with a bit predicate so that the protocol instance can be self-updated along with the password spending. A similar construction idea utilizing OTS for reinitialization is also adopted in [19], [20]. However, the above schemes are developed from S/KEY without considering the time constraint as in TOTP. Their solutions inherited the shortcomings of S/KEY [16], such as password loss and abuse, which may lead to the mis-binding of new protocol instances established by attackers. In this work, we are particularly interested in designing a PoA protocol that supports auto-replenishment.

Program Attestation. Program attestation allows one to verify the program running on a device remotely. Intuitively, one can run remote attestation repeatedly to realize a PoA scheme, but hardware-based attestation requires additional trusted hardware platforms [21], and software-based attestation requires the attestation program to be implemented in the most time-efficient way on a given hardware platform [22], [23]. The device must disable its interrupts to achieve the time optimization required by software-based attestation methods. However, this action would halt the system's response to interrupt triggers during normal operations in a real-time system, potentially resulting in catastrophic consequences such as system failure. This implies that repeated execution of program attestation is not an ideal solution for checking the aliveness of a CPS device.

Authentication Channel Establishment. The challenge-response authentication style protocols [24], [25], [26] can indeed serve as a PoA scheme. However, such an approach needs more than one message flow, rendering it less efficient when compared to the straightforward authentication of a timestamp utilizing a specific authentication-related cryptographic primitive (as mentioned in Section 1). Another way of realizing PoA is to establish a secure communication channel via leveraging authenticated key exchange protocols (AKE) (such as [27], [28], [29]). AKE protocols enable two parties to establish session-aware, confidential, and authenticated channels for exchanging sensitive information, effectively achieving the secure functionality of PoA. However, this approach remains susceptible to the compromise of the verifier, if both the client and the verifier rely on a "long-term" session key (used many times) established via an AKE protocol instance to attest and verify the aliveness, respectively. For security, an alternative strategy is to engage in periodic execution of the AKE protocol between the client and verifier. However, this approach is also inefficient since it involves many public cryptographic operations and message flows in each protocol instance.

3 Preliminaries

The following briefly reviews the notions and cryptographic primitives our constructions may rely on. We denote the security parameter by κ, an empty string by $\emptyset$, and the set of integers between 1 and n by $[n] = \{1, \ldots, n\} \subset \mathbb{N}$. Let $\|$ be an operation to get the bit-length of a string. 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 runs with fresh random coins and returns x. We denote the binary representation of a value x with bit size l_n as $x = (x[1], x[2], \ldots, x[l_n]) \in \{0,1\}^{l_n}$. The other notations used in this paper are summarized in Table 1. In the following, we review the syntax and security definitions of the main cryptographic primitives used to build PoA schemes.

One-way Functions (OWF) is known as a function that is easy to evaluate but hard to invert. We let $\mathcal{I}_{\mathsf{OWF}} = \{0,1\}^{l_r}$ and $\mathcal{R}_{\mathsf{OWF}} = \{0,1\}^{\nu \cdot l_r}$ be the input and output space of OWF respectively, where $l_r \in \mathbb{N}$ is determined by security parameter κ, and $0 < \nu \leq 1$. Namely, we mainly consider length-preserving one-way functions that have identical input and output space. We denote a one-way function by two algorithms OWF = (Setup, Eval). The setup algorithm $\mathsf{Setup}(1^\kappa)$ takes input 1^κ as the security parameter, and initializes an OWF and outputs the parameter pms which will be implicitly used by the other algorithm. The evaluation algorithm $\mathsf{Eval}(x)$ takes as input a value $x \in \mathcal{I}_{\mathsf{OWF}}$, and outputs a value $y \in \mathcal{R}_{\mathsf{OWF}}$. The security of a strong OWF requires that, on given $y = \mathsf{Eval}(x)$ for an *arbitrary* x, it is hard to find its pre-image. Here, we require the one-way function to provide *almost 1-1* property that each OWF value y only has one pre-image with overwhelming probability. Note that almost 1-1 is defined as a standard property [30]. One of the famous concrete instances of OWF with almost 1-1 is the subset sum-based OWF in [31]. We define a security game $G^{\mathsf{OWF}}_{\mathcal{A},\mathsf{F}}(\kappa)$ that is played between an adversary $\mathcal{A}$ and a challenger on a one-way function F and the security parameter κ. In the security game, $\mathcal{A}$ may ask the procedures defined in Figure 1. Concretely, the adversary $\mathcal{A}$ proceeds with the game by sequentially calling the procedures Proc.Init, Proc.Challenge, and Proc.Finalize, as defined in Figure 1. We sample a uniform random value $x \xleftarrow{\$} \mathcal{I}_{\mathsf{OWF}}$ in the Proc.Challenge query (meaning that x is a random value that is unpredictable by $\mathcal{A}$). The output of the game is the result of the procedure Proc.Finalize. The adversary $\mathcal{A}$ wins if Proc.Finalize outputs 1.

TABLE 1: Notations

	Description
κ	Security parameter
ℓ	Number of clients
l_r	Input bit-length of OWF
l_s	Input bit-length of PRG
l_g	Output bit-length of PRG
l_m	Bit-length of the message to be signed
idC	Identity of a Client
idS	Identity of a Server
N	Number of nodes in an OWF-chain
η	Number of sub-chains in $\Pi^{\mathsf{PRG}}_{\mathsf{OWF}}$
ρ_{idC}	Initial secret of idC
π_{idC}	Verify-point of idC
Δ_s	The size of one time slot
Δ_{rc}	Life-span of a PoA instance
T_i	i-th time slot
T_{att}	Aliveness tolerance time
T_{start}	Start time of a PoA instance
T_{end}	End time of a PoA instance
T_{ack}	Aliveness check time
T_{ver}	Last verified time

```
Proc.Init() :                      Proc.Finalize(x*) :
OUTPUT F.Setup(1^κ)                IF F.Eval(x*) = y
                                     OUTPUT 1
Proc.Challenge() :                 ELSE OUTPUT 0
x ←$ I_OWF; y ← F.Eval(x)
OUTPUT y
```

Fig. 1: Procedures used to define security for OWF.

Definition 1. *We denote with* $\mathsf{Adv}^{\mathsf{OWF}}_{\mathcal{A},\mathsf{F}}(\kappa) := \Pr[G^{\mathsf{OWF}}_{\mathcal{A},\mathsf{F}}(\kappa) = 1]$ *the advantage of a probabilistic polynomial time (PPT) adversary* $\mathcal{A}$ *in breaking the security of a one-way function* F *under the security parameter* κ. *We say* F *is secure if no PPT adversary has non-negligible advantage* $\mathsf{Adv}^{\mathsf{OWF}}_{\mathcal{A},\mathsf{F}}(\kappa)$. *We say the one-way function* F *is* almost *1-1, if* $\forall x \neq y$ *it holds that* $\Pr[\mathsf{F.Eval}(x) = \mathsf{F.Eval}(y)] \leq \epsilon^{\text{1-1}}_{\mathsf{F}}$ *where* $\epsilon^{\text{1-1}}_{\mathsf{F}} = \epsilon^{\text{1-1}}_{\mathsf{F}}(\kappa)$ *is a negligible function in* κ *and* $\epsilon^{\text{1-1}}_{\mathsf{F}} \leq \mathsf{Adv}^{\mathsf{OWF}}_{\mathcal{A},\mathsf{F}}(\kappa)$.

Pseudo-random Generators (PRG) is a cryptographic primitive that amplifies the randomness from a short random seed into much longer "pseudo-random" bit sequences. We let $\mathcal{SS}_{\mathsf{PRG}} = \{0,1\}^{l_s}$ be the input space and $\mathcal{R}_{\mathsf{PRG}} = \{0,1\}^{l_g}$ be the output space of PRG, where the bit-lengths $(l_s, l_g) \in \mathbb{N}$ are determined by κ. We denote a pseudo-random generator by two algorithms PRG = (Setup, Gen). The setup algorithm $\mathsf{Setup}(1^\kappa)$ takes as input 1^κ, and initializes the PRG and outputs the parameter pms which will be implicitly used by the other algorithm. The randomness generation algorithm $\mathsf{Gen}(s)$ takes as input a random seed $s \xleftarrow{\$} \mathcal{SS}_{\mathsf{PRG}}$, and outputs a pseudo-random value $r \in \mathcal{R}_{\mathsf{PRG}}$. We define a security game $G^{\mathsf{PRG}}_{\mathcal{A},\mathsf{G}}(\kappa)$ that is played between an adversary $\mathcal{A}$ and a challenger based on a pseudo-random generator G and the security parameter κ. The procedure of $G^{\mathsf{PRG}}_{\mathcal{A},\mathsf{G}}(\kappa)$ is defined in Figure 4.

```
Proc.Init() :                               Proc.Finalize(b*) :
OUTPUT G.Setup($1^\kappa$)                  IF $b^* = b$
                                                OUTPUT 1
Proc.Challenge() :                          ELSE OUTPUT 0
$s \xleftarrow{\$} \mathcal{SS}_{\mathsf{PRG}}$; $b \xleftarrow{\$} \{0,1\}$
$r_0 \xleftarrow{\$} \mathcal{R}_{\mathsf{PRG}}$; $r_1 \leftarrow$ G.Gen($s$)
OUTPUT $r_b$
```

Fig. 2: Procedures used to define security for PRG.

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

Collision-resistant Hash Functions. Let $\mathsf{CRH} : \mathcal{K}_{\mathsf{CRH}} \times \mathcal{M}_{\mathsf{CRH}} \to \mathcal{Y}_{\mathsf{CRH}}$ be a family of keyed-hash functions where $\mathcal{K}_{\mathsf{CRH}} = \{0,1\}^{l_k}$ is the key space, $\mathcal{M}_{\mathsf{CRH}} = \{0,1\}^*$ is the message space and $\mathcal{Y}_{\mathsf{CRH}} = \{0,1\}^{l_h}$ is the hash value space, and the bit-lengths $(l_k, l_h) \in \mathbb{N}$ are determined by κ. The public key $hk_{\mathsf{CRH}} \in \mathcal{K}_{\mathsf{CRH}}$ defines a hash function, denoted by $\mathsf{CRH}(hk_{\mathsf{CRH}}, \cdot)$. On input a message $m \in \mathcal{M}_{\mathsf{CRH}}$, this function $\mathsf{CRH}(hk_{\mathsf{CRH}}, m)$ generates a hash value $y \in \mathcal{Y}_{\mathsf{CRH}}$. For simplicity, we may write $\mathsf{CRH}(m)$ for $\mathsf{CRH}(hk_{\mathsf{CRH}}, m)$ when hk_{CRH} is clear in the context. Given an adversary $\mathcal{A}$ and a CRH family CRH, the CRH security game $G^{\mathsf{CR}}_{\mathcal{A},\mathsf{CRH}}(\kappa)$ is defined in Figure 3.

```
Proc.Init() :                                   Proc.Finalize(m, m') :
$hk_{\mathsf{CRH}} \xleftarrow{\$} \mathcal{K}_{\mathsf{CRH}}$       IF $m \neq m' \wedge \mathsf{CRH}(m) = \mathsf{CRH}(m')$
OUTPUT $hk_{\mathsf{CRH}}$                          OUTPUT 1
                                                ELSE OUTPUT 0
```

Fig. 3: Procedures used to define security for CRH.

Definition 3. *We denote with* $\mathsf{Adv}^{\mathsf{CR}}_{\mathcal{A},\mathsf{CR}}(\kappa) := \Pr[G^{\mathsf{CR}}_{\mathcal{A},\mathsf{CRH}}(\kappa) = 1]$ *the advantage of a PPT adversary* $\mathcal{A}$ *in breaking the security of* CRH *under the security parameter* κ*. We say* CRH *is secure if no PPT adversary has non-negligible advantage* $\mathsf{Adv}^{\mathsf{CR}}_{\mathcal{A},\mathsf{CRH}}(\kappa)$.

Bloom filter. Bloom filter [32] 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 positives but no false negatives. It is well-known that the more elements are added to the BF, the larger the probability of false positives gets. To reduce the false positive rate, we follow the approach of [33], 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:

- $\mathsf{Init}(N, \epsilon)$: On input, a size N, the initialization algorithm initiates the Bloom filter of bit length $1.44\epsilon N$.
- $\mathsf{Insert}(m)$: Element insertion algorithm takes an element m as input, and inserts m into BF.
- $\mathsf{Check}(m)$: Element check algorithm returns 1 if an element m is in BF, and 0 otherwise.

Here, we need an adversarial-resilient Bloom filter with a steady representation [34]. That is, the Bloom filter has a randomized initialization algorithm but a deterministic query algorithm that does not change the representation of the a set $IS = \{m_1, \ldots, m_N\}$ (inserted into the Bloom filter). In the following, we review the adversarial resilience property [34] of a secure Bloom filter with steady representation. We define a security game (adapted from [34]) $G^{\mathsf{AR}}_{\mathcal{A},\mathsf{BF}}(\kappa, T, N, \epsilon)$ that is played between an adversary $\mathcal{A}$ and a challenger based on a Bloom filter BF scheme and the parameters (κ, T, N, ϵ), where T is the time of the Bloom filter being used. Also, the parameter T defines the running time of adversaries in the game. We assume that the Bloom filter instance BF itself does not record any randomness used during the execution of the initiation algorithm BF.Init. The procedure of $G^{\mathsf{AR}}_{\mathcal{A},\mathsf{BF}}(\kappa, T, N, \epsilon)$ is defined in Figure 4. Meanwhile, we let Get_CurrentTime() be a public function to get the current system time. Note that we model a polynomial time adversary with explicit time parameter T that is similar to the approach [34] on restricting the query number of adversaries.

```
Proc.Init(T, N, ε) :                               Proc.Finalize(m*) :
BF.Init($N, \epsilon$)                             IF $m^* \notin IS \wedge$ BF.Check($m^*$) = 1
$IS \leftarrow \mathcal{A}(T, N, \epsilon)$             $\wedge$ Get_CurrentTime() $- T_s \leq T$
BF.Insert($m_i$) for $\forall\, m_i \in IS$            OUTPUT 1
$T_s$ := Get_CurrentTime()                         ELSE OUTPUT 0
OUTPUT BF, $T_s$
```

Fig. 4: Procedures used to define security for BF.

Definition 4. *Let* $\mathsf{Adv}^{\mathsf{AR}}_{\mathcal{A},\mathsf{BF}}(\kappa, T, N, \epsilon) := \Pr[G^{\mathsf{AR}}_{\mathcal{A},\mathsf{BF}}(\kappa, T, N, \epsilon) = 1]$ *be the advantage of a PPT adversary* $\mathcal{A}$ *in breaking the security of a Bloom filter* BF *under the security parameter* κ*. We say* BF *is secure if no PPT adversary has non-negligible advantage* $\mathsf{Adv}^{\mathsf{AR}}_{\mathcal{A},\mathsf{BF}}(\kappa, T, N, \epsilon)$.

4 SECURITY NOTIONS OF PROOF OF ALIVENESS

Syntax. A proof of aliveness (PoA) scheme is a two-party protocol in which one party, client idC, proves its aliveness at a certain time to a server idS. We divide time to a collection of discrete time slots $\{T_i\}$, each of which has a time length Δ_s, i.e., $T_{i+1} - T_i = \Delta_s$. The time Δ_s is used to model the network delay while transmitting a proof, which will be considered as a global parameter. Let Δ_{rc} denote the lifespan of a PoA protocol instance, and $\mathcal{R}_{\mathsf{PoA}}$ denote the space, where a secret will be drawn from, defined by κ. Let T_{att} be the aliveness tolerance time, which determines the aliveness of a client device. That is, if the server idS fails to receive a valid proof from the client idC within T_{att}, then idS would declare the death of idC.

We denote a PoA protocol by three algorithms Π = (Setup, NextPf, Verify) which are defined as follows:

- $\mathsf{Setup}(1^\kappa, \rho_{\mathsf{idC}}, T_{start}, \Delta_{rc}, T_{att}, \mathsf{aux})$: The setup algorithm takes as input 1^κ for the security parameter κ, the client

```
Globe time slot $T$ and $\mathcal{IDP} := \{\mathsf{idC}_1, \dots, \mathsf{idC}_\ell\}$, and additional public parameter aux

Proc.Init($T_{start}$) :
for each $\mathsf{idC} \in \mathcal{IDP}$:
    $\rho_{\mathsf{idC}} \xleftarrow{\$} \mathcal{R}_{\mathsf{PoA}}$
    $(\rho_{\mathsf{idC}}, \pi_{\mathsf{idC}}, T^{\mathsf{idC}}_{ack}, pms_{\mathsf{idC}}) \xleftarrow{\$} \Pi.\mathsf{Setup}(1^\kappa, \rho_{\mathsf{idC}}, T_{start}, \Delta_{rc}, T_{att}, \mathsf{aux})$
    $\mathsf{st}_{\mathsf{idC}} \leftarrow (\rho_{\mathsf{idC}}, pms_{\mathsf{idC}})$; $\mathsf{st}_{\mathsf{idS},\mathsf{idC}} \leftarrow (\pi_{\mathsf{idC}}, T^{\mathsf{idC}}_{ack}, T_{att}, pms_{\mathsf{idC}})$
    $\mathsf{compromised}_{\mathsf{idC}} \leftarrow \mathsf{false}$
OUTPUT $\{\pi_{\mathsf{idC}}, pms_{\mathsf{idC}}\}_{\mathsf{idC} \in \mathcal{IDP}}$

Proc.Finalize() :
IF $\exists(\mathsf{idC}^*, x^*, T^*, T^{\mathsf{idC}^*,T^*}_{ack}) \in \mathsf{HD}$ $s.t.$ $(\Pi.\mathsf{Verify}(\mathsf{st}_{\mathsf{idS},\mathsf{idC}^*}, x^*, T^*, T^{\mathsf{idC}^*,T^*}_{ack}) = 1$
    $\wedge$ $\mathsf{compromised}_{\mathsf{idC}^*} = \mathsf{false}$ $\wedge$ no Proc.Get_NextPf($\mathsf{idC}^*$) at $\tilde{T}$
    s.t. $(\tilde{T} \geq T^* \wedge \tilde{T} - T^{\mathsf{idC}^*,T^*}_{ack} \leq T_{att}))$
    OUTPUT 1
ELSE OUTPUT 0

Get_CurrentTime() :
OUTPUT current_time

Proc.Start_TimeElapse() :
WHILE(1):
    IF $T + \Delta_s \leq$ Get_CurrentTime()
        $T := T + \Delta_s$

Proc.Get_NextPf(idC) :
OUTPUT $\Pi.\mathsf{NextPf}(\mathsf{st}_{\mathsf{idC}}, T)$

Proc.Receive_Pf(idC, $x$) :
$T^{\mathsf{idC},T}_{ack} := T^{\mathsf{idC}}_{ack}$
APPEND $(\mathsf{idC}, x, T, T^{\mathsf{idC},T}_{ack}) \rightarrow \mathsf{HD}$
OUTPUT $\Pi.\mathsf{Verify}(\mathsf{st}_{\mathsf{idS},\mathsf{idC}}, x, T, T^{\mathsf{idC}}_{ack})$

Proc.Corrupt(idC) :
$\mathsf{compromised}_{\mathsf{idC}} \leftarrow \mathsf{true}$
OUTPUT $\mathsf{st}_{\mathsf{idC}}$
```

Fig. 5: Procedures used to define security for PoA protocol $\Pi = (\mathsf{Setup}, \mathsf{NextPf}, \mathsf{Verify})$.

idC's initial *check-secret* $\rho_{\mathsf{idC}} \in \mathcal{R}_{\mathsf{PoA}}$, a protocol start time T_{start}, life-span Δ_{rc}, aliveness tolerance time T_{att}, and an auxiliary parameter aux, and generates the system parameter pms, an initial aliveness verify-point π_{idC} and its checked time $T^{\mathsf{idC}}_{ack} = T_{start}$. The first verify-point π_{idC} and its T^{idC}_{ack}, and pms are given to idS over a secure channel; whereas, idC stores the initial *check-secret* ρ_{idC} privately. Each client keeps a state variable $\mathsf{st}_{\mathsf{idC}} = (\rho_{\mathsf{idC}}, pms)$; and the server idS has a state $\mathsf{st}_{\mathsf{idS},\mathsf{idC}} = (\pi_{\mathsf{idC}}, T^{\mathsf{idC}}_{ack}, T_{att}, pms)$ for each client idC. These states of parties might be updated dynamically in sessions. We assume that the parameters in pms will be implicitly used by other algorithms.

- $\mathsf{NextPf}(\mathsf{st}_{\mathsf{idC}}, T)$: The proof generation algorithm takes as input the current state $\mathsf{st}_{\mathsf{idC}}$ of the client idC and a time slot T as input, and outputs the proof x that should be sent out at time T. The state $\mathsf{st}_{\mathsf{idC}}$ may be updated along with the invocation with this algorithm.
- $\mathsf{Verify}(\mathsf{st}_{\mathsf{idS},\mathsf{idC}}, x, T, T^{\mathsf{idC}}_{ack})$: The proof verification algorithm takes as input the current state $\mathsf{st}_{\mathsf{idS},\mathsf{idC}}$ (at time T) used to verify the aliveness of the client idC, a proof x, and a time slot T and the checked time T^{idC}_{ack}, and outputs 1 if x is valid for T and $T - T^{\mathsf{idC}}_{ack} \leq T_{att}$, and 0 otherwise. Meanwhile, the verify-point π_{idC} and T^{idC}_{ack} may be updated to the new ones, i.e., $\pi_{\mathsf{idC}} := x$ and $T^{\mathsf{idC}}_{ack} := T$, if this algorithm returns 1.

The *correctness* of a PoA protocol requires that $\mathsf{Verify}(\mathsf{st}_{\mathsf{idS},\mathsf{idC}}, \mathsf{NextPf}(\mathsf{st}_{\mathsf{idC}}, T), T, T^{\mathsf{idC}}_{ack}) = 1$ for any party idC and T such that $T < T^{\mathsf{idC}}_{ack} + T_{att}$.

Threats. A PoA scheme essentially allows the prover to generate an authenticated heartbeat signal. By periodically sending this signal, the verifier can know that the prover is still up and running. The main threat scenario is that the signal should be secure against man-in-the-middle attacks on the network, which can passively eavesdrop on the communication and actively tamper with the communication. In particular, the attacker may also want to inject valid aliveness proofs without learning its secret key. Meanwhile, an attacker can compromise the verifier to learn all its stored information (including secrets, if any) without actively controlling the verifier, i.e., the attacker cannot tamper with the verification program to output 1 even for invalid proofs. In addition, a client may also be corrupted. However, a secure PoA scheme should guarantee that a corrupted client would not affect the security of the other uncorrupted clients.

Security Definition. We define a security game for PoA protocols by following the game-based approach in [35], and present the relevant procedures of the security game in Figure 5. The game is defined in a multi-client setting that consists of ℓ (for $\ell \in \mathbb{N}$) honest clients with identities $\{\mathsf{idC}_1, \dots, \mathsf{idC}_\ell\}$ and one server idS. In the game, the first query is Proc.Init and the last one must be Proc.Finalize. The adversary $\mathcal{A}$ first calls Proc.Init (receiving the output) to start the game and then makes various calls to the other sub-procedures with its own inputs if needed. Specifically, the adversary $\mathcal{A}$ can call Proc.Get_NextPf(idC) to get a proof of the client idC at time T which is self-updated after calling Proc.Start_TimeElapse query, and send a proof x on behalf of idC through a Proc.Receive_Pf(idC, x) query for verification. We also model the corruption of a client via a Proc.Corrupt query. Note that the Proc.Corrupt query cannot be simulated by Proc.Get_NextPf query for some PoA protocols (e.g., our second protocol). The compromise of the verifier is modeled by allowing the adversary to learn all information that is stored at the verifier (i.e., $\{\pi_{\mathsf{idC}}, pms_{\mathsf{idC}}\}_{\mathsf{idC} \in \mathcal{IDP}}$ returned by the Proc.Init(T_{start}) procedure). In addition, we assume that all parties have a (reliable) synchronized time source (e.g., GPS time-stamp, industrial time server, or built-in device clocks). This is modeled by enabling procedures in the game to get the current time (storing in a variable *current_time*) from the public process Get_CurrentTime. We assume that the values of the variable *current_time* are monotonically increasing.

The adversary $\mathcal{A}$ calls the sub-procedures of the security game and receives the output from the corresponding sub-procedure. We say the adversary $\mathcal{A}$ wins the game if and only if the output of Proc.Finalize is 1. Let $G^{\mathsf{PoA}}_{\mathcal{A},\Pi}(\kappa, \ell, \Delta_{rc}, T_{att})$ denote the PoA game that an adversary $\mathcal{A}$ plays with a PoA protocol Π in Figure 5 with parameters κ, ℓ, Δ_{rc} and T_{att}. [1]

Definition 5. *We denote with* $\mathsf{Adv}^{\mathsf{PoA}}_{\mathcal{A},\Pi}(\kappa, \ell, \Delta_{rc}, T_{att}) := \Pr\left[G^{\mathsf{PoA}}_{\mathcal{A},\Pi}(\kappa, \ell, \Delta_{rc}, T_{att}) = 1\right]$ *the advantage of a probabilistic polynomial time (PPT) adversary* $\mathcal{A}$ *in breaking the security of a PoA protocol* Π *under parameters* κ, ℓ, Δ_{rc} *and* T_{att}. *We say*

1. Here we use identical parameters for all parties for simplicity (though each party may have distinct parameters in practice).

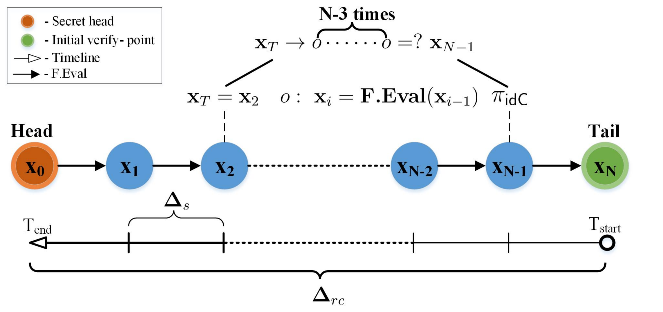


Fig. 6: Overview of Π_{OWF}. An example shows the most recent valid proof received by the server is used as the verify-point $\pi_{\mathsf{idC}} = x_{N-1}$ to verify the authenticity of the next proof. The server accepts the proof x_T (for $T = T_{end} - 2\Delta_s$) if it results in π_{idC} by $N-3$ times one-way function evaluations.

a PoA protocol is secure if no PPT adversary has non-negligible advantage $\mathsf{Adv}^{\mathsf{PoA}}_{\mathcal{A},\Pi}(\kappa, \ell, \Delta_{rc}, T_{att})$.

5 A PoA Protocol from a Single OWF-Chain

In this section, we propose a basic PoA protocol based on length-preserving strong one-way functions (OWF) with almost 1-1 property, which is named as Π_{OWF}. We show the overview of Π_{OWF} in Figure 6. Our PoA protocol has a chain structure that is adapted from Lamport's one-time password [36]. Each internal chain node (except for the tail node) is a proof that is used to prove the aliveness of the client to the server at the corresponding time. If the verifier received at least one valid proof within the aliveness tolerance time T_{att} starting from T^{idC}_{ack}, then it is convinced that the client is alive. These nodes will be consumed in a direction from the tail node to the head node of the chain.

Protocol Description of Π_{OWF}. Our protocol relies on a strong one-way function family $\mathsf{OWF} = (\mathsf{Setup}, \mathsf{Eval})$ which is almost 1-1. Basically, the PoA protocol Π_{OWF} consists of three phases which are briefly illustrated as follows:

- *Initialization*: Each client idC chooses a random initial secret $\rho_{\mathsf{idC}} \xleftarrow{\$} \mathcal{I}_{\mathsf{OWF}}$ and runs $\Pi_{\mathsf{OWF}}.\mathsf{Setup}(1^\kappa, \rho_{\mathsf{idC}}, T_{start}, \Delta_{rc}, T_{att}, \emptyset)$ algorithm to initializes an *OWF-chain* which uses a one-way function F that is uniformly selected from the OWF family by invoking $pms_{\mathsf{idC}} \xleftarrow{\$} \mathsf{F.Setup}(1^\kappa)$. The OWF-chain starts from a head node $x_0 = \rho_{\mathsf{idC}}$ and ends at a tail node x_N, where $N = \lfloor \frac{\Delta_{rc}}{\Delta_s} \rfloor$ is the length of the chain, and each node x_i for $1 \le i \le N$ is computed as $x_i := \mathsf{F.Eval}(x_{i-1})$. Eventually, the head node $\rho_{\mathsf{idC}} = x_0$ is privately stored by the client, which is the authentication secret; whereas the tail node x_N is sent to the server idS, which is set as the initial verify-point $\pi_{\mathsf{idC}} = x_N$ used to verify the proof (i.e., any node x_i for $1 \le i \le N$). We assume that the *first* verify-point is handed over to all potential verifiers[2] by a trusted party (which is responsible for deploying the client device) via an authenticated channel. For aliveness, one should ensure that $T_{att} > c \cdot \Delta_s$ for some constant $c \in \mathbb{N}$, where c is determined by the demands of specific systems.
- *Proof Generation*: The client idC runs the proof generation algorithm $\Pi_{\mathsf{OWF}}.\mathsf{NextPf}(\mathsf{st}_{\mathsf{idC}}, T)$ to obtain the proof x_T for the time slot $T := \frac{\lceil T_{end} - \mathsf{current\_time} \rceil}{\Delta_s}$. Note that all OWF-chain nodes can be computed based on x_0 thanks to the chain structure by design. Then, idC can prove its aliveness by sending the credential x_T to the server.
- *Proof Verification:* Upon receiving a proof $x = x_T$ of idC, the server idS verifies it by checking that whether the difference between the receiving time T and the last aliveness check time T^{idC}_{ack} is smaller than the aliveness tolerance time T_{att}, and the current verify-point $\pi_{\mathsf{idC}} \in \mathsf{st}_{\mathsf{idS},\mathsf{idC}}$ can be computed from x_T (as defined by the algorithm $\Pi_{\mathsf{OWF}}.\mathsf{Verify}$). If $\Pi_{\mathsf{OWF}}.\mathsf{Verify}(\mathsf{st}_{\mathsf{idS},\mathsf{idC}}, x_T, T, T^{\mathsf{idC}}_{ack}) = 1$, idS accepts the client's proof and updates the verify-point by setting $\pi_{\mathsf{idC}} := x_T$ and $T^{\mathsf{idC}}_{ack} := T_{ver} := T$.[3] Note that, in the definition of $\Pi_{\mathsf{OWF}}.\mathsf{Verify}$, we implicitly check the time T by specifying Z regarding the distance from the time T to the last verified time T_{ver}. If the time difference between the client and the server is too large such that $T - T^{\mathsf{idC}}_{ack} > T_{att}$, then the verification will fail as well.

We highlight that each party should uniformly select the OWF function (from the OWF family) to prevent the pre-processing attacks [37], [38] for inverting the function. Furthermore, to instantiate the OWF with a public function (such as SHA2), each client could (implicitly) feed the function with a randomly selected salt as in [10], [38], i.e., $\mathsf{OWF.Eval}(\cdot) = \mathsf{SHA2}(salt, \cdot)$. As each client independently executes the PoA scheme, a verifier can likewise verify the aliveness proofs of its clients in parallel.

Correctness. It is straightforward to see that the verify-point π_{idC} recorded at time T^{idC}_{ack} is a proof opened before the time T for verifying the aliveness proof x_T. Therefore, the proof $x_T \leftarrow \mathsf{NextPf}(\mathsf{st}_{\mathsf{idC}}, T)$ obtained at time T can pass the verification when $T - T^{\mathsf{idC}}_{ack} < T_{att}$, since x_T can be used to generate π_{idC} using the OWF-chain structure of π_{idC}.

Security Analysis of Π_{OWF}. Before analyzing Π_{OWF}, we first study the security of the tail node x_N of an OWF-chain in Π_{OWF} when the underlying one-way function F satisfies the almost 1-1 property. Here we define a new one-way function $\mathsf{F}_N : x_N \leftarrow \mathsf{F}_N.\mathsf{Eval}(x_0)$. Namely, F_N takes as input $x_0 \in \mathcal{I}_{\mathsf{OWF}}$ and output $x_N \in \mathcal{I}_{\mathsf{OWF}}$, where $x_0 \xleftarrow{\$} \mathcal{I}_{\mathsf{OWF}}$, and x_N is computed as in $\Pi_{\mathsf{OWF}}.\mathsf{Setup}$ based on F.

Lemma 1. *[1] Suppose that* F *is a secure almost 1-1 one-way function,* F_N *is a secure almost 1-1 one-way function with* $N \ge 2$ *and* $\mathsf{Adv}^{\mathsf{OWF}}_{\mathcal{A},\mathsf{F}_N}(\kappa) \le (N+1) \cdot \mathsf{Adv}^{\mathsf{OWF}}_{\mathcal{F},\mathsf{F}}(\kappa) + \mathcal{O}((N \cdot \mathsf{Adv}^{\mathsf{OWF}}_{\mathcal{F},\mathsf{F}}(\kappa))^2)$.

The proof of this lemma can be found in [1, Section 5]. Lemma 1 allows us to analyze the entropy of any OWF-chain node.

Theorem 1. *[1] Suppose that the one-way function* F *is almost 1-1 and secure, then* Π_{OWF} *with given parameters* κ, ℓ, Δ_{rc} *and* T_{att} *is secure with* $\mathsf{Adv}^{\mathsf{PoA}}_{\mathcal{A},\Pi_{\mathsf{OWF}}}(\kappa, \ell, \Delta_{rc}, T_{att}) \le \frac{\ell^2}{2^{l_r+1}} + \ell^2(N+1)^2((N+1)\mathsf{Adv}^{\mathsf{OWF}}_{\mathcal{F},\mathsf{F}}(\kappa) + \mathcal{O}((N \cdot \mathsf{Adv}^{\mathsf{OWF}}_{\mathcal{F},\mathsf{F}}(\kappa))^2))$*, where* $N = \lfloor \frac{\Delta_{rc}}{\Delta_s} \rfloor$.

The full proof of this theorem is given in [1].

2. One can determine the potential verifiers by a client's role in specific applications (as discussed in Section 8) of PoA protocols.

3. In this protocol, the last verified time T_{ver} is identical to the aliveness check time T^{idC}_{ack}. But in our second protocol (which will invoke the first protocol), they are different. For clarity, we use two variables here.

6 A Multiple OWF-Chain based PoA Protocol

In this section, we propose a PoA protocol based on pseudo-random generators (PRG), one-way functions (OWF), a collision-resistant hash function (CRH), and a Bloom filter (BF). Compared to the multi-chain PoA protocol $\Pi_{\mathsf{OWF}}^{\mathsf{PRG}}$ proposed in [1], this new protocol leverages an additional building block, i.e., a Bloom filter. As discussed below, the new protocol is mainly proposed to improve the shortcomings of Π_{OWF}.

We notice that there are two potential performance bottlenecks. The first one is the computation of the proof for a time T, when T is much earlier than the end time of the protocol T_{end}. To get the proof, the client needs to run $\frac{T_{end}-T}{\Delta_s}$ times OWF, which can be quite expensive. The second bottleneck is the OWF computations involved in the verification when the time T of the proof is far from the last checked time T_{ver} of the verify-point that is received by the server. This case may happen when the device is back online from the "death" status for a long period of time, e.g., the physical problem of the client is fixed. To verify the proof, the server needs to run $\frac{T-T_{ver}}{\Delta_s}$ times OWF, which may be costly as well.

Construction Idea. A solution to solve those performance bottlenecks is to explore a time-space trade-off that leverages multiple OWF chains in Π_{OWF} instances. We can split the life-span Δ_{rc} of a PoA protocol instance into η time periods, where $\eta \in \mathbb{N}$. In each period, we could use an independent OWF chain of Π_{OWF}. In the initialization phase, the client stores all OWF head nodes, and the server stores all OWF tail nodes. Then the client and the server can choose the corresponding head node and tail node for authentication and verification respectively based on the current time. It is not hard to see that the above solution can obtain better computational performance if each OWF chain is not too long. However, this solution will bring about $O(\eta)$ extra storage overhead to both the client and the server. Since the client might be storage-constrained, we further improve the storage cost for the client of the above naive solution from $O(\eta)$ to $O(1)$. This improvement eventually yields the PoA protocol $\Pi_{\mathsf{OWF}}^{\mathsf{PRG}}$ in [1]. The main idea is to leverage the PRG to generate the head nodes for those OWF chains, see Figure 7. Note that PRG is less efficient than OWF, so we keep the structure of the protocol as simple as possible, i.e., only two layers are used (rather than a multi-layer tree structure).

However, the above solution would increase the storage overhead at the verifier. Though the verifier would be a more powerful server whose storage space is large, it may need to store and manage multiple provers' verify-points, which may not be small in total. Therefore, it is also desired to reduce the verifier's storage overhead as much as possible. To achieve this goal, we adapt the original $\Pi_{\mathsf{OWF}}^{\mathsf{PRG}}$ [1] by incorporating a Bloom filter (BF) into the protocol construction. The BF is used to compress the initial verify-points of sub-chains. Besides, we let the verifier store the most recent valid proof as the current verify-point for verifying the next proof. Then, a verifier would accept a proof if it can result in a verify-point that is either stored in the Bloom filter or equal to the current cached verify-point. However, we cannot trivially insert the initial verify-point into the Bloom filter. Note that $\Pi_{\mathsf{OWF}}^{\mathsf{PRG}}$ implicitly binds the time slots with the indices of proofs. For example, from the sub-chain index of the initial verify-point, the verifier would know the valid time period of the proofs in the corresponding sub-chain. However, if we naively insert the verify-points into a Bloom filter, the verifier would no longer know the sub-chain indices of the initial verify-points. In this case, an adversary may trick the verifier into accepting an expired proof. To avoid this problem, we explicitly bind the start time of each sub-chain to the corresponding verify-point with CRH. That is, we insert $\mathsf{CRH}(x_N^i||T_{\mathsf{end}}^{i-1})$ into the Bloom filter instead. We will call this new protocol as $\Pi_{\mathsf{OWF}}^{\mathsf{PRG}}$-BF.

Protocol Description of $\Pi_{\mathsf{OWF}}^{\mathsf{PRG}}$-BF. In this protocol, we make use of a pseudo-random generator $\mathsf{G} = (\mathsf{Setup}, \mathsf{Gen})$, a Bloom filter $\mathsf{BF} = (\mathsf{Init}, \mathsf{Insert}, \mathsf{Check}, \mathsf{Pos})$, a collision-resistant hash function CRH with a key $hk_{\mathsf{CRH}} \xleftarrow{\$} \mathcal{K}_{\mathsf{CRH}}$, and the PoA protocol $\Pi_{\mathsf{OWF}} = (\mathsf{Setup}, \mathsf{NextPf}, \mathsf{Verify})$. Here, we need to initialize G with a range such that $\mathcal{R}_{\mathsf{PRG}} = \mathcal{SS}_{\mathsf{PRG}} \times \mathcal{I}_{\mathsf{OWF}}$.

The algorithms of $\Pi_{\mathsf{OWF}}^{\mathsf{PRG}}$-BF are defined in Figure 8. The protocol execution of $\Pi_{\mathsf{OWF}}^{\mathsf{PRG}}$-BF is quite similar to that of Π_{OWF} (as shown in [1, Figure 6]) so that we omit it here. We briefly introduce the three protocol execution phases of $\Pi_{\mathsf{OWF}}^{\mathsf{PRG}}$-BF in the following:

- *Initialization*: Each client idC divides the life-span Δ_{rc} of the protocol into η time periods each of which has time length $\Delta'_{rc} = \lfloor \frac{\Delta_{rc}}{\eta} \rfloor$, where η is a parameter determined by the instantiation of the PRG (that will be discussed later). For the i-th ($i \in [\eta]$) time period, idC runs the pseudo-random generator G to compute the corresponding initial secret ρ_{idC}^i and the PRG seed ss_{idC}^i (for computing the OWF-chain head node of the next time period), i.e., $ss_{\mathsf{idC}}^i||\rho_{\mathsf{idC}}^i := \mathsf{PRG.Gen}(ss_{\mathsf{idC}}^{i-1})$; and it initializes the i-th OWF chain by running the setup algorithm of Π_{OWF}, i.e., $\Pi_{\mathsf{OWF}}.\mathsf{Setup}(1^\kappa, \rho_{\mathsf{idC}}^i, T_{end}^{i-1}, \Delta'_{rc}, T_{att}, \emptyset)$, where the start time T_{start}^i of the i-th OWF chain is the end time T_{end}^{i-1} of the (i-1)-th OWF chain, and $T_{end}^1 = T_{start}$. The returned client state is $\mathsf{st}_{\mathsf{idC}} = (ss_{\mathsf{idC}}^u, \rho_{\mathsf{idC}}^u, \{T_{end}^i\}_{0 \le i \le \eta}, \eta, u, pms)$ which only includes the first OWF chain head ρ_{idC}^1 and PRG seed ss_{idC}^1. The returned server state is $\mathsf{st}_{\mathsf{idS,idC}} = (\{\mathsf{st}_{\mathsf{idS,idC}}^{\mathsf{i}}\}_{i\in[\eta]}, \{T_{end}^i\}_{i\in[\eta]}, T_{ack}^{\mathsf{idC}}, \eta)$ which include the states of all OWF chains, where $\mathsf{st}_{\mathsf{idS,idC}}^{\mathsf{i}} = (\pi_{\mathsf{idC}}^i, T_{ver}^i, pms_{\mathsf{G}})$. Then, idC initializes a Bloom filter instance by running $\mathsf{BF.Init}(\eta, \epsilon)$. For $i \in [\eta]$, idC inserts the initial verify-points into BF as $\mathsf{BF.Insert}(\mathsf{CRH}(\pi_{\mathsf{idC}}^i||T_{end}^i))$. Meanwhile, the client would keep a variable u, which is initialized to be 1, to track the index of the stored PRG seed and the head node of the OWF chain.
- *Proof Generation*: To get a proof for time T, the client idC should first figure out the index i of the OWF chain that should be used. idC should first get the i-th OWF chain head node based on the current stored PRG seed ss_{idC}^i by literally running $u := u + 1$ and $ss_{\mathsf{idC}}^u||\rho_{\mathsf{idC}}^u := \mathsf{G.Gen}(ss_{\mathsf{idC}}^{u-1})$ until $u = i$. Then, idC can ask the proof generation algorithm of Π_{OWF} to get the corresponding proof.
- *Proof Verification:* Upon receiving a proof $x = x_T$ of the client idC, the server idS declares the death of the

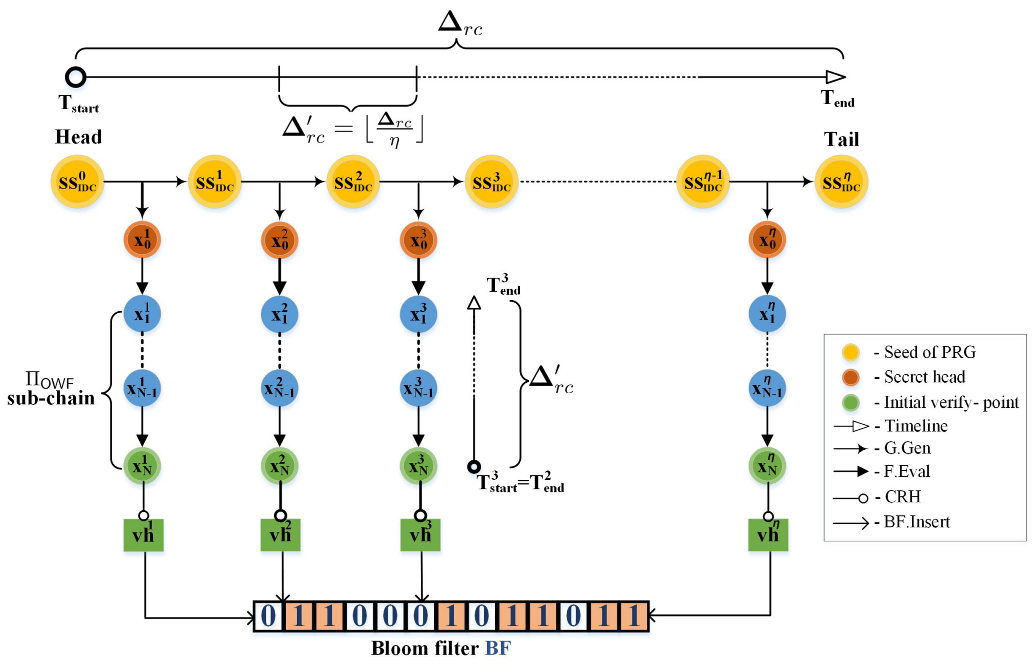

Δ_{rc}
T_{start}
T_{end}
$\Delta'_{rc} = \lfloor \frac{\Delta_{rc}}{\eta} \rfloor$
Head
Tail
SS^0_{IDC} SS^1_{IDC} SS^2_{IDC} SS^3_{IDC} $SS^{\eta-1}_{IDC}$ SS^{η}_{IDC}
x^1_0 x^2_0 x^3_0 x^{η}_0
x^1_1 x^2_1 x^3_1 x^{η}_1
x^1_{N-1} x^2_{N-1} x^3_{N-1} x^{η}_{N-1}
x^1_N x^2_N x^3_N x^{η}_N
vh^1 vh^2 vh^3 vh^{η}
Π_{OWF} sub-chain
T^3_{end}
Δ'_{rc}
$T^3_{start}=T^2_{end}$
0 1 1 0 0 0 1 0 1 1 0 1 1
Bloom filter BF
- Seed of PRG
- Secret head
- Initial verify- point
- Timeline
- G.Gen
- F.Eval
- CRH
- BF.Insert

Fig. 7: Overview of $\Pi^{\mathsf{PRG}}_{\mathsf{OWF}}$-BF. It consists of η sub-chains of Π_{OWF}. Each sub-chain is used for a time period $\Delta'_{rc} = \lfloor \frac{\Delta_{rc}}{\eta} \rfloor$.

```
$\Pi^{\mathsf{PRG}}_{\mathsf{OWF}}$-BF.Setup($1^\kappa, \rho_{\mathsf{idC}}, T_{start}, \Delta_{rc}, T_{att}, (\eta, \epsilon)$):
$pms_{\mathsf{G}} \xleftarrow{\$} \mathsf{G.Setup}(1^\kappa)$
$\Delta'_{rc} := \lfloor \frac{\Delta_{rc}}{\eta} \rfloor$
$ss^0_{\mathsf{idC}} = \rho_{\mathsf{idC}}$
$T^{\mathsf{idC}}_{ack} := T^1_{end} := T_{start}$
$\mathsf{BF.Init}(\eta, \epsilon)$
For $i \in [\eta]$:
   $ss^i_{\mathsf{idC}} || \rho^i_{\mathsf{idC}} := \mathsf{G.Gen}(ss^{i-1}_{\mathsf{idC}})$
   $(\mathsf{st}^{\mathsf{i}}_{\mathsf{idC}}, \mathsf{st}^{\mathsf{i}}_{\mathsf{idS,idC}}) := \Pi_{\mathsf{OWF}}.\mathsf{Setup}(1^\kappa, \rho^i_{\mathsf{idC}}, T^{i-1}_{end}, \Delta'_{rc}, T_{att}, \emptyset)$
   $\mathsf{st}^{\mathsf{i}}_{\mathsf{idC}} = (\rho^i_{\mathsf{idC}}, T^i_{end}, pms_{\mathsf{G}})$
   $\mathsf{st}^{\mathsf{i}}_{\mathsf{idS,idC}} = (T^i_{ver}, pms_{\mathsf{G}})$
   $vh^i_{\mathsf{idC}} := \mathsf{CRH}(\pi^i_{\mathsf{idC}} || T^{i-1}_{end})$
   $\mathsf{BF.Insert}(vh^i_{\mathsf{idC}})$
$u := 1$
$\mathsf{st}_{\mathsf{idC}} = (ss^u_{\mathsf{idC}}, \rho^u_{\mathsf{idC}}, \{T^i_{end}\}_{0 \le i \le \eta}, \eta, u, pms_{\mathsf{G}})$
$\mathsf{st}_{\mathsf{idS,idC}} = (\mathsf{BF}, \{\mathsf{st}^{\mathsf{i}}_{\mathsf{idS,idC}}\}_{i \in [\eta]}, \{T^i_{end}\}_{0 \le i \le \eta}, T^{\mathsf{idC}}_{ack}, \eta)$
Return $(\mathsf{st}_{\mathsf{idC}}, \mathsf{st}_{\mathsf{idS,idC}})$
```

```
$\Pi^{\mathsf{PRG}}_{\mathsf{OWF}}$-BF.NextPf($\mathsf{st}_{\mathsf{idC}}, T$):
$i := 0$
For $j \in [\eta]$:
   If $T^{j-1}_{end} < T \le T^j_{end}$ then
      $i := j$
If $i = 0$ then
   Return $\perp$
While $u < i$:
   $u := u + 1$
   $ss^u_{\mathsf{idC}} || \rho^u_{\mathsf{idC}} := \mathsf{G.Gen}(ss^{u-1}_{\mathsf{idC}})$
   Delete $(ss^{u-1}_{\mathsf{idC}}, \rho^{u-1}_{\mathsf{idC}})$
Return $\Pi_{\mathsf{OWF}}.\mathsf{NextPf}(\mathsf{st}^{\mathsf{i}}_{\mathsf{idC}}, T)$
```

```
$\Pi^{\mathsf{PRG}}_{\mathsf{OWF}}$-BF.Verify($\mathsf{st}_{\mathsf{idS,idC}}, x_T, T, T^{\mathsf{idC}}_{ack}$):
$i := vr := 0$
If $T - T^{\mathsf{idC}}_{ack} > T_{att}$ then
   Return $vr$
For $j \in [\eta]$:
   If $T^{j-1}_{end} < T \le T^j_{end}$ then
      $i := j$
$vr := \Pi_{\mathsf{OWF}}.\mathsf{Verify}(\mathsf{st}^{\mathsf{i}}_{\mathsf{idS,idC}}, x_T, T, T^{\mathsf{idC}}_{ack})$
If $vr = 0$ then
   Get $x_Z$ and index $i$
      computed by $\Pi_{\mathsf{OWF}}.\mathsf{Verify}$
   If $T < T^{i-1}_{end} \vee T - T^{i-1}_{end} > T_{att}$ then
      Return $vr$
   $vh^i_{\mathsf{idC}} := \mathsf{CRH}(x_Z || T^{i-1}_{end})$
   $vr := \mathsf{BF.Check}(vh^i_{\mathsf{idC}})$
If $vr = 1$ then
   $\pi_{\mathsf{idC}} := x_T; T^{\mathsf{idC}}_{ack} := T_{ver} := T$
Return $vr$
```

Fig. 8: Algorithms of the Proposed PoA Protocol $\Pi^{\mathsf{PRG}}_{\mathsf{OWF}}$-BF.

client if $T^{\mathsf{idC}}_{ack} - T$ is not smaller than T_{att}. Otherwise, idS further checks that if the current verify-point $\pi_{\mathsf{idC}} \in \mathsf{st}_{\mathsf{idS,idC}}$ can be computed from x_T (as defined by the algorithm $\Pi^{\mathsf{PRG}}_{\mathsf{OWF}}$-BF.Verify) or x_T can result in an initial verify-point that corresponds to a valid start time (i.e., $0 < T - T^{i-1}_{end} \le T_{att}$) and is stored in BF. If $\Pi^{\mathsf{PRG}}_{\mathsf{OWF}}$-BF.Verify($\mathsf{st}_{\mathsf{idS,idC}}, x_T, T, T^{\mathsf{idC}}_{ack}$) $= 1$, idS accepts the client's proof and updates the verify-point by setting $\pi_{\mathsf{idC}} := x_T$ and $T^{\mathsf{idC}}_{ack} := T_{ver} := T$.

CORRECTNESS. As the pseudo-random generator is deterministic and the lifetime of a client's OWF chain is unique, the correctness of $\Pi^{\mathsf{PRG}}_{\mathsf{OWF}}$-BF is then guaranteed by that of Π_{OWF}. Note that the Bloom filter does not change the verification concept of Π_{OWF}.

STORAGE COST COMPARISON. Here we compare the storage cost of our new protocol $\Pi^{\mathsf{PRG}}_{\mathsf{OWF}}$-BF with $\Pi^{\mathsf{PRG}}_{\mathsf{OWF}}$ [1]. We will study the computation cost of $\Pi^{\mathsf{PRG}}_{\mathsf{OWF}}$-BF later.

Both the prover and the verifier need to store the new verification state of the new protocol instance (i.e., OWF-chain tail nodes in $\Pi^{\mathsf{PRG}}_{\mathsf{OWF}}$ and Bloom filter BF in $\Pi^{\mathsf{PRG}}_{\mathsf{OWF}}$-BF). The prover needs to store the information for both the current and the new protocol instances because of the replenishment procedures. We list the minimum storage costs (without considering any optimizations discussed in Sec. 9) for running these protocols in Table 2.

TABLE 2: Storage Cost Comparison.

	Prover	Verifier
$\Pi^{\mathsf{PRG}}_{\mathsf{OWF}}$	$2l_s + 2l_r + \eta l_r$	ηl_r
$\Pi^{\mathsf{PRG}}_{\mathsf{OWF}}$-BF	$2l_s + 2l_r + 1.44\epsilon\eta$	$1.44\epsilon\eta$

Since ϵ is much smaller than l_r, $\Pi^{\mathsf{PRG}}_{\mathsf{OWF}}$-BF roughly needs $\frac{l_r}{1.44\epsilon}$ less storage space on the verifier side than that is required by $\Pi^{\mathsf{PRG}}_{\mathsf{OWF}}$. If we set $l_r = 384$ (as discussed in implementation) and $\epsilon = 56$ (which leads to a negligible false positive probability), $\Pi^{\mathsf{PRG}}_{\mathsf{OWF}}$-BF roughly saves 4.7 times storage cost comparing to $\Pi^{\mathsf{PRG}}_{\mathsf{OWF}}$.

FURTHER REMARKS. In $\Pi^{\mathsf{PRG}}_{\mathsf{OWF}}$-BF, we leverage PRG to provide a shortcut for the client to quickly generate the proof for the current time. Note that, it is possible to properly instantiate the PRG so that the $(i\text{-}1)$-th seed can be quickly obtained from the $(i\text{-}1)$-th seed without calling Gen. For example, if we instantiate the PRG with counter-mode based symmetric key encryption [39], [40], the difference between i-th and $(i\text{-}1)$-th seeds is only the "counter".

Also, one can easily generalize the two-layer structure of $\Pi^{\mathsf{PRG}}_{\mathsf{OWF}}$-BF into a multi-layer structure. However, we consider this as a trade-off between fast proof generation and fast skipping proof generation. This is because if a multi-layer structure is used, one can quickly jump over a few sub-chains and compute the proof on a new sub-chain. In a PoA application, every single proof on the chains will be used in the protocol. More layers imply more PRG invocations, which slow down the overall speed for computing all the proofs. As we will show later, by our instantiation of PRG (i.e., CTR mode AES), we can jump to any sub-chain via only one PRG operation. Additionally, a multi-layer structure may introduce more verify-points that will increase the storage cost for both the verifier and the prover. To summarize, we think the two-layer structure is necessary for proof auto-replenishment and sufficient for PoA application.

The verification time is independent of the number of sub-chains since the verifier can find the index of a specific verification key based on the current time, and thus no PRG evaluation is needed.

SECURITY ANALYSIS OF $\Pi^{\mathsf{PRG}}_{\mathsf{OWF}}$-BF. We first investigate the security of those OWF-chain head nodes generated by the PRG G. It is not hard to see that the i-th OWF-chain head node can be seen as being generated based on the pseudo-random function (PRF) proposed by Goldreich, Goldwasser, and Micali (GGM) [41] with a message with $(i\text{-}1)$-bit 0 appended with a bit 1. Therefore, we have the following corollary immediately.

Corollary 1. *[1] Suppose that* G *is a secure pseudo-random generator, then the i-th OWF-chain head node ρ^i_{idC} of the client* idC *generated in $\Pi^{\mathsf{PRG}}_{\mathsf{OWF}}$-BF is statistically close to a uniform random value with distance $i \cdot \mathsf{Adv}^{\mathsf{PRG}}_{\mathcal{A},\mathsf{G}}(\kappa)$.*

The corollary's proof follows that of GGM PRF [41].

Theorem 2. *We assume Π_{OWF} is a secure PoA protocol,* G *is a secure pseudo-random generator, and the Bloom filter has a false positive error at most $2^{-\epsilon}$. Then $\Pi^{\mathsf{PRG}}_{\mathsf{OWF}}$-BF with given parameters κ, ℓ, η, Δ_{rc} and T_{att}, is secure with $\mathsf{Adv}^{\mathsf{PoA}}_{\mathcal{A},\Pi^{\mathsf{PRG}}_{\mathsf{OWF}}\text{-BF}}(\kappa,\ell,\Delta_{rc},T_{att}) \leq \frac{\ell^2}{2^{l_r}} + \ell^2\eta^3 \cdot \mathsf{Adv}^{\mathsf{PRG}}_{\mathcal{G},\mathsf{G}}(\kappa) + \mathsf{Adv}^{\mathsf{CR}}_{\mathcal{A},\mathsf{CRH}}(\kappa) + (\ell + \ell^2\eta) \cdot \mathsf{Adv}^{\mathsf{AR}}_{\mathcal{A},\mathsf{BF}}(\kappa,\Delta_{rc},\eta,\epsilon) + \frac{\ell^2 N^3}{\eta} \cdot (\mathsf{Adv}^{\mathsf{OWF}}_{\mathcal{F},\mathsf{F}}(\kappa) + \mathcal{O}(\frac{N}{\eta} \cdot (\mathsf{Adv}^{\mathsf{OWF}}_{\mathcal{F},\mathsf{F}}(\kappa))^2))$.*

Proof. We present the full proof via the following games.

Game 0. This is the real security experiment. We have that $\Pr[\mathsf{BK}_0] = \mathsf{Adv}_0 = \mathsf{Adv}^{\mathsf{PoA}}_{\mathcal{A},\Pi^{\mathsf{PRG}}_{\mathsf{OWF}}\text{-BF}}(\kappa,\ell,\Delta_{rc},T_{att})$.

Game 1. The challenger $\mathcal{C}$ proceeds this game exactly like the previous game but aborts if two parties have the same initial secret ρ_{idC}. Since the secret ρ_{idC} is randomly chosen from the space $\mathcal{SS}_{\mathsf{PRG}} = \{0,1\}^{l_s}$, we have that $\Pr[\mathsf{BK}_0] \leq \Pr[\mathsf{BK}_1] + \frac{\ell^2}{2^{l_s+1}}$. Since the initial secrets of parties are distinct in this game, the adversary cannot exploit the Proc.Corrupt to obtain the initial secret of the challenged party.

Game 2. In this game, $\mathcal{C}$ aborts if a collision exists between the outputs of all G computations during the initialization. Due to the corollary 1, those OWF-chain head nodes are statistically close to the uniform random values with distance at most $\eta \cdot \mathsf{Adv}^{\mathsf{PRG}}_{\mathcal{G},\mathsf{G}}(\kappa)$ where η is the number of sub-chains. For ℓ parties, the total number of sub-chains is $\ell\eta$. By applying the birthday paradox, we have that $\Pr[\mathsf{BK}_1] \leq \Pr[\mathsf{BK}_2] + \frac{(\ell\eta)^2\eta}{2} \cdot \mathsf{Adv}^{\mathsf{PRG}}_{\mathcal{G},\mathsf{G}}(\kappa)$. This game's result ensures distinct OWF-chain head nodes for all parties.

Game 3. This game is simulated as before, but $\mathcal{C}$ aborts if any two OWF-chain nodes collide. As each sub-chain is a protocol instance of our single-chain PoA protocol Π_{OWF}, we can claim that the above abort event occurs with negligible probability due to the security of Π_{OWF}. Recall that each sub-chain has $\tilde{N} = \lfloor \frac{\Delta_{rc}}{\Delta_s \eta} \rfloor$ nodes. With the similar argument in the proof of Theorem 1 [1, Appendix B] (i.e., the security proof of Π_{OWF}), we have that $\Pr[\mathsf{BK}_2] \leq \Pr[\mathsf{BK}_3] + \frac{(\ell\tilde{N}\eta)^2}{2} \cdot ((\tilde{N}+1)\cdot\mathsf{Adv}^{\mathsf{OWF}}_{\mathcal{F},\mathsf{F}}(\kappa) + \mathcal{O}((\tilde{N}\cdot\mathsf{Adv}^{\mathsf{OWF}}_{\mathcal{F},\mathsf{F}}(\kappa))^2))$. As a result, the adversary cannot query Proc.Get_NextPf to get an unused aliveness proof to break the PoA security because each aliveness proof is unique in this game.

Game 4. In this game, we add an abort rule that the challenger aborts if the adversary outputs a password which results in a verify-point π^* such that $\mathsf{CRH}(\pi^*||T_\alpha) = \mathsf{CRH}(\pi^i_{\mathsf{idC}_j}||T_\beta)$ and idC_j is not corrupted, where $\pi^i_{\mathsf{idC}_j}$ is generated by the challenger, and (i,j,α,β) are arbitrary indices. I.e., the adversary finds a hash collision to an honest verify-point. If such an abort event occurs non-negligible, we could make use of the adversary to break the security of the collision-resistant hash function CRH. Thus, we have $\Pr[\mathsf{BK}_3] \leq \Pr[\mathsf{BK}_4] + \mathsf{Adv}^{\mathsf{CR}}_{\mathcal{A},\mathsf{CRH}}(\kappa)$.

Game 5. The challenger proceeds as before but rejects any password, resulting in a hashed verify-point vh^i_{idC}, which is not generated by herself. This abort rule excludes the case that the adversary successfully exploits the false positive error $2^{-\epsilon}$ of the Bloom filter. Since the adversary only needs to break one of the Bloom filters of ℓ parties, we have that $\Pr[\mathsf{BK}_4] \leq \Pr[\mathsf{BK}_5] + \ell \cdot \mathsf{Adv}^{\mathsf{AR}}_{\mathcal{A},\mathsf{BF}}(\kappa,\Delta_{rc},\eta,\epsilon)$.

Game 6. $\mathcal{C}$ in this game proceeds as the previous game but guesses in advance the OWF-chain that the adversary can successfully attack, i.e., the OWF-chain which is related to the tuple $(\mathsf{idC}^*, T^*) \in \mathsf{HD}$ leading the Proc.Finalize query to output 1. Note that idC^* must be uncorrupted. If $\mathcal{C}$'s guess is incorrect, then it aborts. Since there are ℓ honest parties and each party has η OWF-chains, the probability of the success guess is bound to $\frac{1}{\ell\eta}$. Thus, we have that $\Pr[\mathsf{BK}_5] \leq \ell\eta \cdot \Pr[\mathsf{BK}_6]$. In the sequel, we will carry out the rest of the proofs under the above success guess.

Game 7. In this game, $\mathcal{C}$ proceeds the game as before but replaces the OWF-chain head node of the guessed chain with a uniform random value. Obviously, if there exists an adversary that can distinguish this game from the previous game, then we can make use of it to construct an efficient algorithm breaking the security of G. By applying the security of G (i.e., the corollary 1), we have that $\Pr[\mathsf{BK}_6] \leq \Pr[\mathsf{BK}_7] + \eta \cdot \mathsf{Adv}^{\mathsf{PRG}}_{\mathcal{A},\mathsf{G}}(\kappa)$.

Game 8. In this game, $\mathcal{C}$ proceeds as before, but aborts if the adversary $\mathcal{A}$ submits a tuple (idC^*, x^*_T) to the Proc.Receive_Pf query such that the x^*_T leads to a value x^*_Z satsifying $\mathsf{BF.Check}(x^*_Z) = 1$ and x^*_T is not the output of any Proc.Get_NextPf(idC^*) query, i.e., $\mathcal{A}$ finds a false positive error of BF of idC^*. Note that there are at most

ℓ uncorrupted parties that may be subject to such a BF injection attack, and each Bloom filter is independently initialized and has a false positive error $2^{-\epsilon}$. By applying the security of the Bloom filter, we have that $\Pr[\mathsf{BK}_7] \leq \Pr[\mathsf{BK}_8] + \ell \cdot \mathsf{Adv}^{\mathsf{AR}}_{\mathcal{A},\mathsf{BF}}(\kappa, \Delta_{rc}, \eta, \epsilon)$.

Game 9. The challenger in this game proceeds as before but changes the $\Pi^{\mathsf{PRG}}_{\mathsf{OWF}}$-BF.Proc.Receive_Pf(idC$^*, x$) query to always return 0 if the received message x is not from a password query $\Pi^{\mathsf{PRG}}_{\mathsf{OWF}}$-BF.Proc.Get_NextPf(idC*). Note that the adversary $\mathcal{A}$, which can break the security of $\Pi^{\mathsf{PRG}}_{\mathsf{OWF}}$-BF, i.e., forging a password of the guessed OWF-chain, can distinguish the difference between this game and the previous game. Since the OWF-chain is just an instance of Π_{OWF} protocol, we can build an efficient algorithm $\mathcal{B}$ to break the security of Π_{OWF} by making use of $\mathcal{A}$. Specifically, $\mathcal{B}$ can simulate the game for $\mathcal{A}$ as the challenger in the previous game. Meanwhile, $\mathcal{B}$ answers all password queries $\Pi^{\mathsf{PRG}}_{\mathsf{OWF}}$-BF.Proc.Get_NextPf(idC*) regarding the guessed target OWF-chain by asking password queries Π_{OWF}.Proc.Get_NextPf(idC) to the Π_{OWF} challenger. $\mathcal{B}$ can answer all other queries of $\mathcal{A}$ using its own simulated secrets. In addition, $\mathcal{B}$ can forward the messages of password verification queries $\Pi^{\mathsf{PRG}}_{\mathsf{OWF}}$-BF.Proc.Receive_Pf(idC$^*, x$) concerning the target OWF-chain to the Π_{OWF} challenger as its password verification queries Π_{OWF}.Proc.Receive_Pf(idC$^*, x$). If $\mathcal{A}$ wins the game, so does $\mathcal{B}$. Thus, we have that $\Pr[\mathsf{BK}_8] \leq \Pr[\mathsf{BK}_9] + \mathsf{Adv}^{\mathsf{PoA}}_{\mathcal{B},\Pi_{\mathsf{OWF}}}(\kappa, 1, \Delta'_{rc}, T_{att})$. Since $\mathcal{A}$ cannot win in this game due to our modification, we have $\Pr[\mathsf{BK}_9] = 0$.

Putting together the probabilities in the above games, we have the result of this theorem:

$$\begin{aligned}
\mathsf{Adv}_0 \leq\ & \frac{\ell^2}{2^{l_s+1}} + \frac{\ell^2\eta^3}{2} \cdot \mathsf{Adv}^{\mathsf{PRG}}_{\mathcal{G},\mathsf{G}}(\kappa) + \mathsf{Adv}^{\mathsf{CRH}}_{\mathcal{A},\mathsf{CRH}}(\kappa) \\
& + (\ell + \ell^2\eta) \cdot \mathsf{Adv}^{\mathsf{AR}}_{\mathcal{A},\mathsf{BF}}(\kappa, \Delta_{rc}, \eta, \epsilon) \\
& + \frac{(\ell\tilde{N}\eta)^2}{2} \cdot ((\tilde{N}+1) \cdot \mathsf{Adv}^{\mathsf{OWF}}_{\mathcal{F},\mathsf{F}}(\kappa) \\
& + \mathcal{O}((\tilde{N} \cdot \mathsf{Adv}^{\mathsf{OWF}}_{\mathcal{F},\mathsf{F}}(\kappa))^2)) \\
& + \ell\eta \cdot (\eta \cdot \mathsf{Adv}^{\mathsf{PRG}}_{\mathcal{G},\mathsf{G}}(\kappa) + \mathsf{Adv}^{\mathsf{PoA}}_{\mathcal{B},\Pi_{\mathsf{OWF}}}(\kappa, 1, \Delta'_{rc}, T_{att})) \\
<\ & \frac{\ell^2}{2^{l_r}} + \ell^2\eta^3 \cdot \mathsf{Adv}^{\mathsf{PRG}}_{\mathcal{G},\mathsf{G}}(\kappa) + \mathsf{Adv}^{\mathsf{CR}}_{\mathcal{A},\mathsf{CRH}}(\kappa) \\
& + 2\ell \cdot \mathsf{Adv}^{\mathsf{AR}}_{\mathcal{A},\mathsf{BF}}(\kappa, \Delta_{rc}, \eta, \epsilon) \\
& + \ell^2\eta^2\tilde{N}^3 \cdot (\mathsf{Adv}^{\mathsf{OWF}}_{\mathcal{F},\mathsf{F}}(\kappa) + \mathcal{O}(\tilde{N} \cdot (\mathsf{Adv}^{\mathsf{OWF}}_{\mathcal{F},\mathsf{F}}(\kappa))^2)).
\end{aligned}$$

Since $\tilde{N} = \frac{N}{\eta}$, we can rewrite the above advantage to eliminate $\tilde{N}$:

$$\begin{aligned}
\mathsf{Adv}_0 <\ & \frac{\ell^2}{2^{l_r}} + \ell^2\eta^3 \cdot \mathsf{Adv}^{\mathsf{PRG}}_{\mathcal{G},\mathsf{G}}(\kappa) + \mathsf{Adv}^{\mathsf{CR}}_{\mathcal{A},\mathsf{CRH}}(\kappa) \\
& + (\ell + \ell^2\eta) \cdot \mathsf{Adv}^{\mathsf{AR}}_{\mathcal{A},\mathsf{BF}}(\kappa, \Delta_{rc}, \eta, \epsilon) \\
& + \frac{\ell^2 N^3}{\eta} \cdot (\mathsf{Adv}^{\mathsf{OWF}}_{\mathcal{F},\mathsf{F}}(\kappa) + \mathcal{O}(\frac{N}{\eta} \cdot (\mathsf{Adv}^{\mathsf{OWF}}_{\mathcal{F},\mathsf{F}}(\kappa))^2)).
\end{aligned}$$

This completes the proof of Theorem 2. □

7 Proof Replenishment

7.1 Problem Statement

Here we will deal with an important open question on how to replenish the proofs when they are used up. A simple solution is to use out-of-band mechanisms to securely send the tail node of the OWF-chain (i.e., the initial verify-point) of Π_{OWF} to the server. However, this solution imposes high maintenance costs on both the client and the server. Hence, we are motivated to resolve the replenishment problem based on the PoA construction itself, i.e., achieving auto-replenishment. We stress that the auto-replenishment feature is critical for deploying PoA in CPS which requires unlimited proofs without interruptions. As we have shown in the security result of Π_{OWF}, the advantage of an adversary is highly related to the length of the chain; i.e., the longer the chain is, the larger parameters are required. For both theoretical and practical demands, the auto-replenishment feature allows us to use a shorter chain with smaller parameters (which would improve the performance as well).

The client should first run algorithm $\Pi^{\mathsf{PRG}}_{\mathsf{OWF}}$-BF.Setup$(1^\kappa, \rho_{\mathsf{idC}}, T^{new}_{start}, \Delta_{rc}, T_{att}, \eta)$ to bootstrap the proof replenishment for a new start time T^{new}_{start}, i.e. to initialize a new PoA protocol instance with states $(\mathsf{st}^{\mathsf{new}}_{\mathsf{idC}}, \mathsf{st}^{\mathsf{new}}_{\mathsf{idS,idC}})$. The client can compute each node in the new protocol instance during the idle time between two proof generations. Here, the main problem is how the client can securely send the state $\mathsf{st}^{\mathsf{new}}_{\mathsf{idS,idC}}$ to the server via an insecure communication channel with active attackers. If the client uses a signature to sign $\mathsf{st}^{\mathsf{new}}_{\mathsf{idS,idC}}$, then the problem is solved immediately. However, a typical digital signature scheme is too complex for a resource-constrained device.

7.2 Auto-replenishment from Commitment

In the section, we introduce a new auto-replenishment scheme that is built from a hash-based commitment scheme. The new scheme does not rely on the structure of the protocol, so it can be used in both PoA schemes. In particular, it is much more efficient than our OTS-based auto-replenishment scheme [1].

Key Idea. Our main idea is to commit the verification state $\mathsf{st}^{\mathsf{new}}_{\mathsf{idS,idC}}$ of the new protocol instance using the i-th proof (which is supposed to be sent in the i-th time slot), and to send the generated commitment along with the (i-1)-th proof in the (i-1)-th time slot. Then, the commitment will be opened when the i-th proof is sent. The commitment is valid if it is received by the verifier around the (i-1)-th time slot and can be computed using the i-th proof and $\mathsf{st}^{\mathsf{new}}_{\mathsf{idS,idC}}$. However, we must deal with the possible loss of commitment due to communication failures or active attacks. This issue can be solved by generating a series of commitments using different aliveness proofs.

Scheme Description. In the following, we will elaborate on our concrete solution. Let $H : \{0,1\}^* \to \{0,1\}^\kappa$ be a cryptographic hash function that shall be modeled as a random oracle. And we let $x^\star_i$ for $i \in [\xi]$ denote the last ξ proofs to be sent in a protocol instance (of either Π_{OWF} or $\Pi^{\mathsf{PRG}}_{\mathsf{OWF}}$-BF), where $\xi = \frac{T_{att}}{\Delta_s} + 1$. That is, $x^\star_\xi$ is the last proof of a protocol instance. Note that, within the time period between $T_{end} - \Delta_s$ and $T_{end} - T_{att} - \Delta_s$ (i.e., the last epoch of aliveness verification), the verifier should receive at least one valid proof to ensure that the client is alive. In the worst case, $x^\star_{\xi-1}$ is the one received by the verifier. Generally speaking, we will use proofs $x^\star_2, \ldots, x^\star_\xi$ to commit $\mathsf{st}^{\mathsf{new}}_{\mathsf{idS,idC}}$. And we expect that at least one commitment is received by the verifier. So we add one additional aliveness verification rule for the last epoch, i.e., the client is dead if no valid commitment is received during the time $T_{end} - \Delta_s$ and $T_{end} - T_{att} - \Delta_s$. Specifically, the i-th commitment is computed as $C_i := H(x_i^\star || \mathsf{st}_{\mathsf{idS},\mathsf{idC}}^{\mathsf{new}})$. The client idC sends C_i to the verifier when sending $x_{i-1}^\star$. idC would repeatedly compute and send the commitment C_i for $i \in [\xi - 1]$. Note that all previous proofs can be computed from the current received one. Hence, to verify a commitment C_i, the verifier idS only needs to receive a valid proof $x_j^\star$ such that $i \leq j \leq \xi$. For security reasons, the verifier idS should only store the commitment C_i at the corresponding pre-defined time (which is associated with the proof $x_{i-1}^\star$. As long as one commitment is valid, idS can accept the new state $\mathsf{st}_{\mathsf{idS},\mathsf{idC}}^{\mathsf{new}}$.

Security Analysis. We first call a PoA scheme with the above auto-replenishment scheme a PoA-Rep scheme for short. The original PoA scheme (on which the PoA-Rep is built) and the resulting PoA-Rep scheme only differ in the aliveness proofs within the last time period between $T_{end} - \Delta_s$ and $T_{end} - T_{att} - \Delta_s$. Namely, an aliveness proof of the PoA-Rep scheme in the last time period consists of the aliveness proof in the original PoA scheme, the new verification state $\mathsf{st}_{\mathsf{idS},\mathsf{idC}}^{\mathsf{new}}$ and its commitments.

To show the security of the replenishment scheme, we directly prove that the PoA-Rep scheme is still a secure PoA scheme in the sense of Definition 5 (which covers the unforgeability of aliveness proofs) Namely, if the special aliveness proofs in the last time period of the current PoA-Rep protocol instance are unforgeable, then the adversary cannot inject any malicious verification state. So, the new proofs associated with the committed state $\mathsf{st}_{\mathsf{idS},\mathsf{idC}}^{\mathsf{new}}$ can only be generated by the corresponding honest client. This also implies that our replenishment scheme satisfies our security requirement of building a secure PoA scheme.

Theorem 3. *Supposed that the original Π_{OWF} and $\Pi_{\mathsf{OWF}}^{\mathsf{PRG}}$-BF are secure PoA schemes, and H is modeled as a random oracle which can be asked at most q_h times. Then a PoA scheme in $\{\Pi_{\mathsf{OWF}}, \Pi_{\mathsf{OWF}}^{\mathsf{PRG}}\text{-BF}\}$ using the above auto-replenishment scheme is still a secure PoA scheme.*

Proof. We prove the security of the PoA-Rep scheme by reducing it to that of the original PoA scheme and the security of the hash-based commitment scheme. The original PoA scheme is secure, which has been independently shown by Theorem 2. Meanwhile, the security of the hash-based commitment scheme relies on the random oracle and the security of the corresponding PoA scheme (which provides the commitment key, i.e., the unused aliveness proof).

We keep using the superscript $\star$ to denote the aliveness proof in the original PoA scheme. We present the proof of this theorem by the following games.

Game 0. This game equals the real security experiment based on the specification of the PoA-Rep scheme. Thus, we have that $\Pr[\mathsf{BK}_0] = \mathsf{Adv}_0 = \mathsf{Adv}_{\mathcal{A},\text{PoA-Rep}}^{\mathsf{PoA}}(\kappa, \ell, \Delta_{rc}, T_{att})$.

Game 1. The challenger aborts this game if the adversary outputs an unused password x_j^* of an uncorrupted party idC that does not belong to the last verified time period. Note that the aliveness proofs of the PoA-Rep scheme and its original PoA scheme are identical in those time periods that do not involve the replenishment scheme. So x_j^* can be used to break the original PoA scheme as well. Therefore, we have that $\Pr[\mathsf{BK}_0] \leq \Pr[\mathsf{BK}_1] + \mathsf{Adv}_{\mathcal{B},\Pi^*}^{\mathsf{PoA}}(\kappa, 1, \Delta'_{rc}, T_{att})$, where $\Pi^* \in \{\Pi_{\mathsf{OWF}}, \Pi_{\mathsf{OWF}}^{\mathsf{PRG}}\text{-BF}\}$.

Game 2. We let *inject-proof* denote an event that the adversary asks an oracle query $H(x_j^\star || \cdot)$ with an input $x_j^\star$ which is either an unused aliveness proof of an uncorrupted party idC (i.e., it is not returned by any Proc.Get_NextPf(idC) query) or a newly forged one of the original PoA scheme for a future time (i.e., it can pass the verification of the original PoA scheme). Intuitively, such an injected proof can enable an attacker to break the original PoA scheme. In this game, the challenger $\mathcal{C}$ proceeds as before but aborts if the inject-proof event happens. If this inject-proof event occurs with a non-negligible advantage, then we can construct an efficient algorithm $\mathcal{B}$ running $\mathcal{A}$ to break the original PoA scheme. During the simulation for $\mathcal{A}$, $\mathcal{B}$ can simulate the proofs required by $\mathcal{A}$ by first asking the corresponding query to its challenger Proc.Get_NextPf(idC) to get the honest proofs of the original PoA scheme for the uncorrupted party idC with index $j < i$. But the challenger changes the simulation of random oracle queries in the Proc.Get_NextPf(idC) queries. Namely, $\mathcal{B}$ pre-generates ξ random values for simulating the random oracles queries with the form $H(x_j^\star || \mathsf{st}_{\mathsf{idS},\mathsf{idC}}^{\mathsf{new}})$, i.e., whenever the value of $H(x_j^\star || \mathsf{st}_{\mathsf{idS},\mathsf{idC}}^{\mathsf{new}})$ (for $j \in [\xi]$) should be returned, $\mathcal{B}$ just picks an unused one from the pre-sampled values. This change is conceptual since the random oracle is simulated by the algorithm $\mathcal{B}$ with uniform random outputs that are uniquely associated with the corresponding inputs. If the adversary can query $H(x_j^\star || \mathsf{st}_{\mathsf{idS},\mathsf{idC}}^{\mathsf{new}})$, $\mathcal{B}$ can make use of $x_j^\star$ to break the security of the original PoA scheme. All the queries regarding other parties can be simulated accordingly based on the secrets chosen by $\mathcal{B}$ herself. By applying the security of the original PoA scheme, we have that $\Pr[\mathsf{BK}_1] \leq \Pr[\mathsf{BK}_2] + \mathsf{Adv}_{\mathcal{B},\Pi^*}^{\mathsf{PoA}}(\kappa, 1, \Delta'_{rc}, T_{att})$.

Game 3. In this game, the challenger rejects any proofs of PoA-Rep that are generated without querying H. Since the adversary does not know the proof value $x_j^\star$ in this game (due to the modification of the previous game), the adversary can only guess correctly on the value of $C'_j := H(x_j^\star || \mathsf{st}_{\mathsf{idS},\mathsf{idC}}^{\mathsf{new}\,\prime})$ for a malicious new state $\mathsf{st}_{\mathsf{idS},\mathsf{idC}}^{\mathsf{new}\,\prime}$. If its guess is correct, the adversary can replace the original commitment C_i with the malicious C'_i. The probability of such a correct guess is also negligible, depending on the size of the proof. Since there are ℓ parties and each party has ξ commitments, we therefore have that $\Pr[\mathsf{BK}_2] \leq \Pr[\mathsf{BK}_3] + \frac{\ell\xi}{2^{l_r}}$. As the adversary cannot win in this game, we have $\Pr[\mathsf{BK}_3] = 0$. The sum of the above advantages is still negligible, which proves this theorem. □

8 Applications

PoA can be used in any system that desires to maximize its uptime, and we just name a few below. For example, power grids, water plants, and even implanted pacemakers. If these devices can remotely prove to a trusted server that they are still running and "alive", then we are able to respond immediately in case the devices stop running.

Critical Infrastructures. Our daily life relies on the continuous operations of critical infrastructures like power grids and water plants. At the core of critical infrastructures, usually Programmable Logic Controllers (PLCs) are used to control the physical processes directly. It is crucial that the PLCs are running all the time and are controlling the processes. In 2007, a power system in California was shut down by a disgruntled contractor by shutting off the power of some computers [42]. If this attacker is more sophisticated like [6] and able to inject packets into the network traffic to deceive the control center, the power systems will not be able to respond to this incident immediately. One can integrate the proof of aliveness into the firmware of PLCs to detect any interrupted executions of the control logic. Notice that on PLCs, most of the real-world attacks occurred on the layer of control logic, which is one layer above the firmware [43], [44].

Implantable Medical Devices. The past decade has witnessed the rapid growth of implantable medical devices (IMD) like pacemakers. They became more affordable and easy to manage; however, they are playing a critical role in the body of patients. If an attacker can halt an IMD and make it undetected by external medical devices [45], it will be a new way of assassination. PoA can be easily integrated into an implantable device and keep it connected to an external device like doctors' programmers, patients' smartphones, or other smart wearable devices. Before it is used in IMD, one needs to evaluate the energy consumption of PoA protocols, as IMDs are highly power-constrained.

Surveillance Cameras. In many Hollywood movies, hackers are capable of hijacking surveillance cameras and replaying a still image despite that someone has intruded on the facilities. This kind of attack is not only shown in spy movies, but it is also feasible in the real world [46]. To prevent this attack, we can use PoA inside surveillance cameras and embed a proof into every frame as an invisible watermark [47], such that the administrators can automatically verify the freshness of each frame and ultimately prevent such a movie-style attack.

9 Implementation and Evaluation

9.1 Implementation Overview

To show the practicality of our proposed PoA protocols Π_{OWF} and $\Pi_{\mathsf{OWF}}^{\mathsf{PRG}}$-BF, we implemented them on embedded devices and evaluated their performance on a Raspberry Pi 3 [4]. We only evaluated the performance in a single-client scenario instead of a multi-client scenario in our proof. This is because the multiple clients setting is significant for security analysis, but the secrets and the chains of clients are independent of one another, and every client will operate independently as well. In other words, the performance of individual clients will remain the same in a multi-client setting. In our implementation, we instantiate the one-way function (OWF) and pseudo-random generator (PRG) with primitives which are secure in the random oracle model and the standard model, respectively. In the standard model instantiation, we implemented the OWF in [31], which is based on the subset sum problem, while the PRG is instantiated with the one built from OWF [48]. At the core of the PRG, we still use the OWF based on the subset sum problem. Under the random oracle model, counter-mode AES [40] and SHA-2 family [49] are used to instantiate the PRG and OWF, respectively. In both computation models, we instantiate CRH with SHA-2. The implementations of AES and SHA-2 are from highly optimized MIRACL library [50]. In addition, we use the adversarial resilient Bloom filter proposed in [34]. The experimental results are explained in the rest of the sub-sections.

4. Our source codes are publicly available at https://github.com/ChengluJin/Proof_of_Aliveness.

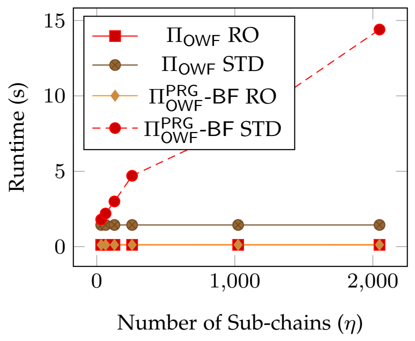


Fig. 9: Setup Time. Π_{OWF} does not have sub-chains.

In the following, we benchmark our proposed PoA protocols in terms of the key performance metrics, including the system setup time, proof generation time, proof verification time, and proof replenishment time. The number of parties that we considered is $\ell = 2^{15}$, Since both schemes are able to replenish the proofs, same amount of proofs (2^{15}) are initialized in Π_{OWF} and $\Pi_{\mathsf{OWF}}^{\mathsf{PRG}}$-BF in both the random oracle model (Π_{OWF} RO) and the standard model (Π_{OWF} STD). As the input length of OWF is determined by N and κ according to the security results, we consider the worst case for l_r when N is fixed, i.e., $l_r = 376$ in the standard model and $l_r = 384$ in the random oracle model. Similarly, we consider $\eta = N$ (for the worst case of η) to calculate the parameter l_s in the implementation; namely, we use $l_s = 256$ in the random oracle and $l_s = 384$ in the standard model. As for the Bloom filter, we choose the parameter $\epsilon = 56$, which is enough to guarantee adversarial resilience.

9.2 System Setup Time

We first measure the setup time of our PoA protocols, i.e., the time consumed for generating the verify-point to be shared with a verifier at the end of the setup procedure. The concrete performance of protocol instantiations is shown in Figure 9. Since Π_{OWF} protocol does not have sub-chains, the runtime of Π_{OWF} protocol does not increase if the total number of nodes N is fixed. However, the runtime of $\Pi_{\mathsf{OWF}}^{\mathsf{PRG}}$-BF goes up with respect to the increasing number of sub-chains. Although it is nearly a flat line, the runtime of $\Pi_{\mathsf{OWF}}^{\mathsf{PRG}}$-BF RO does increase with almost negligible increments. Clearly, Π_{OWF} RO has the best performance in its setup, as all the primitives in the random oracle model are usually more efficient in computation than the ones in the standard model in practice. Also, when the performance of Π_{OWF} RO is compared with that of $\Pi_{\mathsf{OWF}}^{\mathsf{PRG}}$-BF RO, Π_{OWF} RO is still faster, because, to generate the same amount of proofs, $\Pi_{\mathsf{OWF}}^{\mathsf{PRG}}$-BF RO needs η additional PRG invocations.

9.3 Proof Generation Time

In a proof of aliveness scheme, the prover has to send every single node on the chain to the verifier. Also, typically,

TABLE 3: Proof Generation Time for Storage Sufficient Device. The proof generation time in average case and the worst case are reported separately in the form of average/worst.

Protocol	Proof Generation Time
Π_{OWF} RO	3.74 μs/3.74 μs
Π_{OWF} STD	44.19 μs/44.19 μs
$\Pi^{\mathsf{PRG}}_{\mathsf{OWF}}$-BF RO	5.50 μs/18.00 μs
$\Pi^{\mathsf{PRG}}_{\mathsf{OWF}}$-BF STD	205.49 μs/10.46 ms

PoA is running on an embedded device, which has very constrained computation power, so we have to schedule its computation intelligently. In the sequel, we consider two implementation strategies regarding the different storage conditions. One of them is for storage-sufficient devices, and the other is for storage-constrained devices. The difference between these two devices is whether the device has enough storage space to store all checkpoints that will be used to accelerate the proof generation.

STORAGE SUFFICIENT DEVICE. This case is considered in [1], we assume that it has enough storage space to cache proofs. Our goal here is to seek a computationally efficient implementation. The best strategy will be to divide the whole chain into multiple segments with the same length and save the head of each segment as a checkpoint in a non-volatile memory. Moreover, one can store the last segment after the setup process in a non-volatile memory as well. After the PoA starts, the client sends one proof in one segment in the reverse order and generates one proof in the previous segment in the forward order. In this way, the client always stores one full segment of nodes in the memory. Suppose that in a time slot, the corresponding proof is the j-th node from the end of segment k (counting from the head node of the chain), then the proof will be fetched directly from the memory and sent to the verifier. In the same time slot, one new proof (the j-th node from the head of segment $k-1$) will be generated and replace the node used in the current time slot, such that when the nodes in segment k are used up, all the nodes in segment $k-1$ will be ready to use. Using this strategy, we equally distribute the proof generation time into every time slot and optimize its performance in the worst case because, in every time slot, the client only needs to fetch one node from the storage and generate one new node. Thus, it only requires a one-way function evaluation in every time slot for Π_{OWF} in both the average case and the worst case. An additional pseudorandom number generator evaluation is needed in the worst case of $\Pi^{\mathsf{PRG}}_{\mathsf{OWF}}$-BF. Let m denote the number of checkpoints. Then, this check-pointing strategy requires m nodes to be stored as checkpoints, and N/m nodes will need to be cached temporarily to quickly access every node in the segment derived by the closest checkpoint to the current node. Thus, in total, $m + N/m$ memory will be needed. Apparently, the most memory-efficient check-pointing strategy will be to use $\sqrt{N}$ checkpoints, which results in a memory requirement of $2\sqrt{N}$ nodes ($\sqrt{N}$ check-points, and $\sqrt{N}$ nodes per segment). Moreover, if one can shorten the overall length of the chain N, PoA can achieve better setup performance and save memory space as well.

The proof generation times of protocols for storage-sufficient devices are shown in Table 3. Since our $\Pi^{\mathsf{PRG}}_{\mathsf{OWF}}$-BF allows replenishment of proofs, we can easily shorten the length of the chain and achieve better performance as shown in the cases of $\Pi^{\mathsf{PRG}}_{\mathsf{OWF}}$-BF RO and $\Pi^{\mathsf{PRG}}_{\mathsf{OWF}}$-BF STD.

STORAGE CONSTRAINED DEVICE. Here, we introduce our implementation strategy for storage-constrained devices, i.e., the memory is not enough to store most of the proofs in an OWF chain. We let T_{F} be the runtime of the OWF function F, T_{G} be the runtime of the pseudo-random function G, and SC be the storage capacity (in bits) of the prover.

Here, the question we want to answer is which protocol is suitable for a storage-constrained device. Note that our first protocol is unsuitable since it must cache many checkpoints for proof generation. Our second protocol with an OTS-based replenishment scheme is also not the right answer since it requires storing the whole OTS signature, which is not small. Speaking of proof replenishment, an ideal choice for storage-constrained devices is the hash-based commitment solution, which is the most efficient one. Hence, we study the cost model for η based on our second protocol with instantiations that are secure in the random oracle model.

For the single chain protocol Π_{OWF}, it must satisfy the following constraint (i.e., the proof generation time $T_{\mathsf{F}}\frac{N}{\frac{SC}{l_r}}$ should be equal to or smaller than the pre-defined time interval Δ_s of two proofs): $T_{\mathsf{F}}\frac{N}{\frac{SC}{l_r}} \leq \Delta_s \Rightarrow N \leq \frac{SC\Delta_s}{T_{\mathsf{F}} l_r}$, where $\frac{SC}{l_r}$ is the maximum number of checkpoints that the prover can cache, and $\frac{N}{\frac{SC}{l_r}}$ is the distance between two checkpoints which approximates the maximum number of OWF evaluations for generating a proof.

In the following, we study the optimal η related to SC for our second protocol $\Pi^{\mathsf{PRG}}_{\mathsf{OWF}}$-BF. Note that the worst case for generating a proof is to generate the first proof in a sub-chain from its chain head node so that $\tilde{N}$ OWF evaluations should be done, where $\tilde{N}$ is the number of nodes in a sub-chain. That is, the runtime in the worst case of proof generation must satisfy the time constraint: $T_{\mathsf{G}} + T_{\mathsf{F}}\tilde{N} \leq \Delta_s$, meaning that the runtime for generating the last proof in a sub-chain should be equal to or smaller than the proof interval Δ_s. Since $\tilde{N} = \frac{N}{\eta}$, we can rewrite the above inequality of the time constraint to the following one:

$$\begin{aligned} T_{\mathsf{G}} + T_{\mathsf{F}}\tilde{N} \leq \Delta_s \Rightarrow\ & T_{\mathsf{F}}\frac{N}{\eta} \leq \Delta_s - T_{\mathsf{G}} \\ & \Rightarrow \eta \geq \frac{NT_{\mathsf{F}}}{\Delta_s - T_G}. \end{aligned} \quad (1)$$

We will study the optimal η under specific situations. By our assumption, the storage SC is smaller than the size of nodes (i.e., $\frac{Nl_r}{\eta}$) in a sub-chain, we have the inequality:

$$SC < \frac{Nl_r}{\eta} \Rightarrow \eta < \frac{Nl_r}{SC}. \quad (2)$$

By combining equations 1 and 2, we can obtain the overall constraints of η:

$$\frac{NT_{\mathsf{F}}}{\Delta_s - T_G} \leq \eta < \frac{Nl_r}{SC}. \quad (3)$$

The remaining open question is to determine the exact value of η based on equation 3. To facilitate the proof generation, we can also cache the checkpoints in each OWF

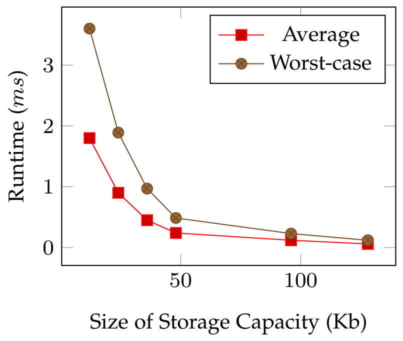


Fig. 10: Proof Generation Time of $\Pi_{\mathsf{OWF}}^{\mathsf{PRG}}$-BF RO with Optimal Performance for Storage Constrained Device. Only implementations in RO are evaluated, as we learned from the previous study that the instantiations in STD are not as efficient as the ones in RO.

chain as done in the implementation of Π_{OWF}. We assume that those checkpoints (i.e., some pre-generated proofs) are uniformly distributed in each OWF chain with the same distance between any two of them. Given the storage capacity SC and the bit-length of each proof (i.e., l_r), the number of checkpoints in each OWF chain is $1 + \frac{SC}{\eta l_r}$ (including the chain head node). We can have the distance between two checkpoints $\frac{\tilde{N}}{1+\frac{SC}{\eta l_r}} - 1$, which indicates the maximum number of OWF evaluations that should be done to compute a proof from the corresponding checkpoint. Here we assume that every proof is generated based on its nearest checkpoint. The runtime of generating a proof in the worst case is denoted by $T_{\mathsf{pf}}^{\mathsf{wo}}$. Note that the worst-case runtime $T_{\mathsf{pf}}^{\mathsf{wo}}$ is mainly determined by the time T_{G} of calculating the OWF-chain head node by running the pseudo-random generator and the maximum time $T_{\mathsf{F}}(\frac{\tilde{N}}{1+\frac{SC}{\eta l_r}} - 1)$ for generating a proof (i.e., by evaluating OWF from the OWF-chain head node to the last node before the first checkpoint). As a result, we can obtain the following equation concerning $T_{\mathsf{pf}}^{\mathsf{wo}}$:

$$T_{\mathsf{pf}}^{\mathsf{wo}} = T_{\mathsf{G}} + T_{\mathsf{F}}(\frac{\tilde{N}}{1 + \frac{SC}{\eta l_r}} - 1) = T_{\mathsf{G}} + T_{\mathsf{F}}(\frac{N l_r - \eta l_r - SC}{\eta l_r + SC}). \quad (4)$$

The proof generation time for better worst case performance for $\Pi_{\mathsf{OWF}}^{\mathsf{PRG}}$-BF is shown in Figure 10. We could see that the more storage capacity the prover has, the more efficient the protocol is. To improve worst-case performance, we can simply maximize η by setting it to $\lfloor \frac{N l_r}{SC} \rfloor$ based on equation 3. Figure 10 illustrates the proof generation time for $\Pi_{\mathsf{OWF}}^{\mathsf{PRG}}$-BF for better worst-case performance. As shown, greater storage capacity leads to a more efficient protocol.

9.4 Proof Verification Time

Note that the server can update the verification state regarding the latest received aliveness proof. In case of network failures and packet loss, the verification can take as many as $\lceil \frac{T_{att}}{\Delta_s} \rceil$ times of the OWF evaluation in Π_{OWF}, and one more hash evaluation and a bloom filter check operation in $\Pi_{\mathsf{OWF}}^{\mathsf{PRG}}$-BF. In practice, for example, when T_{att} is 5 minutes, and Δ_s is 30 seconds, then the verifier needs to run at most $\frac{300}{30} = 10$ times of OWF to verify the proof in both protocols. Hence, the verification time of Π_{OWF} instantiated in the random oracle model and the standard model can be up to 4.7 μs and 41.2 μs, respectively. The verification time of $\Pi_{\mathsf{OWF}}^{\mathsf{PRG}}$-BF is 9.4 μs in the random oracle model and 45.9 μs in the standard model.

9.5 Proof Replenishment Performance

When the proofs are about to be used up, $\Pi_{\mathsf{OWF}}^{\mathsf{PRG}}$-BF RO and $\Pi_{\mathsf{OWF}}^{\mathsf{PRG}}$-BF STD will start to generate the next chain and create the one-time signature for it. This is very critical for PoA protocols because the critical infrastructures cannot be stopped from replenishing new proofs, and there is no way to commit the new proof over an insecure communication channel securely.

In the comparison, we let "OTSRep" be the one-time signature-based auto-replenishment scheme, and "CommitRep" denote the commitment-based auto-replenishment scheme. Let "FQ" denote the frequency of the corresponding operation, "PPR" be shorthand for per-proof-generation, and "OT" denote one-time. In Table 4, we report the performance of both replenishment schemes from the perspectives of computation and communication (which also implies the storage cost). It is not hard to see that the commitment-based auto-replenishment scheme outperforms the OTS-based one from both perspectives. However, the shortcoming of CommitRep is that it cannot provide standard model security (unlike the OTS-based auto-replenishment scheme). Besides, CommitRep needs to generate a fresh commitment along with each proof generation during the aliveness tolerance time. Due to the extra cost introduced by the replenishment procedure, we recommend reducing its frequency in practice. One could enlarge the life span of a PoA protocol instance, i.e., enlarging the parameter N. The reported proof replenishment time does not include the setup time of a new set of proofs since the setup of a new instance can be done during the idle time of proof generation.

TABLE 4: Replenishment Performance.

	Applicability	Computation			Communication		
		STD	RO	FQ	STD	RO	FQ
OTSRep [1]	$\Pi_{\mathsf{OWF}}^{\mathsf{PRG}}$	5.28s	2.65ms	OT	$128 l_r$	128κ	PPR
CommitRep	Π_{OWF} $\Pi_{\mathsf{OWF}}^{\mathsf{PRG}}$-BF	-	11.22μs	PPR	-	κ	PPR

9.6 Parameter Selection in Practice

The concrete parameter setting always depends on the applications and the specific systems. The general methodology is to first identify the time-to-critical states of the system. Then, we can use the minimal time-to-critical-states as the aliveness tolerant time T_{att}, or more conservatively, multiply a certain constant (e.g., 0.5) with it. The selection of a time slot period Δ_s needs to guarantee that at least one proof will be delivered to the server within T_{att}, considering the benign package loss and network failure. The lifespan of a PoA instance Δ_{rc} can be the time between two planned maintenance of the system, as the system has to stop during maintenance.

Case Study for Parameter Selection in Water Treatment System. Let us take a real-world water treatment testbed at Singapore University of Technology and Design (SWaT) as an example [51]. Researchers have shown that the minimal time-to-critical state of the first stage (raw water processing) of SWaT is 149 seconds (2 minutes and 29 seconds) [52]. Thus 149 seconds can be set as T_{att} for this stage, and Δ_s can be set to 30 seconds. Given the parameters above, $N = 2^{22}$ will be able to support the system to operate continuously for 4 years, and therefore Δ_{rc} can be around 4 years. Also, $\Pi_{\mathsf{OWF}}^{\mathsf{PRG}}$-BF offers proof replenishment, and effectively Δ_{rc} will not be limited by N anymore.

Case Study for Parameter Selection in Evacuation Control in Metro Systems. PoA is also desired in metro systems. According to [53], the evacuation control (EVAC) in the Singapore SMRT metro system takes a heartbeat signal from Automatic Train Control (ATC) every 300 ms. If the EVAC system does not receive the heartbeat signal in 10 s, the evacuation protocol will be triggered as it means there is a train without any awareness of other trains on the same track or potential bends in the track. Naturally, T_{att} is 10 s, and δ_s is 0.3 s. If a round-trip of a train takes 5 hours, and the auto-replenishment only takes place in a terminal, then N will be 60000 $\approx 2^{16}$. Depending on the memory size of the device on the train, we can choose either Π_{OWF} or $\Pi_{\mathsf{OWF}}^{\mathsf{PRG}}$-BF for this application.

10 Discussions

This section presents additional discussions regarding our PoA construction and optimization.

Constructions in the Standard Model and the Random Oracle Model. As for the basic PoA (without replenishment), we still stick to the constructions that can be proven secure in the standard model. So we can precisely instantiate the building blocks with instances that are secure in either the standard model (for theoretical interests) or the random oracle model (for practical consideration), respectively. In Theorem 2, we have analyzed our new PoA protocol $\Pi_{\mathsf{OWF}}^{\mathsf{PRG}}$-BF in the standard model. It is suitable for a scenario in which the proofs are initialized enough for a short lifespan of the application. When considering a longer lifespan, a PoA scheme can integrate with an independent replenishment scheme to automatically replenish aliveness proofs by the PoA protocol itself. Hence, we propose a new replenishment scheme using a hash-based commitment scheme that is secure in the random oracle model. That is, we here are interested in the efficiency of the PoA scheme in practice by making the trade-off regarding the provable security in the standard model and the random oracle model. In Theorem 3, we have proved that our standard model secure PoA scheme using our new auto-replenishment scheme is still a secure PoA scheme but instead in the random oracle model (due to the requirement of the analysis of the new auto-replenishment scheme). Meanwhile, our security reduction of the new PoA schemes (with replenishment) is conducted modularly to make the most use of the existing security of the basic PoA schemes. That is, we reduce the impact of the random oracle model as much as possible so that we can precisely instantiate the parameters and keys (especially for the length of keys) in terms of the security of basic PoA schemes that have proofs in the standard model. For practicality (as in [1]), we also consider the instantiations of all other building blocks in the random oracle models since they are well-known to be more efficient than their standard model secure counterparts.

Resilience of Verifier Compromise. Our PoA schemes can resist verifier compromise because the initial *check-secret* ρ_{idC} is privately stored on the client side and is never shared with the verifier. Even if the corresponding verifier is compromised, the client's private key ρ_{idC} is not exposed as it is not stored at the verifier. Furthermore, we explicitly formalize the compromise of the verifier in our security model of PoA (especially in the Game) by allowing an adversary to learn all information (i.e., $\pi_{\mathsf{ID_C}}$) that will be stored at the verifier via the Proc.Init procedure. Hence, the security proofs of our schemes given under such a PoA security model can imply that adversaries who compromise the verifier cannot gain non-negligible advantages to break our PoA schemes.

Comparison with Related Authentication Schemes. Let us first compare our PoA schemes with three general ways of realizing authentication schemes. 1) One can build an authentication scheme based on symmetric key cryptography, like message authentication codes [7]. The client device can keep sending a pre-determined sequence of MAC values at pre-determined time windows to realize a similar functionality with only one-way communication as our PoA scheme. However, the security assumption of a symmetric key-based authentication scheme is generally weaker than our PoA schemes, as the security will be broken if an attacker is allowed to compromise the verifier's secret, as in our security model. 2) One can also rely on well-established asymmetric cryptography to build a PoA scheme: the prover can keep sending a pre-determined sequence of signatures at pre-determined time windows. This construction can also satisfy similar security requirements as ours, e.g., securing against verifier compromise. However, asymmetric key-based schemes (signature schemes) usually incur a bigger performance overhead on resource-constraint CPS devices. A detailed discussion in this regard is provided later. 3) As a recent trend, physical characteristics of devices have been leveraged for authentication as well, like using Physical Unclonable Functions in devices [54] or using the existing physical characteristics of CPS devices [55], [56]. However, they either suffer from a non-negligible false positive/negative rate [55], [56] or incur a large overhead on the client side to enable strong error corrections [57].

Secondly, we compare our schemes with digital signature-based authentication schemes more thoroughly. As pointed out by Kogan et al. [10], the hash-chained TOTP scheme is well-known as a lightweight authentication scheme that can outperform the digital signature-based authentication schemes. This is because a TOTP scheme only needs a few hash operations to generate an aliveness proof rather than more expensive public key cryptographic (PKC) operations. Although there exist lightweight signature schemes (e.g., [58], [59]) that do not run PKC operations at the signer, they need trusted third parties (TTP) to replenish verify-points. As a result, such lightweight signature schemes may incur single-point failure. Namely, the signature scheme becomes insecure once the TTP is compromised. In contrast, our TOTP-based PoA schemes do not need any TTP, since they can automatically replenish proofs via our tailored replenishment scheme. In a nutshell, an asymmetric TOTP scheme is the most suitable lightweight authentication primitive (to the best of our knowledge) for building a PoA scheme without TTPs.

Lastly, in Table 5, we present a more concrete com-

TABLE 5: Comparison with Signatures.

Scheme	Client			Verifier	
	Secret Size	Proof Size	Proof Gen Cost	Verify-point Size	Proof Verify Cost
ECDSA [60]	$\|q\|$	$2\|q\|$	EMul	$\|q\|$	1.3 EMul
SchnorrQ [61]	$\|q\|$	$2\|q\|$	EMul	$\|q\|$	1.3 EMul
Π_{OWF} RO	l_r	l_r	H	l_r	10 H
$\Pi^{\mathsf{PRG}}_{\mathsf{OWF}}$-BF RO	l_s	l_r	H+PRG	$1.44\epsilon\eta$	11 H

parison between our PoA schemes (in the random oracle model) for storage-sufficient devices and two digital signature schemes. These signature schemes could potentially serve as PoA alternatives, offering similar security properties to our solutions (capable of providing unlimited proofs and resisting verifier compromise). Let q be a large prime number representing the order of a cyclic group used by the corresponding schemes. The detailed performance of our schemes (with exemplary parameters) can be found in Section 9. The secret and the verify-point sizes of the compared schemes are close. However, based on the benchmark results of NIST [62], we conclude that the proof generation and the verification costs of our PoA schemes are more efficient than the compared signature schemes.

11 Conclusions

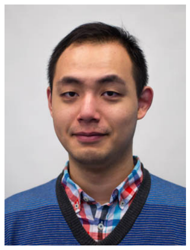
We first revisited the second PoA construction previously presented in [1]. We further improved its storage cost by using a Bloom filter that is used to compress the verification key. We also introduced a new approach to achieve auto-replenishment based on a commitment scheme in the random oracle model. Compared to the OTS-based auto-replenishment scheme in [1], the new scheme is much more efficient (i.e., only one additional hash is required), and it can be applied to both single-chain and multi-chain PoA protocols. Moreover, we benchmarked the PoA protocols and the auto-replenishment schemes with more fine-grained parameters.

References

[1] C. Jin, Z. Yang, M. van Dijk, and J. Zhou, "Proof of aliveness," in *ACSAC*. ACM, 2019, pp. 1–16.
[2] K. Zetter, "Inside the cunning, unprecedented hack of ukraine's power grid," https://www.wired.com/2016/03/inside-cunning-unprecedented-hack-ukraines-power-grid/, 2016, accessed: 2019-05-18.
[3] B. Johnson, D. Caban, M. Krotofil, D. Scali, N. Brubaker, and C. Glyer, "Attackers deploy new ics attack framework "triton" and cause operational disruption to critical infrastructure," 2017, [Accessed Oct., 2018]. [Online]. Available: https://www.fireeye.com/blog/threat-research/2017/12/attackers-deploy-new-ics-attack-framework-triton.html
[4] C. Bing, "Trisis has the security world spooked, stumped and searching for answers," 2018, [Accessed Oct., 2018]. [Online]. Available: https://www.cyberscoop.com/trisis-ics-malware-saudi-arabia/
[5] T. Seals, "Sas 2019: Triton ics malware hits a second victim," 2019, [Accessed Jun., 2019]. [Online]. Available: https://threatpost.com/triton-ics-malware-second-victim/143658/
[6] S. Kalle, N. Ameen, H. Yoo, and I. Ahmed, "Clik on plcs! attacking control logic with decompilation and virtual plc," in *Binary Analysis Research (BAR) Workshop @ NDSS*, 2019.
[7] J. Katz and Y. Lindell, *Introduction to modern cryptography*. CRC press, 2014.
[8] D. Liu and P. Ning, "Multilevel μtesla: Broadcast authentication for distributed sensor networks," *ACM Trans. Embedded Comput. Syst.*, vol. 3, no. 4, pp. 800–836, 2004.
[9] D. M'Raihi, S. Machani, M. Pei, and J. Rydell, "Totp: Time-based one-time password algorithm," Tech. Rep., 2011.
[10] D. Kogan, N. Manohar, and D. Boneh, "T/key: Second-factor authentication from secure hash chains," in *CCS*. ACM, 2017.
[11] Duo, "Duo security," https://duo.com/, 2018, Accessed: Dec. 2018.
[12] Google, "Google 2-step verification," https://www.google.com/landing/2step/, 2018, Accessed: Dec. 2018.
[13] L. Lamport, "Password authentication with insecure communication," *Commun. ACM*, vol. 24, no. 11, pp. 770–772, Nov. 1981.
[14] N. Haller, "The s/key one-time password system," 1995.
[15] S. Even, O. Goldreich, and S. Micali, "On-line/off-line digital signatures," *J. Cryptology*, vol. 9, no. 1, pp. 35–67, 1996.
[16] C. J. Mitchell and L. Chen, "Comments on the S/KEY user authentication scheme," *Operating Systems Review*, vol. 30, no. 4, pp. 12–16, 1996.
[17] V. Goyal, "How to re-initialize a hash chain," *IACR Cryptology ePrint Archive*, vol. 2004, p. 97, 2004.
[18] H. Zhang and Y. Zhu, "Self-updating hash chains and their implementations," in *WISE*, vol. 4255. Springer, 2006, pp. 387–397.
[19] H. Zhang, X. Li, and R. Ren, "A novel self-renewal hash chain and its implementation," in *EUC (2)*. IEEE, 2008, pp. 144–149.
[20] X. Yang, J. Wang, J. Chen, and X. Pan, "A self-renewal hash chain scheme based on fair exchange idea(srhc-fei)," in *ICIT*, vol. 8, July 2010, pp. 152–156.
[21] V. Costan and S. Devadas, "Intel sgx explained." *IACR Cryptology ePrint Archive*, vol. 2016, no. 086, pp. 1–118, 2016.
[22] C. Castelluccia, A. Francillon, D. Perito, and C. Soriente, "On the difficulty of software-based attestation of embedded devices," in *in CCS*. ACM, 2009, pp. 400–409.
[23] B. Chen, X. Dong, G. Bai, S. Jauhar, and Y. Cheng, "Secure and efficient software-based attestation for industrial control devices with arm processors," in *ACSAC*. ACM, 2017, pp. 425–436.
[24] C. Wang, D. Wang, Y. Duan, and X. Tao, "Secure and lightweight user authentication scheme for cloud-assisted internet of things," *IEEE Trans. Inf. Forensics Secur.*, vol. 18, pp. 2961–2976, 2023.
[25] D. Liu, Q. Wang, M. Zhou, P. Jiang, Q. Li, C. Shen, and C. Wang, "Soundid: Securing mobile two-factor authentication via acoustic signals," *IEEE Trans. Dependable Secur. Comput.*, vol. 20, no. 2, pp. 1687–1701, 2023.
[26] K. Yang, Z. Zhang, Y. Tian, and J. Ma, "A secure authentication framework to guarantee the traceability of avatars in metaverse," *IEEE Trans. Inf. Forensics Secur.*, vol. 18, pp. 3817–3832, 2023.
[27] Y. Lyu, S. Liu, S. Han, and D. Gu, "Privacy-preserving authenticated key exchange in the standard model," in *ASIACRYPT (3)*. Springer, 2022, pp. 210–240.
[28] Q. Fan, J. Chen, M. Shojafar, S. Kumari, and D. He, "Sake*: A symmetric authenticated key exchange protocol with perfect forward secrecy for industrial internet of things," *IEEE Trans. Ind. Informatics*, vol. 18, no. 9, pp. 6424–6434, 2022.
[29] C. Jin, Z. Yang, T. Xiang, S. Adepu, and J. Zhou, "HMACCE: establishing authenticated and confidential channel from historical data for industrial internet of things," *IEEE Trans. Inf. Forensics Secur.*, vol. 18, pp. 1080–1094, 2023.
[30] O. Goldreich, *The Foundations of Cryptography - Volume 1, Basic Techniques*. Cambridge University Press, 2001.
[31] R. Impagliazzo and M. Naor, "Efficient cryptographic schemes provably as secure as subset sum," *J. Cryptol.*, vol. 9, no. 4, pp. 199–216, 1996.
[32] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, 1970.
[33] A. Pagh, R. Pagh, and S. S. Rao, "An optimal bloom filter replacement," in *SODA*. SIAM, 2005, pp. 823–829.
[34] M. Naor and E. Yogev, "Bloom filters in adversarial environments," in *CRYPTO (2)*. Springer, 2015, pp. 565–584.
[35] M. Bellare and P. Rogaway, "The security of triple encryption and a framework for code-based game-playing proofs," in *EUROCRYPT*, vol. 4004. Springer, 2006, pp. 409–426.

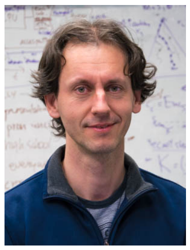
[36] L. Lamport, "Constructing digital signatures from a one-way function," SRI International, Tech. Rep., 1979.
[37] A. De, L. Trevisan, and M. Tulsiani, "Time space tradeoffs for attacks against one-way functions and prgs," in *CRYPTO*, vol. 6223. Springer, 2010, pp. 649–665.
[38] Y. Dodis, S. Guo, and J. Katz, "Fixing cracks in the concrete: Random oracles with auxiliary input, revisited," in *EUROCRYPT (2)*, 2017, pp. 473–495.
[39] N. F. Pub, "197: Advanced encryption standard (aes)," *Federal information processing standards publication*, vol. 197, no. 441, p. 0311, 2001.
[40] E. B. Barker and J. M. Kelsey, *Recommendation for random number generation using deterministic random bit generators (revised)*. US Department of Commerce, Technology Administration, NIST, 2007.
[41] O. Goldreich, S. Goldwasser, and S. Micali, "How to construct random functions," *J. ACM*, vol. 33, no. 4, pp. 792–807, 1986.
[42] C. Lamb, "Man pleads guilty in caliso computer attack," 2007, [Accessed Oct., 2018]. [Online]. Available: https://www.bizjournals.com/sacramento/stories/2007/12/10/daily65.html
[43] R. Spenneberg, M. Brüggemann, and H. Schwartke, "Plc-blaster: A worm living solely in the plc," *Black Hat Asia (p. N/A)*, 2016.
[44] J. Klick, S. Lau, D. Marzin, J.-O. Malchow, and V. Roth, "Internet-facing plcs as a network backdoor," in *CNS*. IEEE, 2015, pp. 524–532.
[45] D. Halperin, T. S. Heydt-Benjamin, K. Fu, T. Kohno, and W. H. Maisel, "Security and privacy for implantable medical devices," *IEEE pervasive computing*, no. 1, pp. 30–39, 2008.
[46] C. Heffner, "Exploiting network surveillance cameras like a hollywood hacker," *Black Hat USA*, 2013.
[47] G. W. Braudaway, "Protecting publicly-available images with an invisible image watermark," in *Image Processing*. IEEE, 1997, pp. 524–527.
[48] Y. Yu, X. Li, and J. Weng, "Pseudorandom generators from regular one-way functions: New constructions with improved parameters," in *AsiaCrypt*. Springer, 2013, pp. 261–279.
[49] S. H. Standard, "Fips pub 180-2," *National Institute of Standards and Technology*, 2002.
[50] "Miracl cryptographic library," 2018. [Online]. Available: https://bit.ly/2MltKVG
[51] "Swat: Secure water treatment testbed," https://itrust.sutd.edu.sg/testbeds/secure-water-treatment-swat/, 2015.
[52] J. H. Castellanos and J. Zhou, "A modular hybrid learning approach for black-box security testing of cps," in *ACNS*. Springer, 2019, pp. 196–216.
[53] E. F. M. Josephlal, S. Adepu, Z. Yang, and J. Zhou, "Enabling isolation and recovery in plc redundancy framework of metro train systems," *Int. J. Inf. Secur.*, pp. 1–13, 2021.
[54] B. Gassend, D. Clarke, M. Van Dijk, and S. Devadas, "Silicon physical random functions," in *AsiaCCS*, 2002, pp. 148–160.
[55] C. M. Ahmed, M. Ochoa, J. Zhou, and A. Mathur, "Scanning the cycle: timing-based authentication on plcs," in *AsiaCCS*, 2021, pp. 886–900.
[56] C. M. Ahmed, J. Zhou, and A. P. Mathur, "Noise matters: Using sensor and process noise fingerprint to detect stealthy cyber attacks and authenticate sensors in cps," in *ACSAC*, 2018, pp. 566–581.
[57] M. van Dijk and C. Jin, "A theoretical framework for the analysis of physical unclonable function interfaces and its relation to the random oracle model," *J. Cryptol.*, vol. 36, no. 4, p. 35, 2023.
[58] Z. Yang, C. Jin, Y. Tian, J. Lai, and J. Zhou, "Lis: Lightweight signature schemes for continuous message authentication in cyber-physical systems," in *AsiaCCS*. ACM, 2020, pp. 719–731.
[59] Z. Yang, T. T. A. Dinh, C. Yin, Y. Yao, D. Yang, X. Chang, and J. Zhou, "LARP: A lightweight auto-refreshing pseudonym protocol for V2X," in *SACMAT*. ACM, 2022, pp. 49–60.
[60] S. A. Vanstone, "Elliptic curve cryptosystem—the answer to strong, fast public-key cryptography for securing constrained environments," *Information security technical report*, vol. 2, no. 2, pp. 78–87, 1997.
[61] C. Costello and P. Longa, "Schnorrq: Schnorr signatures on fourq," *MSR Tech Report*, 2016.
[62] H. Tschofenig, M. Pegourie-Gonnard, and I. B. Unit, "Nist lightweight cryptography workshop 2015 session vii: Implementations & performance."

Zheng Yang received his Ph.D. degree from Horst Görtz Institute for IT Security, Ruhr-University Bochum, in 2013. He is currently a Professor with the College of Computer and Information Science, Southwest University, China. He was a post-doc researcher with the University of Helsinki, and the Singapore University of Technology and Design. His main research interests include information security, cryptography, and privacy.



Chenglu Jin received his Ph.D. degree from the Electrical and Computer Engineering Department, University of Connecticut, USA, in 2019. He is currently a tenure-track researcher in the Computer Security Group in Centrum Wiskunde & Informatica (CWI Amsterdam), the Netherlands. His research interests are cyber–physical system security, hardware security, and applied cryptography

Xuelian Cao received her MS degree in Computer Application Technology from Southwest University, in 2021. She is currently studying for a Ph.D. degree in Computer Science and Technology at the Southwest University. Her research interests include Blockchain and cryptography.



Marten van Dijk (Fellow, IEEE) is currently Group Leader of the Computer Security Group, CWI, The Netherlands, with over 20 years of experience in both industry (Philips Research and RSA Laboratories) and academia (MIT and UConn). His work has been recognized by the IEEE CS Edward J. McCluskey Technical Achievement Award 2023 and the IEEE & ACM A. Richard Newton Technical Impact Award in Electronic Design Automation 2015, and has received several best (student) paper awards.

Jianying Zhou is a professor and center director for iTrust at Singapore University of Technology and Design (SUTD). He received PhD in Information Security from Royal Holloway, University of London. His research interests are in applied cryptography and network security, cyber-physical system security, mobile and wireless security. He is a co-founder & steering committee co-chair of ACNS. He is also steering committee chair of ACM AsiaCCS, and steering committee member of Asiacrypt. He has served 200+ times in international cyber security conference committees (ACM CCS & AsiaCCS, IEEE CSF, ESORICS, RAID, ACNS, Asiacrypt, FC, PKC etc.) as general chair, program chair, and PC member. He is associate editor-in-chief of IEEE Security & Privacy. He has also been in the editorial board of top cyber security journals including IEEE TDSC, IEEE TIFS, Computers & Security. He is an ACM Distinguished Member. He received the ESORICS Outstanding Contribution Award in 2020, in recognition of his contributions to the community.