Lightweight Delegated Authentication with Identity Fraud Detection for Cyber-physical Systems

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

Chao Yin  
Chongqing University of Technology  
Chongqing, China  
cy_yin@2019.cqut.edu.cn

Chenglu Jin  
CWI Amsterdam  
Amsterdam, The Netherlands  
chenglu.jin@cwi.nl

Jianting Ning  
Singapore Management University  
Singapore  
jting88@gmail.com

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

ABSTRACT

Delegated authentication is a very popular and effective paradigm to deal with entity authentication problems for resource-constrained clients in cyber-physical systems; namely, the authentication between two clients is proxied by a trusted authentication server. However, an attacker may compromise the authentication server to impersonate the clients for sabotaging the cyber-physical systems. To detect the identity fraud attacks caused by an authentication server compromise, we propose two mutual authentication protocols by using a pseudo-random function family and a one-time signature (OTS) scheme. Our idea is to leverage the continuously evolving OTS signing and verifying keys at the signer and the verifier sides respectively for identity fraud detection because an identity fraud attack would violate the victim’s honest OTS key update procedure. The proposed protocols are proven secure under a new mutual authentication security model that formulates the identity fraud detection.

CCS CONCEPTS

• Security and privacy → Key management; Multi-factor authentication.

KEYWORDS

Authentication, Identity fraud detection, Intrusion detection, Key management

ACM Reference Format:

1 INTRODUCTION

A cyber-physical system (CPS) is a system comprised of various physical devices that are interconnected over a network to exchange data autonomously. Nowadays, CPS can not only facilitate the daily lives of people but also improve industrial efficiency, where the concrete CPS includes railway systems, water treatment systems, and many other Internet-of-Things systems. However, due to the computational limitation of those client devices, the security of CPS is always not a priority, which results in that malicious attackers can attack CPS in many ways [14, 21]. The first step to prevent unauthorized access to those critical devices is to authenticate the identity of communication principles before they can accept commands or exchange data. Practical cross-domain communication scenarios in CPS include: multi-channel communications (e.g., communication between two stations of a railway system), and edge-computing driven communication between two companies’ CPS. In these scenarios, each entity is typically registered to a domain-specific server, such as a station management server (for railway communications) or enterprise authentication server (for edge-computing communications). The two communicating entities from different CPS domains typically neither share a symmetric key nor possess each other’s public key certificate, so the authentication between them is usually delegated by their own domain servers.

Many lightweight authentication protocols have been proposed in the context of delegated authentication [4, 10, 11, 19, 23, 24, 37, 38]. Specifically, the authentication of clients (including users and constrained devices) is outsourced to a trustworthy third-party authentication server, which relies on a symmetric key framework. In the traditional framework, there is no shared key between clients, but each client may have a pre-shared key with the authentication server. However, an obvious drawback is that the compromise of an authentication server implies a compromise of the whole system since the attacker would know all authentication keys. That is, an attacker who compromised the authentication server can impersonate any client. In practice, the compromise of an authentication server may also be caused by malicious insiders who try to sabotage (e.g., impersonate clients to send malicious instructions to client devices in a critical CPS). To the best of our knowledge, there is no existing technique to identify which authentication server is compromised in such a delegated authentication framework. Due to the existence of zero-alarm attacks [3, 29], it may be even harder for
Figure 1: Overview of Our Delegated Mutual Authentication Protocol

administrators to identify the compromised server since there is no obvious trace to track such attacks. To this end, we are motivated to seek a more affirmative way to detect the identity fraud events caused by a compromised authentication server, so that we can not only fix the compromised server but also help administrators identify the responsible staff for the identity fraud events (e.g., in the case of insider attacks).

In a large CPS, like manufacturing systems and water treatment systems, multiple controllers (called Programmable Logic Controller or PLC in such systems) sometimes need to communicate with each other to cooperate to complete a complex process. Since each controller only has very limited computational power for security enhancement, it is particularly interesting to shift the burden of authentication to the servers, which leads to the delegated authentication paradigm. Moreover, due to the air-gapped nature of these systems, the servers, which are connected to the enterprise network and sometimes the Internet, are more vulnerable to remote attacks. Thus, we are also interested in solving the security issues after a fraction of servers are compromised and impersonate other controllers in the network. By introducing identity fraud detection, we will be able to audit the system and detect such an attack.

Our Work. To mitigate the compromise of an authentication server in the delegated authentication framework, we focus on studying identity fraud detection (IFD) in this work. Our goal is to design lightweight mutual authentication protocols that enable an honest authentication server to detect not only a compromised client but also the identity fraud events caused by other compromised authentication servers. An overview of our proposed protocol is presented in Fig. 1. Note that we adopt a similar system model standardized for cross-domain authentication [34].

In our protocol, each client has a separate authentication delegator which is selected from a pool of authentication servers. Our main idea for achieving IFD is to leverage evolving one-time signature (OTS) keys (of clients and servers). We allow a party to commit (sign) its next public key in a signature using its current OTS secret key, which would lead to a signature chain on the fly. The authentication delegator selected by a client is responsible for the update of its OTS key once so that it will be marked as unavailable after the corresponding session is over. The authentication delegator will become available after its authentication task is checked by the other authentication servers. For a pool with size \( z \) workers, every OTS key update procedure in a session will be checked by \( z - 1 \) authentication servers in the subsequent sessions.

To resist network failures, a party should locally ‘cache’ the currently unchecked signature-chain and ‘replay’ it in the next session. When an authentication server finishes its delegated authentication task, it should faithfully include the OTS public key evolving procedure into a public database to ensure that all sessions should be auditable. Hence each of the \( z - 1 \) authentication servers should check the OTS key update procedure of the client by comparing the client’s OTS cache and the historical protocol transcripts recorded in the database to figure out identity fraud attacks. However, using only OTS for authentication is subject to network attacks. This is because an attacker can interfere with the OTS public key update procedure and force the cached signature-chain to be over-long. For example, the attacker can keep initiating sessions on the client-side to generate and cache many new OTS public keys (in the signature chain) without sending back the authentication server’s acknowledgment messages.

To prevent network attackers, we adopt a hybrid approach such that the mutual authentication protocol is achieved by using both long-term symmetric key-based message authentication code and asymmetric key-based one-time signature (OTS). Firstly, most of the protocol messages are protected by the message authentication code computed based on the long-term pre-shared key. The long-term key-based authentication is useful in the IFD. This is because, if the message authentication codes are valid in a session, but the relevant one-time signature is invalid, then it implies that the honest party must have been impersonated before. Secondly, the OTS ensures the following two security properties (as defined in [20]): 1) forward security, which protects the compromised OTS secret keys from affecting the security of previous sessions using uncorrupted keys; 2) backward security, which means that if a party runs the mutual authentication to update the compromised OTS secret key before the adversary uses it in an impersonation attack, then the compromised key would become invalid (due to its one-time feature). Note that the backward security cannot be achieved by using forward secure signature [18]. And OTS is more computationally efficient.

To evaluate the performance of our proposed methods in a real-world scenario, we implemented a two-party system. The server in the system is a laptop, while a Raspberry Pi based OpenPLC is used as a CPS device. We choose to use OpenPLC because the commercial PLCs on the market are all closed-source, and do not allow us to modify their implementations. OpenPLC is an open-source implementation of a programmable logic controller (PLC) [5], which is widely used in the community of CPS security researchers. The experimental results show that our proposed framework is efficient in terms of communication, computation, and memory.

\(^1\)In contrast, we here focus on delegated authentication for machine-to-machine, not password-based setting.
utilization in a cyber-physical system if the parameters are chosen properly.

Our contributions can be summarized as follows:

- A two-party mutual authentication protocol 2MA, which provides weak impersonation detection when the victim is not controlled by the attackers.
- A delegated mutual authentication protocol 4MA, which is built from 2MA and provides the server’s strong identity fraud detection even when the server is a malicious insider.
- A new security model that is proposed to capture the mutual authentication with the server’s strong identity fraud detection.
- The proposed protocols are proven secure in the standard model.

**Organization.** We introduce necessary preliminaries in Section 2. The security model of mutual authentication with identity fraud detection is presented in Section 3. Section 4 presents two mutual authentication protocols of which the second protocol can provide the server’s identity fraud detection. The security of the proposed protocols is analyzed in Section 5. Performance analysis and evaluation results are presented in Section 6. We review the literature related to our work in Section 7. The paper concludes in Section 8.

## 2 PRELIMINARIES

**Notations.** Let $k \in \mathbb{N}$ be the security parameter and $1^k$ be a string that consists of $k$ ones. We denote with $[n] = \{1, \ldots, n\} \subset \mathbb{N}$ the set of integers between 1 and $n$. If $X$ is a set, then $x \xleftarrow{} X$ denotes the action of sampling a uniformly random element from $X$. If $X$ is a probabilistic algorithm, then $x \xleftarrow{} X$ denotes that $X$ is run with fresh random coins and returns $x$. Assuming that $X$ is a set, we denote with $|X|$ the operation to obtain the number of elements in $X$. We may use the operation $X[i]$ to get access to the $i$-th element of $X$. We let the operation $\| X \|$ denote the concatenation of two strings.

In the following, we review the cryptographic primitives [16] used in our constructions.

**Digital Signature Schemes.** We define a (regular) digital signature scheme $\text{SIG}$ with three probabilistic polynomial time (PPT) algorithms ($\text{SIG.Gen}, \text{SIG.Sign}, \text{SIG.Vfy}$). A one-time signature (OTS) scheme is denoted by $\text{OTS.Gen}, \text{OTS.Sign}, \text{OTS.Vfy}$, which is a special case of $\text{SIG}$. We assume that a signature scheme is associated with public and secret key spaces $\mathcal{PKSIG}, \mathcal{SKSIG}$; a randomness space $\mathcal{R}_k$ for key generation, message space $\mathcal{M}$, and signature space $\mathcal{S}$ in the security parameter $k$. We denote the bit-length of space $\mathcal{R}_k$ by $\ell_k$ which is determined by $k$. The algorithms of $\text{SIG}$ are defined as follows:

- $\text{SIG.Gen}(1^k, rs)$: This algorithm takes as input the security parameter $1^k$ and a random value $rs \xleftarrow{} \mathcal{R}$, and outputs a (public) verification key $ok \xleftarrow{} \mathcal{PKSIG}$ and a signing key $sk \xleftarrow{} \mathcal{SKSIG}$.
- $\text{SIG.Sign}(sk, m)$: This is the signing algorithm that generates a signature $\sigma \in \mathcal{S}$ for a message $m \in \mathcal{M}$ using $sk$.
- $\text{SIG.Vfy}(ok, m, \sigma)$: This is the verification algorithm that takes as input a verification key $ok$, a message $m$ and a signature $\sigma$, outputs 1 if $\sigma$ is a valid signature for $m$ under $ok$, and 0 otherwise.

Let $\text{SIG}(sk, m)$ be a signing oracle which on input $m$ returns a signature $\sigma \xleftarrow{} \text{SIG.Sign}(sk, m)$. We use a list $\text{SR}$ to record all tuple $(m, \sigma)$, where $m_1$ and $\sigma_1$ are the input and output of $i$-th $\text{SIG}$ oracle query respectively.

**Definition 1.** For a signature scheme $\text{SIG} = (\text{SIG.Gen}, \text{SIG.Sign}, \text{SIG.Vfy})$ and an adversary $\mathcal{F}$, we define the following experiment:

\[ \text{Exp}_{\text{SIG}, \mathcal{F}}^{\text{sig-cma}}(k, q) := \text{SIG.Gen}(1^k); (m^*, \sigma^*) \xleftarrow{} \text{SIG.Vfy}(ok), \text{where } \text{SIG}(sk, m) \text{ can be asked at most } q \text{ times; Return 1, if SIG.Vfy}(pk, m^*, \sigma^*) = 1, \text{and } (m^*, \sigma^*) \notin \text{SR}; } \text{Output 0 otherwise.} \]

We define the advantage of $\mathcal{F}$ in the above experiment as $\text{Adv}_{\text{SIG}, \mathcal{F}}^{\text{sig-cma}}(k, q) := \text{Pr}[\text{Exp}_{\text{SIG}, \mathcal{F}}^{\text{sig-cma}}(k, q) = 1]$. We say that $\text{SIG}$ is secure against strong existential forgeries $\mathcal{F}$ under adaptive chosen-message attacks (SEUF-CMA), if for all PPT adversaries $\mathcal{F}$ the advantage $\text{Adv}_{\text{SIG}, \mathcal{F}}^{\text{sig-cma}}(k, q)$ is negligible. If $q = 1$ then $\text{SIG}$ is called as a SEUF-CMA secure OTS scheme.

**Collision-Resistant Hash Functions.** Let $\mathcal{CRH} : \mathcal{K}_{\mathcal{CRH}} \times \mathcal{M}$ be a family of keyed-hash functions where $\mathcal{K}_{\mathcal{CRH}}$ is the key space, $\mathcal{M}$ is the message space and $\mathcal{CRH}$ is the hash value space. The public key $h_{\mathcal{CRH}} \in \mathcal{K}_{\mathcal{CRH}}$ defines a hash function, denoted by $\mathcal{CRH}(m)$. On input a message $m \in \mathcal{M}$, this function $\mathcal{CRH}(h_{\mathcal{CRH}}, m)$ generates a hash value $y \in \mathcal{CRH}$. When the hash key $h_{\mathcal{CRH}}$ is obvious from the context, we write $\mathcal{CRH}(m)$ for $\mathcal{CRH}(h_{\mathcal{CRH}}, m)$.

**Definition 2.** For a hash function $\mathcal{CRH} : \mathcal{K}_{\mathcal{CRH}} \times \mathcal{M} \rightarrow \mathcal{CRH}$ and an adversary $\mathcal{H}$, we define the following experiment:

\[ \text{Exp}_{\mathcal{CRH}, \mathcal{H}}^{\mathcal{H}}(k) := h_{\mathcal{CRH}} \xleftarrow{} \mathcal{K}_{\mathcal{CRH}}; (m^*, m) \xleftarrow{} \mathcal{H}(h_{\mathcal{CRH}}); \text{Return 1, if } (m^*, m) \in \mathcal{M} \text{ and } \mathcal{CRH}(m^*, m) = \mathcal{CRH}(m); \text{Output 0 otherwise.} \]

We define the advantage of $\mathcal{H}$ in the above experiment as $\text{Adv}_{\mathcal{CRH}, \mathcal{H}}^{\mathcal{H}}(k) := \text{Pr}[\text{Exp}_{\mathcal{CRH}, \mathcal{H}}^{\mathcal{H}}(k) = 1]$. We say that $\mathcal{CRH}$ is collision-resistant if for all PPT adversary $\mathcal{H}$ the advantage $\text{Adv}_{\mathcal{CRH}, \mathcal{H}}^{\mathcal{H}}(k)$ is negligible.

**Pseudo-Random Functions.** We denote by $\text{PRF} : \mathcal{K}_{\text{PRF}} \times \mathcal{D}_{\text{PRF}} \rightarrow \mathcal{R}_{\text{PRF}}$ a pseudo-random function family, where $\mathcal{K}_{\text{PRF}}$ is the key space, $\mathcal{D}_{\text{PRF}}$ is the domain and $\mathcal{R}_{\text{PRF}}$ is the range of $\text{PRF}$. Let $\text{PList}$ be a list to store the messages queried in the $\text{PRF}$ oracle $\mathcal{F}(k, x)$ which just returns $\text{PRF}(k, x)$.

**Definition 3.** Given a $\text{PRF} : \mathcal{K}_{\text{PRF}} \times \mathcal{D}_{\text{PRF}} \rightarrow \mathcal{R}_{\text{PRF}}$ and an adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$, we define the following experiment:

\[ \text{Exp}_{\text{PRF}, \mathcal{B}}^{\text{PRF-CMA}}(k, q) := \begin{cases} 1, & \text{if } (x, s) \xleftarrow{} \mathcal{PList}, (x_1, s_1) \xleftarrow{} \mathcal{D}_{\text{PRF}}, (s_1, x_1) \xleftarrow{} \mathcal{R}_{\text{PRF}}; \text{if } x \in \text{PList} \text{ then return a failure } 1; \text{ otherwise } V_1^x := \text{PRF}(k, x_1), V_2^s := V_1^x \xleftarrow{} \mathcal{B}_1(k); \text{if } b = b' \text{ then return } 1; \text{ otherwise return } 0. \end{cases} \]

We define the advantage of $\mathcal{B}$ in the above game as $\text{Adv}_{\text{PRF}, \mathcal{B}}^{\text{PRF-CMA}}(k, q) := \text{Pr}[\text{Exp}_{\text{PRF}, \mathcal{B}}^{\text{PRF-CMA}}(k, q) = 1]$, where $q$ is the number of $\mathcal{F}(k, x)$ queries that can be asked by $\mathcal{B}$. The pseudo-random function family $\text{PRF}$ is said to be secure, if for all PPT adversaries $\mathcal{B}$ the advantage $\text{Adv}_{\text{PRF}, \mathcal{B}}^{\text{PRF-CMA}}(k, q)$ is negligible.

## 3 SECURITY MODEL

In this section, we define a security model for mutual authentication with identity fraud detection (MA-IDF) protocols, which is adapted from [1, 7, 26]. In particular, we formulate two security properties of MA-IDF protocols, including mutual authentication (MA) and identity fraud detection (IFD).

**Execution Environment.** Let $u, n, d \in \mathbb{N}$ be positive integers. We consider an environment with multiple parties that include $u$ clients...
and a top authentication servers, where \( u \geq v \geq 2 \). We assume that each client has a unique identity \( C \) and each server is uniquely identified by an identity \( S \). We will denote with \( I \) the general identity for a party such that \( I \in \{C, S\} \). Each party \( I \) may carry out at most \( \rho \in \mathbb{N} \) sessions that are simulated by a set of oracles \( \Pi^x_{I} \): \( i \in [\ell], s \in [\rho] \), where \( \ell = u \) and \( \rho = d \) if \( I \) is client, otherwise \( \ell = v \) and \( \rho = w_{-}-.d+.e+.d. \). We assume that an oracle would terminate a session with acceptance if and only if it has sent or received the last protocol message and all received messages are valid according to the protocol specification; otherwise it would terminate with rejection. Meanwhile, each client can only sequentially execute the protocol, i.e., the \( s + t \)-th oracle can be activated if and only if the \( s \)-th oracle of that party is terminated. The authentication server \( S \) can only sequentially run oracles with a specific client but execute oracles with different clients concurrently. That is, \( \Pi^x_{S} \) and \( \Pi^y_{S} \) run simultaneously with a different communication partner. For entity authentication purpose, we assume that each party has a secret/public key pair \( (sk_{I}, pk_{I}) \) which is evolvable (so that each oracle may have a distinct secret/public key pair), but we do not require the client and the authentication server has the same type of secret/public key pair, e.g., the authentication server’s secret/public key pair \( (sk_{S}, pk_{S}) \) may have many sub-secret/public key pairs each of which is used to authenticate itself to a specific client. Each client can have access to its own secret key, all public keys of other parties, and any other public information. Moreover, each oracle \( \Pi^x_{I} \) maintains a list of independent internal state variables including:

- \( \Phi^x_{I} \) - execution state \( \Phi^x_{I} \in \{\text{accept, reject}\} \);
- \( \pi^x_{I} \) - identities of the communication partners;
- \( I^x_{I} \) - messages ordered sent and received by \( \Pi^x_{I} \).

Here we review a notion regarding the partnership of two oracles during online interaction. We denote with \( PTF : (I(D), s, \pi^x_{I(D)}, I^{x}_{I(D)}) \rightarrow (I(D), t) \) a partner function \cite{7} which takes input the internal states \( \pi^x_{I(D)} \) and \( I^{x}_{I(D)} \) of an oracle \( \Pi^x_{I(D)} \) and outputs its partner oracle (session) \( \Pi^y_{I(D)} \), such that \( I(D) \in \pi^x_{I(D)} \). We realize the partner function \( PTF \) which outputs \( (I(D), t) \) if and only if all the following conditions hold: i) both \( \Pi^x_{I(D)} \) and \( \Pi^y_{I(D)} \) have sent their last protocol message without rejection; ii) \( I(D) \in \pi^x_{I(D)} \) and \( I(D) \in \pi^y_{I(D)} \); iii) \( I^{x}_{I(D)} \subseteq I^{y}_{I(D)} \) or \( I^{y}_{I(D)} \subseteq I^{x}_{I(D)} \); iv) \( \Pi^y_{I(D)} \) is unique for satisfying all above conditions.

Adversarial Model. We define an active adversary \( \mathcal{A} \) as a probabilistic polynomial time (PPT) algorithm which can ask the following queries:

- Send \( (I(D), s, m) \): \( \mathcal{A} \) can use this query to send any message \( m \) to an initialized oracle \( \Pi^x_{I(D)} \). The oracle will respond to the next message \( m^* \) (if any) to be sent according to the protocol specification and its internal states.
- Corrupt \( (I(D)) \): This query returns the current secret key \( sk_{I(D)} \) to \( \mathcal{A} \). We say that the party \( I(D) \) submitted to this query is corrupt. Once a party \( I(D) \) is corrupted, no oracle of \( I(D) \) will be initialized any more.

Security Definitions. In the following, we formulate the security properties for mutual authentication protocols, and, especially for server identity fraud detection. To formulate identity fraud detection, we first define a class of clients that are deceived by a compromised authentication server.

Deceived Clients. We say a client \( C_{i} \) is deceived if there is an oracle \( \Pi^x_{C_{i}} \) that accepts without a partner oracle at the uncompromised client \( C_{j} \in \pi^x_{C_{i}} \) (if any) which will be called an impersonated client. This issue also implies an identity fraud event.

Direct Delegator. We say an authentication server \( S_{j} \) is a direct delegator for \( C_{i} \) in an oracle \( \Pi^x_{C_{i}} \) if \( S_{j} \in \pi^x_{C_{i}} \) and \( S_{j} \) is supposed to directly verify the authentication proof (e.g., a signature) of \( C_{i} \). For security definition, we first define a security experiment that is played between an adversary \( \mathcal{A} \) and a challenger \( C \) based on an MA-IFD protocol \( \Sigma \). Let \( MP \) be a variable denoting one of the security properties in \{MA, MA-IFD\}, where MA is mutual authentication, and MA-IFD is mutual authentication with (server) identity fraud detection.

Security Experiment EXP_{\mathcal{A}, \Sigma} (q, k, MP): the security experiment is represented as a game between a challenger \( C \) and an adversary \( \mathcal{A} \) based on a target AKP protocol \( \Sigma \), where the following steps are performed: i) Setup: \( C \) initializes the execution environment by realizing the oracles \( \{\Pi^x_{C_{i}} : i \in [u], s \in [d]\} \) for clients and \( \{\Pi^x_{S_{j}} : j \in [v], t \in [2u+d]\} \) for authentication servers. \( C \) returns all identities and public keys to \( \mathcal{A} \); ii) Query Phase: \( \mathcal{A} \) may issue a polynomial number of Send and Corrupt queries; iii) Finalization: \( \mathcal{A} \) terminates the game and outputs a tuple \( (I(D)^{s}, s^{*}, t^{*}) \). The experiment outputs 1 if and only if \( \Pi^x_{I(D)} \) accepts and one of the following conditions is held:

- \( C: MP \in \{\text{MA, MA-IFD}\} \), and all parties in \( \pi^x_{I(D)} \) are uncompromised, and there is a party \( I(D) \in \pi^x_{I(D)} \) such that \( I(D) \) has no unique partner oracle to \( \Pi^y_{I(D)} \);
- \( C: MP \in \{\text{MA-IFD}\} \), and all the following conditions are satisfied: i) \( \mathcal{A} \) only asked \( q \) times Corrupt query to authentication servers; ii) \( C_{i} \) is deceived in \( \Pi^{x}_{I(D)} \) such that \( t^{*} < s^{*} \); iii) \( C_{i} \)'s direct delegator \( S_{c} \in \pi^x_{C_{i}} \) is not corrupted and \( S_{c} \) has a partner oracle \( \Pi^x_{S_{c}} \) to \( \Pi^y_{S_{c}} \), and the other client \( C_{j} \in \pi^{y}_{I(D)} \) is not corrupted before \( \Pi^y_{S_{c}} \) accepts.

Definition 4 (MA-IFD Security). We define the advantage of an adversary \( \mathcal{A} \) against a MA-IFD protocol \( \Sigma \) with parameter \( q < v \) as \( Adv_{\mathcal{A}, \Sigma} (q, k, MP) = Pr[EX_{\mathcal{A}, \Sigma} (q, k, MP) = 1] \). We say that MA-IFD protocol \( \Sigma \) is secure, if for all PPT adversaries \( \mathcal{A} \), the advantage \( Adv_{\mathcal{A}, \Sigma} (q, k, MP) \) is negligible.

4 A MUTUAL AUTHENTICATION FRAMEWORK WITH IDENTITY FRAUD DETECTION

In this section, we will introduce a new authentication framework that can provide identity fraud detection (IFD) capability. We start

\footnote{This IFD rule indicates that the honest authentication server \( S_{e} \) fails to identify that \( C_{i} \) is deceived.}
with the construction for two-party mutual authentication (2MA), which relies on both PRF and OTS. The proposed 2MA can provide weak client identity fraud detection. The 'weak' means that the identity fraud event is caused by outsider attackers who compromise the authentication keys without controlling the victim’s device. Next, we show how to generically leverage on the proposed 2MA protocol as a building block to construct a MA-IFD protocol 4MA that can provide strong server identity fraud detection (i.e., even if the compromised server is an insider attacker which takes control of the server). Our MA-IFD protocol 4MA involves four parties including two clients and two authentication servers (AS) for simplicity. Each client’s authentication is outsourced to an authentication server. In our protocol constructions, we include three standard building blocks: i) a one-time signature scheme OTS = (OTS.Gen, OTS.Sign, OTS.Vfy); ii) a regular (multiple-time) signature scheme SIG = (SIG.Gen, SIG.Sign, SIG.Vfy); iii) a collision-resistant hash function CRH which has a random hash key \( h_{CRH} \leftarrow \mathcal{K}_{CRH} \); and iv) a pseudo-random function family PRF.

4.1 Two-party Mutual Authentication with Weak Identity Fraud Detection

**Design Rationale.** Our 2MA protocol is adapted from the traditional challenge-response paradigm with several important modifications. Essentially, we combine symmetric key-based PRF with asymmetric key-based OTS for authentication. First, the pre-shared symmetric key can be compromised from either key sharer, so it is hard to identify the party who leaks the key. To mitigate the drawback of symmetric key-based authentication and achieve identity fraud detection, we leverage OTS for authentication that each OTS key pair needs to be updated to a new one in each session. If the OTS key update procedure is abnormal, then there must be some identity fraud events.

Since the OTS secret key can be used only one-time, we adopt a cached signature-chain technique to update the OTS public key. Specifically, a party commits its next \((n+1)\) OTS public key in the signature generated based on the current OTS secret key, so that the OTS public keys of a party are chained along with the generation of signatures. In particular, the transmitted signatures may be lost due to network failures. Each party should cache the signature relevant data (e.g., the public key used to verify the signature and sign hashed message) until it is verified. A party can send its cached data for verification in the next session. That is, the current unverified public key of a client can be validated by checking whether there is a valid signature-chain starting from using the secret key of the last verified public key (see more details in Algorithm 1). To track the protocol executions, the authentication server must record the transcript of protocol messages in a session, which is used later for identity fraud detection. According to the recorded transcript and the evolving OTS public key, a client can identify whether it has been impersonated before.

In practice, an adversary who eavesdrops on the network may disrupt the communication so that the cached signature-chain may grow very fast on the client-side. To deal with this problem, we rely on the symmetric key-based authentication to guarantee that network adversary cannot manipulate the exchanged messages (including nonce and one-time signatures). Meanwhile, the result of symmetric key-based authentication can help a party identify the key compromise events. Namely, the proposed protocol can ensure that only the compromised party leads to its communication partner’s signature-chain being over-long.

**Protocol Description.** We let 2MA(I \( \mathcal{D}_a, S_c, Al \mathcal{D}_a, Al \mathcal{S}_c, I \mathcal{D}_a \)) denote a protocol instance of our two-party MA protocol executed between two parties \( I \mathcal{D}_a \) and \( S_c \) with auxiliary inputs \( Al \mathcal{D}_a \) and \( Al \mathcal{S}_c \) from them respectively. Each execution of 2MA outputs the protocol transcript \( T_{I \mathcal{D}_a} \) and \( T_{S_c} \) from the view of \( I \mathcal{D}_a \) and \( S_c \) respectively, where \( I \mathcal{D}_a \) can be either client or server. Here we mainly depict the protocol from the view of the client, i.e., \( I \mathcal{D}_a = \mathcal{C}_a \) for instance. As for \( I \mathcal{D}_a = \mathcal{S}_c \), the protocol execution is similar, but no OTS public key update is required since both parties will use long-term keys for authentication. We denote by \( \ast \in \{\text{No-Client-Partner}, \text{Client-Partner-Auth}\} \) the authentication assertion issued by an authentication server, where No-Client-Partner means there is no partnered client, and Client-Partner-Auth means the partner’s identity is authenticated. For 2MA, \( \ast \) is always No-Client-Partner. If \( \ast = \text{No-Client-Partner} \), then the communication entities include the client and its direct delegator. Let \( R_N = \{0,1\}^\ell \) be a nonce space, where \( \ell \) is the bit-length determined by \( \kappa \).

We describe the proposed two-party protocol 2MA as follows.

**Parties.** In our construction, we consider that there are \( u \) clients which could be either users or devices (e.g., programmable logic controller), and \( v \) authentication servers (AS) (which form a pool). Each client is identified by a unique identifier \( \mathcal{C}_a \), for \( a \in [u] \), and each authentication server has an identity \( \mathcal{S}_c \), for \( c \in [v] \). For the IFD purpose, we will let these authentication servers work in turns for a client (as mentioned in Section 1).

**Setup.** To track the OTS public key update procedure, we assume each party exploits a queue data structure called ‘SCache’, such that each item in this list is a tuple \( (I \mathcal{D}_a, pk^{ots}_a, h, pk^{ots,x}_a, \sigma) \), where \( \sigma := \text{OTS.Sign}(sk^{ots}_a, h, pk^{ots,x}_a) \) is a signature of the hashed message \( h \) concatenating with the public key \( pk^{ots,x}_a \); \( pk^{ots}_a \) is the public key used for verifying the current signature, and \( I \mathcal{D}_a \) is the party which is supposed to verify the signature (i.e., the intended communication partner of the owner of \( pk^{ots}_a \)). We assume that an element can be only popped up from the queue’s head and pushed into the tail of the queue.

Let \( \text{ClearSC}(\text{parm}, \text{SCache}) \) be a function that takes as input a cache \( \text{SCache} \) and a parameter \( \text{parm} \in \{pk^{ots}, I \mathcal{D}_a\} \), and it does the following steps: i) If \( \text{parm} \) is a public key \( pk^{ots}_a \), then it deletes all tuples before the one containing \( pk^{ots}_a \) from \( \text{SCache} \); ii) If \( \text{parm} \) is an identity \( I \mathcal{D}_a \), it deletes all tuples identified by \( I \mathcal{D}_a \) from \( \text{SCache} \).

The authentication server \( \mathcal{S}_c \) first generates a set of pre-shared symmetric keys \( \{\mathcal{K}_{S_c, \mathcal{C}_a}\}_{a \in [u]} \) and securely distributes them to the corresponding client over a secure channel, e.g., the client \( \mathcal{C}_a \) has the key \( \mathcal{K}_{S_c, \mathcal{C}_a} \). Second, \( \mathcal{S}_c \) generates OTS secret/public pairs, which are used for authenticating itself to each client, i.e., for \( a \in [u] \) it runs \((sk^{ots}_{S_c, \mathcal{C}_a}, pk^{ots}_{S_c, \mathcal{C}_a}) \leftarrow \text{OTS.Gen}(1^\kappa, r_{S_c, \mathcal{C}_a}), \) where \( r_{S_c, \mathcal{C}_a} \leftarrow \mathcal{R}_N \). Meanwhile, the public key \( pk^{ots}_{S_c, \mathcal{C}_a} \) will be sent to the client \( \mathcal{C}_a \).

Third, \( \mathcal{S}_c \) generates a long-term secret/public pair \((sk^{sig}_{S_c}, pk^{sig}_{S_c}) \leftarrow \)
SIG.Gen(\(1^k\)). In particular, the authentication server \(S_c\) would keep variables:

- \([S\text{Cache}_{S_c}, c_a]_{c \in \{a\}}\): a set of queues, each of which stores the one-time signature relevant data of \(S_a\) in sessions communicating with the client \(C_a\). Each used OTS public key of the signee \(S_c\) which is not checked by the intended verifier \(C_a\) (i.e., without receiving the acknowledgement from \(C_a\)), will be stored into the corresponding \(S\text{Cache}_{S_c}, c_a\).

Each client \(C_a\) initiates the protocol by generating a secret/public pair \((sk^{\text{ots}}_{C_a}, pk^{\text{ots}}_{C_a}) \leftarrow \text{SIG.} \text{Gen}(1^k, s_{C_a}), \) where \(s_{C_a} \leftarrow R_s\). \(C_a\) initializes and locally keeps the following variables:

- \(\text{AS_PK}_{c_a} = \{pk^{\text{ots}}_{S_i, c_a}\}_{c \in \{a\}}\): a set of authentication server’s OTS public keys. Each public key will be used to verify the signature from the corresponding authentication server \(S_c\).

- \([C_a, S_{c, \text{Available}}]_{c \in \{a\}}\): a set of variables regarding available status of authentication servers, such that \(C_a, S_{c, \text{Available}} \in \{0, 1\}\), where 1 denotes that \(S_{c, \text{Available}}\) is available for the client \(C_a\) and 0 otherwise. These variables are initially set to be 1. Note that when \(C_a, S_{c, \text{Available}} = 0\), another party \(C_i \neq C_a\) can still choose \(S_c\) for authentication if \(C_a, S_{c, \text{Available}} = 1\).

- \(\text{ASQueue}_{c_a}\): a queue that stores the identities of direct authentication servers, which are currently selected to do both authentication and identity fraud detection for \(C_a\). We implicitly assume that the queue is initialized with size \(z = |\text{ASQueue}_{c_a}|\) by a queue initialization algorithm, and \(z\) available authentication servers’ identities are pushed into this queue initially.

- \(\text{IFDCache}_{c_a}\): a queue that stores the information regarding the OTS key update procedure, which is used for identity fraud detection. Each record in \(\text{IFDCache}\) is a tuple in the form of \((s_c, pk^{\text{ots}}_{S_c, c_a}, h_{S_c, c_a}, \sigma_{S_c, c_a}, pk^{\text{ots, nx}}_{S_c, c_a}, S\text{Cache}_{S_c, c_a}, ast_{S_c, c_a})\), where \(S_c \in \text{ASQueue}_{c_a}\), and \(pk^{\text{ots}}_{S_c, c_a}\) is the public key used to verify the received signature \(\sigma_{S_c, c_a}, h_{S_c, c_a}\) is the signed hash message, and \(pk^{\text{ots, nx}}_{S_c, c_a}\) is supposed to be the next OTS public key of \(S_c\) for authenticating to \(C_a\).

- \(S\text{Cache}_{c, c_a}\): OTS signature relevant data kept by \(C_a\), which has the data structure described above.

In addition, we assume that there is a public database \(\mathcal{PDB}\), such that each record in \(\mathcal{PDB}\) is a tuple \((s_c, h, T, \sigma_{c_a})\), where \(s_c\) is the authentication server which submitted the record, \(h\) is a hash value related to \(T\), \(T\) is a transcript including all protocol messages sent and received by \(s_c\) in a session, and \(\sigma_{c_a}\) is a signature \(\sigma_{c_a} = \text{SIG.} \text{Sign}(sk_{S_c}, T)\). We assume that an authentication server inserts new items into \(\mathcal{PDB}\).

**Authentication.** The online authentication steps are shown in Figure 2. Two parties would first generate two new OTS public keys respectively, and exchange two nonces \((N_{C_a}, N_{S_c, c_a})\), where \(S_c\) is the last the authentication server in the queue \(\text{ASQueue}_{c_a}\). Meanwhile, \(S_c\) should authenticate the identities of participants and exchange nonces with a message authentication code \(A_1\) generated by PRF based on the pre-shared key \(K_{S_c, c_a}\) which convinces the client regarding the existence of \(S_c\) in the corresponding session.

Next, in addition to signing the protocol transcript of messages exchanged, the client \(C_a\) would sign the data in \(\text{IFDCache}_{c_a}\) and \(S\text{Cache}_{c, c_a}\), which include the used OTS public keys from both client and authentication servers respectively. Namely, \(C_a\) generates the signature \(\sigma_{c_a} = \text{OTS.} \text{Sign}(sk^{\text{ots}}_{C_a}, h_{C_a}, pk^{\text{ots, nx}}_{C_a})\), where \(h_{C_a}\) := \(CRH(T_{C_a})\) and \(T_{C_a} = T_{C_a}||S_{C_a}||N_{C_a}||\text{Alice}||N_{S_c, c_a}||\text{Alice}||N_{S_c, c_a}||A_1\) \(\text{||} pk^{\text{ots}}_{C_a}||S\text{Cache}_{C_a}||\text{IFDCache}_{C_a}||pk^{\text{ots, nx}}_{C_a}||pk^{\text{ots, nx}}_{S_c, c_a}\). The client would also record the tuple \((pk^{\text{ots}}_{S_c, c_a}, h_{S_c, c_a}, pk^{\text{ots, nx}}_{S_c, c_a}, \sigma_{S_c, c_a})\) into \(S\text{Cache}_{S_c, c_a}\). In case of any message loss, so that \(S_c\) can show the OTS public key evolving procedures recorded in \(S\text{Cache}_{S_c, c_a}\) next time. The signature \(\sigma_{c_a}\) would also be protected by a message authentication code \(A_2\) generated based on \(K_{S_c, c_a}\) and PRF. Then, the signature \(\sigma_{c_a}, C_a\)’s current public key \(pk^{\text{ots}}_{S_c, c_a}\) the cached client signature relevant data set \(S\text{Cache}_{S_c, c_a}\), the cached servers’ signature relevant data set \(\text{IFDCache}_{S_c, c_a}\), its next (nx) public key \(pk^{\text{ots, nx}}_{S_c, c_a}\), the latest OTS public key of \(pk^{\text{ots}}_{S_c, c_a}\) stored at \(S_a\), and the message authentication code \(A_2\), are sent to the authentication server \(S_c\).

Upon receiving the message \(m_3, S_c\) would first check the message authentication code \(A_2\). If \(A_2\) is valid, then \(S_c\) runs the signature-chain verification algorithm \(\text{SChainVfy}\) which is defined by Algorithm 1, to verify the signature based on the data cached in \(S\text{Cache}_{S_c}\). If \(C_a\)’s signature is valid and the corresponding public key is recorded correctly, then \(\text{SChainVfy}\) returns 1. After this, \(S_c\) would sign the hash of the protocol transcript \(T_{S_c, c_a} = T_{C_a}||\sigma_{c_a}||\text{\text{||} pk^{\text{ots}}_{S_c, c_a}||S\text{Cache}_{S_c, c_a}||pk^{\text{ots, nx}}_{S_c, c_a}||ast_{S_c, c_a}\)\). Next, \(S_c\) would clear its signature cache by running \(\text{ClearSC}(pk^{\text{ots}}_{S_c, c_a}, S\text{Cache}_{S_c, c_a})\), according to the public key \(pk^{\text{ots}}_{S_c, c_a}\) stored at \(C_a\), i.e., all tuples before the one containing \(pk^{\text{ots}}_{S_c, c_a}\) will be deleted from \(S\text{Cache}_{S_c, c_a}\). Furthermore, \(S_c\) would push the tuple \((pk^{\text{ots}}_{S_c, c_a}, h_{S_c, c_a}, pk^{\text{ots, nx}}_{S_c, c_a}, \sigma_{S_c, c_a})\) into \(S\text{Cache}_{S_c, c_a}\), and update its current OTS public key \(pk^{\text{ots}}_{S_c, c_a}\) to the next one \(pk^{\text{ots, nx}}_{S_c, c_a}\). Finally, \(\sigma_{S_c, c_a}\) and its relevant message authentication code \(A_3\), and the messages in \(T_{S_c, c_a}\) which are not known by \(C_a\) are sent to \(C_a\) in \(m_4\). Upon receiving \(m_4\), the client \(C_a\) verifies \(A_3\) and \(\sigma_{S_c, c_a}\) analogously.

**Post-acceptance.** This phase is shown in Figure 2. For identity fraud detection, we need first to set the available status of \(S_c\) at \(C_a\) to be 0, i.e., \(C_a, S_{c, \text{Available}} = 0\). Before confirming that \(S_c\) is not a cheater, it should be no longer selected to authenticate for \(C_a\).
\[ C_a \]

would also push the tuple \((S_c, p_{S_c,C_a}^{OTS}, h_{S_C,C_a}, \sigma_{S_C,C_a}, p_{S_c,C_a}^{OTS, \text{NX}}, \text{SCache}_{S_c,C_a})\) into \text{IFDCache}_{C_a}, which reflects the usage of \(S_c\)'s secret key. The contents of \text{IFDCache}_{C_a} will be checked by other authentication servers later. Meanwhile, \(C_a\) would pop up the first authentication server \(S_y\) and delete all its related data in \text{IFDCache}_{C_a} and \text{SCache}_{C_a} since its cached data must have been verified by \(z - 1\) authentication servers (if no one has reported the identity fraud event about it). Eventually, \(C_a\) randomly selects an authentication server \(S_x\) whose available status \(C_a, S_x, \text{Available} = 1\).

On the server side, \(S_c\) is supposed to faithfully store its protocol transcript \(T_{S_c,C_a}\) into \text{PDB} together with a signature \(\sigma_{S_c}\) that is generated based on its long-term secret key, i.e., the tuple \((S_c, h_{S_c,C_a}, \sigma_{S_c,C_a}, p_{S_c,C_a}^{OTS, \text{NX}}, \text{SCache}_{S_c,C_a})\) is stored into \text{PDB}. Therefore, any other parties (e.g., the next authentication server selected for authenticating \(C_a\)) can check the honesty of \(S_c\) (e.g., as done by Algorithm 2 which is discussed later).

**Weak Client Identity Fraud Detection.** If the client \(C_a\)'s secret keys (including OTS secret key and pre-shared key \(K_{S_c,C_a}\)) are compromised by an attacker \(\mathcal{A}\), then \(\mathcal{A}\) can impersonate \(C_a\). If \(C_a\) is impersonated but not controlled by \(\mathcal{A}\), then the most recent OTS public recorded in \text{PDB} must be distinct to the one kept by \(C_a\), which means that the real \(C_a\) cannot be authenticated by an authentication server anymore. In particular, if an uncorrupted authentication server received a valid \(A_2\) but the underlying signature \(\sigma_{C_a}\) (authenticated by \(A_2\)) is invalid, then \(S_c\) would know that \(C_a\) must have been impersonated before. Similar detection approaches can also be applied to the authentication server. However, the above IFD approach can only resist outsider attackers who do not control the device of a victim (either client or server). An attacker who controls the victim's device would not leave out any proof for IFD (i.e., inconsistent OTS public key), because all OTS keys would be updated and stored in the device faithfully by the attacker.

**4.2 Delegated Mutual Authentication with Strong Server Identity Fraud Detection**

In this section, we focus on constructing a MA-IFD protocol 4MA by generically leveraging 2MA, which can provide strong server identity fraud detection.
DESIGN RATIONALE. In 4MA, the authentication between two clients \( C_a \) and \( C_b \) are delegated by authentication servers \( S_a \) and \( S_b \) respectively, so that the two-party mutual authentication protocol instances 2MA(\( C_a, S_a, A_{L_a,C_a}, A_{L_s,S_a} \)), 2MA(\( S_a, S_b, A_{L_a,S_b}, A_{L_s,S_b} \)) and 2MA(\( C_b, S_d, A_{L_b,C_b}, A_{L_s,S_d} \)) are executed. Meanwhile, the 2MA instance between client and authentication server is run based on OTS, but the 2MA instance between two authentication servers is run based on regular multiple-time signature scheme (without SCache). Recall that, in 2MA, each party can specify auxiliary input \( A_i \) to run the protocol. Therefore, we can specifically define the contents of auxiliary inputs to glue up the 2MA instances to eventually yield 4MA. Namely, we leverage on \( A_i \) to convey the intended partner’s identity and nonce generated in 2MA instances so that each party can uniquely have a partner session at each intended communication partner. Furthermore, we will add a few protocol steps to achieve identity fraud detection capability.

PROTOCOL DESCRIPTION. The setup and the post-acceptance procedures are identical to that of 2MA. The protocol execution is shown by Figure 3, and the protocol messages are listed in Table 1. In the protocol execution of 4MA, we only slightly modify the timing of the protocol messages of 2MA instances and the assertion variables as follows:

- After first two protocol steps in 2MA(\( S_c, S_d, A_{L_s,S_d}, A_{L_s,S_d} \)), two authentication servers suspend to wait for the one-time signatures and the relevant message authentication codes from the clients respectively (before sending \( m_4 \)). If a server received either an invalid signature or invalid message authentication code from the client, then it rejects the session 2MA(\( S_c, S_d, A_{L_s,S_d}, A_{L_s,S_d} \)).
- If \( S_x \) for \( x \in \{ a, d \} \), the corresponding party rejects the session 2MA(\( S_c, S_d, A_{L_s,S_d}, A_{L_s,S_d} \)), then \( S_x \) sets the authentication assertion as \( a_{s_x, c_y} = \text{No-Client-Partner} \), where \( C_y \) for \( y \in \{ a, b \} \) is the client that is directly delegated by \( S_x \); Otherwise, \( S_x \) sets \( a_{s_x, c_y} = \text{Client-Partner-Auth} \).
- If \( S_c \) happens to be equivalent to \( S_d \), then the session 2MA(\( S_c, S_d, A_{L_s,S_d}, A_{L_s,S_d} \)) does not need to execute.

STRONG SERVER IDENTITY FRAUD DETECTION. Our goal is to enable a client to provide a ‘proof’ regarding any identity fraud event that a compromised authentication server, which can be controlled by attackers (e.g., malicious insiders), impersonated as the honest client to communicate with another honest client. Recall that in the post-acceptance phase, each authentication server should sign and insert their protocol transcript into the public database \( \mathcal{DB} \) so that all other authentication servers can verify the protocol executions later. Note that from the protocol transcript, any party can ‘reply’ the corresponding 2MA instance. If the most recent protocol transcript in \( \mathcal{DB} \) involving \( S_c \) and \( S_b \) is valid then the up-to-date OTS public key \( p_{OTS,S_c} \) is recorded and valid as well. We stress that the identity fraud cache IFDCache\( _{C_{\text{b}}}(x) \) can provide a ‘proof’ if \( C_{\text{b}} \) is deceived by an authentication server before. Generally speaking, an honest authentication server can identify an identity fraud event from IFDCache\( _{C_{\text{b}}}(x) \) by checking whether the relevant transcript has been correctly recorded in \( \mathcal{DB} \) and the sessions implied by the corresponding transcript have been faithfully executed (e.g., each session has a unique partner session). We stress that the current OTS public key \( p_{OTS,S_c} \) of \( S_c \) stored at either the client \( C_a \) must be checked by other \( z \leq 1 \) authentication servers in ASQueue before it can be used to verify the next signature (i.e., a successful OTS public key update in the post-acceptance phase of 2MA). Among those \( z \leq 1 \) authentication servers, we assume that at least one of them is compromised, so that the IFD works if and only if the adversaries corrupted at most \( z \) servers. By applying our IFD procedure, the OTS public key \( p_{OTS,S_c} \) in a tuple \( (S_c, p_{OTS,S_c}, h_{S_c,S_d}, I_{S_c,S_d}, \sigma_{S_c,S_d}, p_{OTS,RX,S_c}, \mathcal{SCache}_{S_c,S_d}) \) from IFDCache\( _{C_{\text{b}}}(x) \) should exist in \( \mathcal{DB} \) as well, otherwise \( S_c \) must have behaved dishonestly. Similarly, the hash value of the protocol transcript, which involves the usage of \( p_{OTS,S_c} \), should be recorded in \( \mathcal{DB} \) if \( S_c \) is honest. However, for a malicious \( S_c \) which tries to impersonate \( C_a \), it cannot write a valid protocol transcript involving \( C_a \) since without knowing \( C_a \)’s secret key \( S_c \) is unable to generate a valid (unforgeable) signature for executing the protocol. This is why we need the client to provide a proof.

To quickly identify a partner session, we let GetSID be a function which takes as input a transcript \( T_{S_c,S_d} \) and outputs a session identifier \( \text{SID} = C_a || S_c || S_d || I_{S_c,S_d} || \sigma_{S_c,S_d} || p_{OTS,RX,S_c} || \mathcal{SCache}_{S_c,S_d} \) where the identities are sorted in lexicographic order. If two sessions recorded by transcripts \( T_{S_c,S_d} \) and \( T_{S_d,S_c} \) (respectively) are partnered then it must have that \( \text{GetSID}(T_{S_c,S_d}) = \text{GetSID}(T_{S_d,S_c}) \).

More specifically, to achieve identity fraud detection, when \( S_c \) receives a valid signature from \( C_a \), it runs the Algorithm 2 which takes as input IFDCache\( _{C_{\text{a}}}(x) \), and output either a tuple \( (S_c, p_{OTS,S_c}, h_{S_c,S_d}, I_{S_c,S_d}, \sigma_{S_c,S_d}, p_{OTS,RX,S_c}, \mathcal{SCache}_{S_c,S_d}) \) related to an identity fraud event or \( 0 \) meaning that no identity fraud event is found. If an identity fraud event is found, then the deceived client \( C_c \) and the cheater \( S_d \) should stop working before they are fixed. Here, we send the IFDCache along with the authentication (for simplicity), that may eat some bandwidth for online authentication. But we stress that IFDCache can be sent to the next intended authentication server \( S_x \) right after the last session is terminated and before the current (newly updated) OTS key of the client is used. In the meantime, the client should use the pre-shared key \( K_{S_x,C_x} \) to protect the integrity of IFDCache\( _{C_{\text{b}}}(x) \) (based on a message authentication code scheme) when transmitting it.4 In this scenario, the client can even trigger the identity fraud detection immediately after a session is done.

5 SECURITY ANALYSIS

5.1 Security of 2MA

THEOREM 1. We assume that \( t_c \leq t_s \), the one-time signature OTS is SEUF-CMA secure, the hash function CRH is collision-resistant, the pseudo-random function family PRF is secure, and the size of ASQueue has a size such that \( 3z \leq z \leq 0 \). Then the proposed two-party mutual authentication protocol 2MA is secure with Adv_{2MA,\mathit{EF}}(z - 2, k, \alpha) \leq \frac{(3u \cdot v - d)^2}{2e} + Adv_{\mathit{CRH}, \mathcal{K}}(2^e) + \mathit{Adv}_{\mathit{PRF}, B}(k, 3u \cdot v - d + v \cdot d) + 2u \cdot v \cdot d + \mathit{Adv}_{\mathit{SEUF-CMA}, F}(k, 1).

We prove this theorem via a sequence of games following [35]. In the proof, we first change the games to exclude the collision

\[ \frac{1}{2} \text{we stress that only the use of the pre-shared key (without the OTS signature) is enough to protect IFDCache}_{S_c} \text{ if } S_c \text{ is honest (not compromised) then the pre-shared key is uncompromised either. For a compromised } S_c, \text{ the integrity of IFDCache}_{S_c} \text{ is useless.} \]
Table 1: Message Transcripts and Auxiliary Inputs.

<table>
<thead>
<tr>
<th>Variables</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T_{C_a} )</td>
<td>( 2MA(C_a, S_c, A_{id}, A_{k_a} \rightarrow S_c) )</td>
</tr>
<tr>
<td>( S_c )</td>
<td>( 2MA(S_c, S_d, A_{id}, A_{id} \rightarrow S_d) )</td>
</tr>
<tr>
<td>( T_{S_d, C_b} )</td>
<td>( 2MA(C_b, S_c, A_{id}, A_{k_a} \rightarrow S_c) )</td>
</tr>
</tbody>
</table>

5.2 Security of 4MA

Theorem 2. We assume that \( t_\epsilon \leq t_\epsilon \), both signature schemes OTS and SIG are SEUF-CMA secure, the hash function CRH is collision-resistant, and the size of ASQueue has a size \( z \) such that \( 2 < z < v \). Then the proposed MA-IFD protocol 4MA is secure with \( Adv_{4MA, A}(\epsilon) \leq \frac{(3u \cdot v \cdot d)^2}{2^{64-1}} + 2 \cdot Adv_{CRH, H}(k) + 2u \cdot v \cdot Adv_{IND-CMA}(k, 3u \cdot v \cdot d + v \cdot d) + 2u \cdot v \cdot d \cdot Adv_{seal-cma, OTS, F}(k, 1) + 2u \cdot v \cdot Adv_{seal-cma, SIG, F}(k, 2u \cdot v \cdot d) \).

We divide adversaries into two categories to analyze the authentication and key exchange respectively:

1. **Authentication adversary** can succeed in breaking the MA property, i.e., condition C1 holds in the security experiment;
2. **IFD adversary** can fail the identity fraud detection, i.e., condition C2 holds in the security experiment.

To prove Theorem 2, we present two lemmas. Each analyzes one of the security properties of the proposed protocol. Specifically, Lemma 1 bounds the success probability \( \epsilon_{auth} \) of authentication adversaries, and Lemma 2 bounds the success probability \( \epsilon_{ifd} \) of IFD adversaries. Then we have \( Adv_{4MA, A}(\epsilon) \leq \epsilon_{auth} + \epsilon_{ifd} \). In the following, we just briefly introduce the idea of the proof.

**Lemma 1.** For any PPT adversary \( A \), the probability that there exists an oracle \( \Pi_{\Pi^*}^{T_i} \) accepting with satisfying condition C1 is at most \( \epsilon_{auth} \leq \frac{(3u \cdot v \cdot d)^2}{2^{64}} + Adv_{CRH, H}(k) + 2u \cdot v \cdot Adv_{IND-CMA}(k, 3u \cdot v \cdot d + v \cdot d) + 2u \cdot v \cdot d \cdot Adv_{seal-cma, OTS, F}(k, 1) + 2u \cdot v \cdot Adv_{seal-cma, SIG, F}(k, 2u \cdot v \cdot d) \).

The proof is similar to that of Theorem 1. Here we just present the proof idea for simplicity. The security reduction concerning randomness collision is identical to that in the proof of Theorem 1. The main difference is that there are two types of signature schemes.

**Lemma 2.** For any PPT adversary \( A \), the probability that \( A \) can output an oracle \( (C_i^*, s^*) \) satisfying condition C2 is at most \( \epsilon_{ifd} \leq \frac{3u \cdot v \cdot d}{2^{64}} + Adv_{CRH, H}(k) + 2u \cdot v \cdot Adv_{IND-CMA}(k, 3u \cdot v \cdot d + v \cdot d) + 2u \cdot v \cdot d \cdot Adv_{seal-cma, OTS, F}(k, 1) + 2u \cdot v \cdot Adv_{seal-cma, SIG, F}(k, 2u \cdot v \cdot d) \).

Supposed that \( A \) corrupted a direct authentication server \( S_c \) in ASQueue and deceived the client \( C_i \) in a session \( \Pi_{\Pi^*}^{T_i} \) with transcript \( T_{S_c, C_i}^I \). Note that each direct authentication server in ASQueue has to execute one complete mutual authentication procedure, and before \( S_c \) is popped out, its protocol transcript must be verified by at least \( z - 1 \) authentication servers among which at least 1 authentication server \( S_e \) is honest and uncorrupted by assumption. According to the identity fraud detection algorithm (i.e., Algorithm 2), \( S_e \) will check: i) Whether \( T_{S_c, C_i}^I \in \mathcal{PDB} \); ii) Whether the session of \( C_i^* \) recorded by \( T_{S_c, C_i}^I \) has a valid partner session at every partner of \( C_i^* \) in that session. If one of the checks fails, then
would report the identity fraud event and block \( C_i \) and \( S_c \). To make \( S_i \) accept, the adversary should successfully impersonate the uncorrupted party \( ID_j \in T_{S_c}^j \), in order that it can record the \( T_{S_c}^j \) without being detected. Hence, the identity fraud detection security property is implied by the mutual authentication security property.

The result of Theorem 2 implies the soundness of the identify fraud detection since an honest authentication server \( S_c \) can always identify that \( C_i \) is deceived within the protocol execution between them according to condition C2 in Definition 4.

6 PERFORMANCE

General Analysis. Here we assume that the 2MA protocol instances between a client \( C_a \) and an authentication server \( S_c \) are run over a network with a good quality of service, e.g., industrial network or intranet. In such a network, man-in-the-middle adversaries cannot continuously interfere with the communication between \( C_a \) and \( S_c \) without getting caught. Before running the 2MA protocol, two parties can diagnose the network via a few hello messages (e.g., via ping). Therefore, the OTS public key of \( S_c \) can be normally updated promptly at \( C_a \). As a result, \( C_a \) does not need to verify a very long signature-chain in \( SCache_{C_a} \). However, the 2MA protocol instances between two servers can run over any network (including the Internet) since long-term keys are used for authentication. And there is no need to cache the OTS update procedure.

The size of \( SCache \) determines the performance of the 2MA instances (involving OTS) since we may need to verify the signature-chain cached in it to verify the current signature. However, a network adversary which drops or modifies the network packets may lead to Denial-of-Service (DoS) attacks. Note that if there are such DoS adversaries, no authentication protocol can work at all. Fortunately, the network adversaries can be identified by a party, when it, for example, continuously received a few invalid message authentication codes (MAC) by our design. If the MAC checks fail, then the OTS keys cached in \( SCache \) would not grow. In this case, the party can initiate the DoS attacks detection to eliminate it before running the 4MA. To mitigate the DoS attacks, both client and authentication server can also have multiple OTS public key pairs as backup. Once a public key fails to update in \( \xi = |SCache| = 5 \) attempts, then it could trigger a DoS or identity fraud attack alarm, and switch to use the backup OTS keys for recovery. And it is very easy to commit new backup OTS keys when the DoS attacks are removed. In practice, it is, therefore, reasonable to assume that the upper-bound of authentication server’s cache \( |SCache_{S_c,C_a}| \) is \( \xi = 5 \), and the upper-bound of client’s cache \( |SCache_{C_a}| \) is \( \xi (z - 1) \) since it needs to keep the all OTS update procedures with \( (z - 1) \) unchecked servers. Furthermore, the minimum size of ASQueue is \( z = 3 \), in which case 4MA can resist 1 compromised server. To resist more compromised servers, it is needed to enlarge the number of \( z \), i.e., deploying more authentication servers.

Further Discussion. Since the DoS tolerant parameter \( \xi \) is small (e.g., \( 1 \leq \xi \leq 5 \) by assumption), to obtain an optimized performance for our protocol, it is sufficient to use a small \( z \) (i.e., the number of the server which are chosen by a client for both authentication and identity fraud detection). One could set the parameter \( z \) such that \( 3 \leq z \leq 6 \). For a cyber-physical system with \( n \) authentication servers such that \( n > 6 \), each client can randomly choose \( z \) servers out of \( n \) servers to run the protocol.

On the other hand, it is also possible to reduce the communication overhead of \( C_a \) in an online authentication session by sending the IFDCache to the next intended authentication server \( S_c \) in the queue ASQueue during the idle time of \( C_a \) (before doing the online authentication with \( S_a \)). Meanwhile, the sent IFDCache can be protected by a message authentication code generated based on the pre-shared key \( K_{S_c,C_a} \). This optimization would roughly reduce the communication cost by a factor of \( z \) for online authentication.

Implementation. We implement our system into the firmware layer of an OpenPLC [5] running on a Raspberry Pi 3 to simulate a client, and a laptop with Intel Core i7-8750H is used to simulate the server in our system. OpenPLC implements the firmware layer of a real PLC, which allows users to implement control logic on top of it. Most importantly, OpenPLC is a fully open-source implementation of a PLC, so we are able to integrate our authentication methods into its firmware. Here we first consider an instantiation instantiate the OTS scheme with the one called HORR [31]. To implement the OTS, we consider two concrete instantiations. We first instantiate the OTS scheme by the one called HORR [31] which has an optimized computational cost. For simplicity, we realized both the hash function and the one-way function (required by HORR) with SHA-256. In the implementation of HORR, the signing key consists of a set of random values \( \{ s_i \}_{i=1}^t \) where \( |s_i| = 80 \) and \( t = 256 \) (which equals to the output length of the SHA-256). Each signature of HORR is a subset of the signing key \( \{ s_i, s_j, \ldots, s_k \} \) where each index \( i_j \) (for \( 1 \leq j \leq k \)) is a \( \log_2^2 \) bits substring of the hash value of a message (see more details in [31]) and \( k = 32 \) in our setting. To obtain better memory and communication costs, we also choose to use Schnorr signature [33] to instantiate both the OTS and the long-term key-based signature (required by 4MA). We implemented the Schnorr signature based on the MNT224 elliptic curve. The PRF is realized by HMAC [22]. We implemented our protocol based on Miralid library [28]. The runtime of 4MA is shown in Figure 4 based on HORR and Schnorr, respectively.

To calculate the communication and the memory overheads, the lengths of identity, timestamp, and random nonce are considered as 32bits, 64bits, and 256bits, respectively. Since the memory and communication cost is close, we roughly depict them together for simplicity. And we fix \( \xi = 5 \) in this benchmark but change \( z \) from 3 to 8. We roughly show the memory and the communication costs in Figure 5.

![Figure 4: Runtimes in Milliseconds (ms)](image-url)
### Table 2: Comparison

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>He et al. [17]</td>
<td>PW+LSK</td>
<td>√</td>
<td>2/700</td>
<td>23H+6MUL</td>
</tr>
<tr>
<td>Challa et al. [9]</td>
<td>Bio+PW+LPK</td>
<td>√</td>
<td>2420</td>
<td>1Fe+14Mul+12h</td>
</tr>
<tr>
<td>Yang et al. [37]</td>
<td>PW+LPK</td>
<td>√</td>
<td>3/944</td>
<td>59h</td>
</tr>
<tr>
<td>4MA (HORS)</td>
<td>LSK+OTS</td>
<td>√</td>
<td>4/(13.4 + 10.72z + 5.42)</td>
<td>(ξ(k + 33z - 3) + t + 266)H + 3MUL</td>
</tr>
<tr>
<td>4MA (Schmoor)</td>
<td>LSK+OTS</td>
<td>√</td>
<td>4/(1.7 + 1.42ξ + 0.72)</td>
<td>(10 + 3ξ)H + (2ξ + 7)MUL</td>
</tr>
</tbody>
</table>

### Figure 5: Communication and Memory in KiloByte (KB).

**Comparison.** Here we briefly compare our proposed schemes with some existing lightweight delegated authentication protocols, including He et al. [17], Challa et al. [9] and Yang et al.[37]. For comparison, we consider the following perspectives: (i) authentication factors, (ii) main security properties, (iii) number of communication passes, and (iv) computation cost. Furthermore, we let ‘FE’ denote a fuzzy extractor operation to obtain a secret from biometrics. We let ‘S-Auth’ denote single-side explicit authentication, ‘M-Auth’ denote mutual authentication, ‘IA-SKS’ denotes implicit authentication with session key security, ‘IFD’ denotes the server’s identity fraud detection, and ‘BS’ denote the backward security. Our protocol can provide backward security since the OTS can evolve to a new one if the compromised key is not used by the adversary.

To calculate the communication overhead, the lengths of identity, timestamp, random nonce, hash value and ECC group value are considered as 32bits, 64bits, 256bits, 256bits and 224bits, respectively. To compare the computation cost, we let ‘H’ be hash function operation and ‘MUL’ be an ECC multiplication. We denote ‘Bio’ to be the biometric authentication factor, ‘PW’ to be password, ‘LSK’ denote long-term symmetric key, and ‘LPK’ denote the long-term public key. The computation and communication costs of 4MA are then calculated based on HORS OTS [31]. As there are many different types of entities in these protocols, we only show the computation cost of all entities for readability and simplicity.

The detailed comparison is shown in Table 2: it shows that our protocol 4MA are less efficient than the previous delegated authentication protocols. But the overall computation cost (as shown in Figure 4) of 4MA is still practical for constrained devices. Although the third message sent in our protocol includes a large cache IFDCache (which dominates the communication cost), it can be transmitted via a high-speed local network (e.g., 1000Mb Ethernet) in cyber-physical systems.

**Summary.** The computational cost of 4MA is with ξ (i.e., the DoS tolerant parameter). The client’s performance is still practical for our recommended parameters. The communication and the memory costs are linear with both parameters z (i.e., the number of authentication servers used to do IFD) and ξ. For 3 ≤ z ≤ 8 and 1 ≤ ξ ≤ 5, those costs are acceptable for modern PLCs, which usually have 1MB to 4MB user memory [6, 36]. To sum up, 4MA instantiating with HORS is much more efficient than 4MA instantiating with Schmoor. But the latter one has much fewer communication and memory overheads.

Although the third message sent in our protocol includes a large cache IFDCache (which dominates both the communication and the memory costs), it can be transmitted via a high-speed local network (e.g., 1000Mb Ethernet) in cyber-physical systems. As our protocol 4MA is generic, one can instantiate it with different concrete algorithms (regarding those building blocks) to obtain either better performance or more security properties (e.g., post-quantum security).

### 7 RELATED WORK

Nali and Oorschot [30] previously studied identity fraud detection by using a single sequentially evolved one-time symmetric authentication key, i.e., a new authentication key is generated solely based on the last key. To deal with the synchronization issue, Yang and Guo proposed a dynamic authentication credential (DAC) framework (hereafter referred to as the YG scheme). However, the YG scheme requires a PKI-based cryptographic building block to generate the ephemeral secret seed for updating DAC, so it is likely to be computationally expensive and is therefore unfit for power-constrained devices. Besides, the YG scheme is designed for two-party secure communication with any authentication delegator, so it cannot be applied to the three-party case. In [37], Yang et al. proposed a lightweight delegated authenticated key agreement (AKA) protocol for wireless sensor networks. It is based on a new DAC framework, which is extended from the YG scheme. One of the features of using DAC in this scheme is to detect previous impersonation attacks on users/sensors by comparing their current DAC with the corresponding ones stored at the gateway node. However, the symmetric DAC cannot resist the server breach.

Recently, Milner et al. [27] developed the foundations and proposed several constructions for security protocols that can automatically detect, if a secret (such as a key or password) has been misused. For key misuse detection, their constructions mainly on an evolving public/secret key scenario that puts an emphasis on a two-party authentication case. However, when considering a specific lightweight public key cryptographic scenario, e.g., one-time signature, new network threats would be sprung up (as we mentioned in Section 1). Moreover, unlike ours, Milner et al.’s work did not consider the key synchronization problem during the key evolving procedure. Nevertheless, it is non-trivial to extend their
construction idea to the lightweight delegated authentication that may involve symmetric key (for efficiency reasons) and more protocol participants.

Another important research direction on the mitigation of server breach is the threshold authentication. Agrawal et al. [2] introduced a generic password-based threshold token-based authentication framework (called it as PASTA) which allows $n$ servers to serve as identity provider. In PASTA, any $t \leq n$ servers can collaboratively verify passwords and generate tokens, while no $t - 1$ servers can forge a valid token and launch online dictionary attacks. PASTA leverages on threshold oblivious pseudo-random function (TOPRF) [15] as its main building block. However, like other threshold cryptosystems (e.g., [8, 12, 13, 32]), the concrete instances of TOPRF still require public key cryptographic operations, which are not suitable for resource-constrained devices in CPS.

We stress that all the works mentioned above do not consider the identity fraud issues caused by the compromised server in lightweight delegated authentication protocols. In this work, we focus on the formulation of a security model and new constructions for lightweight delegated authentication with identity fraud detection.

8 CONCLUSION

We have proposed the first lightweight key management framework and a delegated authentication protocol with identity fraud detection (IFD). We have shown that the proposed protocol can be applied to cyber-physical systems since the scheme is based on lightweight cryptographic primitives. We have also presented the formal security result of the proposed scheme under a new security model which formulates mutual authentication and server’s identity fraud detection simultaneously. We leave the construction of an efficient lightweight authentication protocol with more security features such as session key security as our future work. For example, one could use our authentication protocol as a compiler to build authenticated key exchange protocols by following [25].

ACKNOWLEDGMENTS

Jianjing Ning is supported by the Natural Science Foundation of China under Grant No.61872051 and 61972094. This work is also supported by the SUTTD start-up research under Grant No.: SRG-ISTD-2017-124.

REFERENCES


