IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 28 November 2022, accepted 12 December 2022, date of publication 21 December 2022,
date of current version 29 December 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3231015

RESEARCH ARTICLE

Accountable Bootstrapping Based on Attack Resilient Public Key Infrastructure and Secure Zero Touch Provisioning

DANU DWI SANJOYO[1,2] AND MASAHIRO MAMBO[3], (Member, IEEE)
[1]Graduate School of Natural Science and Technology, Kanazawa University, Kanazawa 920-1192, Japan
[2]School of Electrical Engineering, Telkom University, Bandung 40257, Indonesia
[3]Institute of Science and Engineering, Kanazawa University, Kanazawa 920-1192, Japan

Corresponding author: Danu Dwi Sanjoyo (danudwj@telkomuniversity.ac.id)

The work of Danu Dwi Sanjoyo was supported in part by the Japanese Government (Ministry of Education, Culture, Sports, Science and Technology/MEXT) Scholarship, and in part by Telkom University.

ABSTRACT Internet Engineering Task Force (IETF) issued Secure Zero Touch Provisioning (SZTP) as a provisioning technique for networking devices without human intervention. SZTP standardizes the provisioning workflow from device enrollment to bootstrapping process. Unfortunately, implementing a single trust model of public key infrastructure scheme in zero-touch device provisioning is vulnerable to impersonation attacks using bogus certificates. This paper proposes a robust protocol for the bootstrapping process of edge devices by integrating the Attack Resilient Public Key Infrastructure (ARPKI) scheme with SZTP. As a transparent and accountable public key infrastructure, ARPKI can prevent the miss-issuance of a certificate. ARPKI offers strong security as certificate management for SZTP. We adopt the security properties of ARPKI to construct an accountable bootstrapping scheme of a zero-touch provisioned edge device against threats, e.g., impersonation, incurred by insiders compromised by adversaries. The edge device and bootstrap server can confidently build mutual authentication using the TLS 1.3 full handshake protocol together with the ARPKI-based certificates built upon a trusted public certificate log, which provides the accountability of the certificate. We analyze our scheme's security properties by performing formal and informal analyses. We show that the combination of ARPKI and SZTP can detect malicious entities and mitigate misbehaving activities. Our provisioning scheme provides accountable bootstrapping for edge devices in a zero-touch fashion with integrity and confidentiality of bootstrapping data.

INDEX TERMS Autonomous bootstrapping, secure onboarding, attack resilient, certificate management, accountable protocol, device provisioning.

I. INTRODUCTION

The Public Key Infrastructure (PKI) scheme in zero-touch provisioning of edge-device plays an essential role in securing the online bootstrapping process. Securely operating the bootstrap is not simple, mainly because the device bootstrapping occurs automatically without human intervention [1]. Unlike manual bootstrapping, which occurs physically, a factory-default state device must connect to a bootstrap server through the internet to perform automated bootstrapping. The automated bootstrapping process is a vulnerable step wherein an edge device lacks network-based protection in this step, mainly if the device is in an insecure public network. Therefore, the first step in bootstrapping is that the device can authenticate a proper bootstrap server. On the other hand, as a part of the Service Provider (Provider), the bootstrap server must also ensure that it provides bootstrap data to an appropriate device. Thus, it is necessary to establish mutual authentication between the device and the bootstrap server. In addition, the integrity and confidentiality of information also need to be ensured. Especially on edge devices that bootstrap with a zero-touch

The associate editor coordinating the review of this manuscript and approving it for publication was Jad Nasreddine.

method, the mutual authentication development process is not trivial because the entire mutual authentication process starts automatically with activating the device. In this situation, all entities belonging to the bootstrapping system should perform the protocol without any deviation.

An ability to identify misbehaving parties, who deviate from the protocol and cause a security violation, is defined as accountability [2]. In an accountable system, every single entity must behave according to the protocol and prove the ownership of the correct and valid certificate to other parties. Therefore, the system requires transparent and accountable certificate management to ensure no miss-issuance of certificates during operation. On the other hand, a system cannot rely on a single trust Root Certificate Authority (RCA) to achieve accountability in a decentralized device network. Even though it has several advantages in ease of maintenance, once the trusted entity is compromised, the system becomes vulnerable to impersonation attacks. An adversary can intentionally control the compromised Certificate Authority (CA) to issue fraudulent certificates [3]. Moreover, the Internet of Things (IoT) network is a potential spot for cyber-criminal [4], [5]. We understand that every certificate management step, from registration to authentication, must be carried out transparently to uncover misbehaving activities immediately. Therefore, distributed device network requires a transparent PKI system instead of a single trust CA.

ARPKI is a transparent PKI system that is solid in maintaining accountability of certificate management [6]. This system regulates certification procedures from registration to revocation. ARPKI offers strong security guarantees against impersonation, although it has a delay due to the involvement of all designated entities in all processes [7].

Meanwhile, IETF's SZTP scheme offers a remote bootstrapping strategy without any technical operator intervention in a secure fashion [8]. Several frameworks and companies have practically implemented device provisioning with a zero-touch approach. However, no work proposes PKI-based accountability preservation in the SZTP framework. On the other hand, the robustness of the SZTP scheme by IETF relies on the correct certificates. From this perspective, providing powerful certificate management is a critical point. A transparent and accountable PKI architecture can protect the provisioning scheme from malicious parties and misbehavior activities.

Certifying zero-touch provisioned devices' public keys and identities is a typical mechanism to secure the devices and the system. By owning a valid certificate, a device can confidently communicate with servers or other devices through an insecure network. Unlike manual bootstrapping, in which an administrator usually installs certificates into the device after bootstrapping, in automated bootstrapping, the device certificate installation can occur before, during, or after bootstrapping. In this paper, we follow the Request for Comments (RFC) 8572 document of SZTP to install the device certificate before bootstrapping occurs remotely through the internet [8]. By possessing a certificate, the device can obtain the bootstrapping data securely from a bootstrap server. Certificate-based mutual authentication between the device and bootstrap server can immediately start by verifying each certificate without wasting time retrieving the certificate from CA. Moreover, installing a certificate into a device during its manufacturing, i.e., at the beginning of its lifecycle, can reduce the consumer's burden and simplify its management. Several existing protocol designs also put the certificate installation before bootstrapping. Höglund et al. [9] develop a light automated certificate enrollment protocol for resource-constrained devices, which occurs before bootstrapping process. Maksuti et al. [10] put the certificate installation before automated and secure bootstrapping in a System of Systems framework. Other approaches [11], [12], [13], [14], [15] also propose certificate installation for IoT devices before bootstrapping, although not all of them are related to automated bootstrapping. On the other hand, Boire et al. [16] design automated certificate provisioning during the bootstrapping process by simplifying the multiple layers operations into one process after connecting to the Wi-Fi network. Another approach, Sousa et al. [17] propose a semi-autonomous certificate provisioning for IoT devices after bootstrapping in the setup process.

Some research proposed various secure schemes for device bootstrapping. Still, to the author's best knowledge, there is no proposed scheme utilizing certificate transparency of PKI to provide accountability for the bootstrapping process. Moreover, very few researchers determine the accountability properties of their system, though accountability is essential to ensure bootstrapping performs safely. We present the integration approach of ARPKI and SZTP for the bootstrapping process to provide the integrity and confidentiality of bootstrapping data so as to achieve accountable bootstrapping.

This paper contributes four essential objectives:

- First, we integrate ARPKI certificate management into the emerging SZTP framework to establish mutual authentication between the edge device and bootstrapping server using TLS 1.3 full handshake (presented in the PROPOSED INTEGRATION OF ARPKI AND SZTP section).
- Second, a straightforward combination of ARPKI and SZTP is not enough to achieve accountable bootstrapping. We apply Keyed-Hashing for Message Authentication Code (HMAC) as message authentication in the Credential Exchanges process between Provider and Manufacturer to verify that communicating entities are behaving honestly (presented in the CREDENTIAL EXCHANGES subsection).
- Third, we provide a formal analysis using extended BAN logic to demonstrate the integrity and confidentiality of bootstrapping data, of which no similar study has been conducted in the SZTP framework (presented in the FORMAL ANALYSIS subsection).
- Forth, we provide an informal analysis of the bootstrapping process to confirm accountability, a security

property that has never been previously discussed in automated device onboarding frameworks, including SZTP (presented in the SECURITY ASSESSMENT subsection).

The rest of this paper is organized as follows: Section II explains related work addressing the secure edge device bootstrapping scheme. Section III presents the key to background perceptions and assumptions. Section IV gives our proposed ARPKI-SZTP scheme. Section V proves the security properties in formal and informal analyses. Section VI outlines our findings and future works.

II. RELATED WORK

Several related works address automated device bootstrapping schemes in a secure style. In [10], the authors present an automated and secure onboarding procedure for System of System. This procedure solves the costly and time-consuming manual process. It offers security preservation of onboarding by establishing a chain of trust between a new hardware device and its provided services. This work also introduces three onboarding mechanisms based on credentials for authentication: preloaded Arrowhead certificate, manufacturer certificate, and shared secret. However, the author does not discuss accountability properties. The work is toward deploying public-key cryptography and delivering software updates through the internet for those constrained-resource devices.

Enabling public key management for the IoT ecosystem is a big challenge. Authors in [9] develop Public Key Infrastructure for the Internet of Things (PKI4IoT), a lightweight and automated certificate enrollment protocol for IoT devices to provide end-to-end security to answer this challenge. The authors also design a compressed-overhead profile of the X.509 certificate. However, the proposed protocol lacks protection against denial-of-service and physical access attacks. Also, key provisioning and key revocation are not available in this proposal.

Belattaf et al. [13] offer a reliable and adaptive decentralized public-key management infrastructure for IoT to provide end-to-end security service. The proposed infrastructure solves the costly and nut scalable solutions of symmetric-key management. Key Distribution Center (KDC) distributes the group keys to handle devices' periodic attachment and release. This work results in better performance than PKI4IoT in cost and delay. However, this work inherits the security issue of PKI4IoT and does not employ automated device provisioning.

Intel established Intel Secure Device Onboarding (SDO) as an industrial-based solution in 2017 and announced it as open-source in 2020 [14]. Intel SDO employs a late-binding approach to configure devices at the installation point instead of pre-customizing each customer system. This method provides a zero-touch onboarding method with fast and enhanced security service. To provide integrity, they utilize a unique root of trust key based on Enhanced Privacy ID (EPID) or (Elliptic Curve Digital Signature Algorithm) ECDSA for each device. However, Intel SDO plays as a trusted manufacturer, service provider, and certificate authority at the same time. This single trust model can become vulnerable when the trusted entities are compromised.

In [15], the authors propose a Zero Touch Provisioning (ZTP) solution involving four big companies to provide a well-secured automated identity chain. Considering Zero Trust Network (ZTN) in their device deployment method, embedding hardware root-of-trust, and pairing the network with march-plural root CAs-based PKI service allows the system only to be available for the trusted device. They also consider implementing a Trusted Platform Module (TPM) [18]. However, Moghimi et al. [19] discover the vulnerabilities of TPM devices by employing side-channel key recovery attacks. Moreover, using multiple root CA certificates is a less efficient solution.

In this paper, we consider engaging log-based PKI to provide certificates. Several related works proposed robust certificate management protocols featuring transparency and public verifiability. Certificate Transparency [3] is the first proposal triggering the idea of the public, verifiable, and append-only log. Kim et al. [20] introduce the utilization of several CAs and Integrity Log Server (ILS) as multiple signing parties as roots of trust. However, this proposal lacks certificate miss-issuance prevention if two out of three signing entities are malicious. Szalachowski et al. [21] propose Policert to secure domain certificates by allowing domain owners to create their policies for the certificates and control the operation over the certificate lifecycle. Inspired by PoliCert, Khan et al. [22] propose Transport Layer Security (TLS) PKI, namely Accountable and Transparent TLS Certificate Management (ATCM), to prevent a Man-In-the-Middle (MITM) attack, though only one party out of multiple entities is trusted. These last two proposals are unsuitable for implementation in the SZTP framework since the certificate domain is an unattended device. Attack-Resilient TLS Certificate Transparency (ARCT) [7] offers robust prevention against adversaries by managing intermediate CA transparently. This proposal resolved the malformed certificate issuance. However, when some entities, including Exclusive Log Server (ELS), are compromised, the adversaries can create a bogus certificate by leveraging the privilege since the Root CA (RCA) sends certificate approval via ELS. Another approach that provides transparent certificate management is Blockchain-based PKI. This approach also offers accountability without certificate logs, nor the necessity of external auditing [23], [24], [25]. However, the Blockchain-based PKI scheme has typical major problems in scalability issues [26] and cost [27].

III. BACKGROUND

The increasing number of edge devices requires a speedy and costless provisioning method while maintaining security and privacy. In this section, we introduce various essential properties needed to establish accountability of the provisioning method.

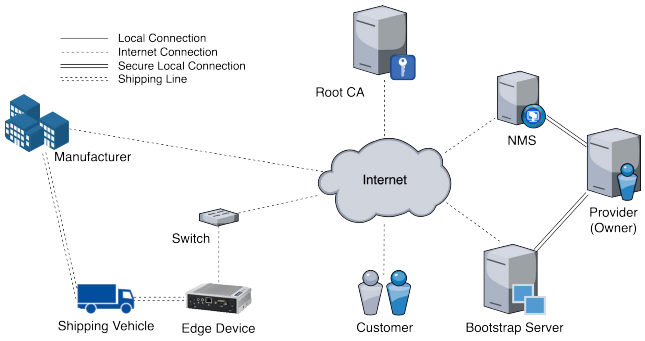


FIGURE 1. RFC 8572-based SZTP architecture with a single Root CA.

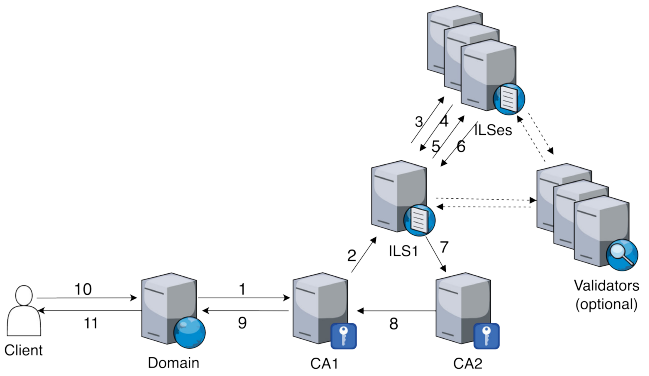


FIGURE 2. Certificate registration process of ARPKI.

A. SECURE ZERO TOUCH PROVISIONING

SZTP is a bootstrap method that empowers devices to securely retrieve bootstrapping data without any technical action beyond physical arrangement and connecting to the network and power station. This method includes updating the boot image, committing an initial configuration, and executing arbitrary scripts to address auxiliary needs [8]. The RFC 8572 document standardizes the procurement of physical equipment, where the device's initial state setting occurs during the device's manufacturing process.

RFC 8572 document covers two use cases of device provisioning scenarios. The first use case assumes the device gets support from a locally administered network and may use a local service to perform bootstrap. Our proposed scheme adopts the second use case, whereby the edge device connects to a network managed by Internet Service Provider (ISP) as an access gateway.

IETF introduces two correlating roles to edge device provisioning: Manufacturer and Owner. Figure 1 depicts the basic architecture of SZTP. After receiving device enrollment from a Prospective Owner, the Manufacturer produces an edge device, equips it with the necessary credentials, and ships it to a Customer's designated location. The Prospective Owner initiates the enrollment process, including one of two following bootstrapping options. The first option is the Prospective Owner tends to bootstrap their device from the Owner's deployment-specific bootstrap server. Therefore, the Manufacturer should provide the trust anchors of the *IDevID* (Initial Device Identifier) certificate to the Prospective Owner. These trust anchor certificates must be available on the Prospective Owner's Network Management System (NMS) to authenticate the device. The second option is the Manufacturer hosts a well-known bootstrap server (manufacturer-hosted bootstrap server) for the devices. In our proposed scheme, we assume the Owner prepares the deployment-specific Bootstrap Server (*BS*) since the Provider, as the Owner, will maintain the machine during operation. Consequently, the Manufacturer should provide the Provider with the trust anchor certificates to authenticate the *IDevID* certificate.

Once the edge device arrives at the designated place, it immediately builds a connection to the listed *BS* via the internet to request the bootstrapping data. The *BS* authenticates the device by verifying the *IDevID* certificate issued by the *RootCA*. Based on RFC 8572, IETF allows any CA to sign the Owner Certificate and entities' certificates of the architecture. The role of the CA is significant, and CA should be a trusted entity that ensures the integrity of entities in the architecture. However, this document does not specify the required number of CA and CA's architecture for SZTP.

In this paper, we adopt a trusted *BS* instead of an untrusted one and a secure channel instead of an insecure one to achieve higher security construction. Consequently, we only need to install four out of seven components defined in the RFC 8572 document: a TLS-level client certificate, trust anchor certificates, a list of trusted *BS*s, and the device's private key, into the device. Note that even though we adopt the higher security construction of legacy SZTP, it cannot satisfy the accountability to deal with the utilization of a single trust Root CA as mentioned in the Introduction.

B. ATTACK RESILIENT PUBLIC KEY INFRASTRUCTURE

ARPKI offers strong security using at least three entities out of n entities consisting of *CA*s and *ILS*es for certificate management: two *CA*s and one Integrity Log Server (*ILS*) [6]. In contrast to Accountable Key Infrastructure (AKI), which cannot avoid a certificate miss-issuance, ARPKI applies an attack resilience mechanism to prevent the adversary gets a bogus certificate even though two out of three signing entities are compromised. AKI requires validators to monitor *ILS* operations and detect misbehavior [20]. In ARPKI, validators are optional entities that can, time after time, validate the certificate log stored in *ILS* to confirm accountability. ARPKI requires another *CA* to take the validator's role. Thus, in ARPKI, *CA*1 is a validator of *CA*2 to check whether correctly monitors *ILS*1.

Figure 2 presents the certificate registration process of ARPKI. Steps 1 and 2 are the certificate registration request flows. Besides the certificate statement, the registration request also contains a list of trusted *CA*s to sign the certificate. The domain interacts only with *CA*1, which is responsible for checking the correctness of the other two entities' operations. The *ILS*1 performs *ILS* synchronization

with a quorum of all existing *ILS*es to ensure that only the requested certificate exists for the domain (Steps 3 - 6). The *ILS*1 synchronizes the new certificate request to a group of *ILS*es. ARPKI protocol requires more than 50% of all *ILS*es' synchronization messages before the *ILS*1 issues a new certificate. After verifying the new certificate, *ILS*1 manages the certificate log requested by *CA*1 and releases the certificate (Step 7). All three entities in ARPKI (*CA*1, *CA*2, and *ILS*1) sign the domain's certificate while ensuring that they belong to the **`CA_LIST`** (list of trusted *CA*s to sign the certificate) or **`ILS_LIST`** (list of trusted *ILS*es to register the certificate) and are different from each other (Step 8). After getting the certificate (Step 9), the *Domain* can establish a TLS connection with the *Clients* (Steps 10 and 11).

C. TLS 1.3

A secure channel based on TLS 1.3 provides authentication, confidentiality, and integrity properties. The TLS 1.3 version improves the previous version in some minor differences. The most noticeable difference is the addition of zero round-trip time (0-RTT) mode that saves a round trip at connection setup. All handshake messages after the Server Hello are encrypted. This encrypted mode leads to a safer channel setup, but it has consequences in the computational cost. Dowling et al. [28] analyze the handshake protocol of TLS 1.3. The authors prove that TLS 1.3 handshake protocol establishes session keys under standard cryptographic assumptions.

Several studies discuss the security performance of TLS 1.3 against the MITM attack. In [29], Arfaoui et al. investigate privacy preservation in TLS 1.3. The full TLS 1.3 handshake has a drawback in the privacy game against MITM attacks since the handshake-secret computation occurs before the authentication step. However, this drawback has no risk related to the security of the keys due to the Elliptic Curve Diffie Hellman Exchange (ECDHE) utilization in TLS 1.3 [30]. Moreover, Lee et al. [31] demonstrate the protection in TLS 1.3 against MITM-based downgrade attacks. Hence, we assume that establishing the TLS 1.3 handshaking guarantees security against MITM attacks, and entities in our scheme reject requests to downgrade to an older version of TLS.

This paper employs TLS 1.3-based authentication for all communications except within the trust boundaries. All entities must perform TLS 1.3 Full Handshaking before starting the first session of communication and Resumption Handshaking for later sessions. Appendix A shows the TLS 1.3 full handshaking process between any two parties (e.g., $\mathcal{A}$ and $\mathcal{B}$). $\mathcal{A}$, as a client, requests to establish TLS 1.3 full handshaking before sending a message to $\mathcal{B}$, as a server. From the full handshake process, both entities possess four session key data, e.g., server application key, server application Initialization Vector (IV), client application key, and client application IV, to encrypt and decrypt messages to each other. To simplify the discussion, we replace these four session key data with one session key notation, i.e., we express session key data between Manufacturer and *CA*1 with K_{MCA1}.

D. ADVERSARY MODEL

In this work, we assume the adversary can compromise entities by obtaining long-term private keys. Note that we refer to an adversary as an external party of our proposed scheme. The adversary can take over the network by eavesdropping, modifying, and inserting fraudulent messages. An adversary may compromise internal entities and make them deviate from the protocol. The adversary aims to impersonate the zero-touch provisioned device and obtain the bootstrapping data from a legitimate bootstrap server. We assume that no internal entity is a fully trusted party. Some internal entities are dishonest, and such dishonest entities misbehave as compromised entities do. In our discussion, we categorize attacks by compromised or dishonest entities as insider threats.

The adversary is presumed unable to compromise all entities simultaneously. As a result, we use the attack model that the adversary can compromise the long-term private key of some, but not all, entities. We use the adversary model in our analyses, except in the formal analysis, because we follow the assumption of the BAN logic where all entities operate honestly.

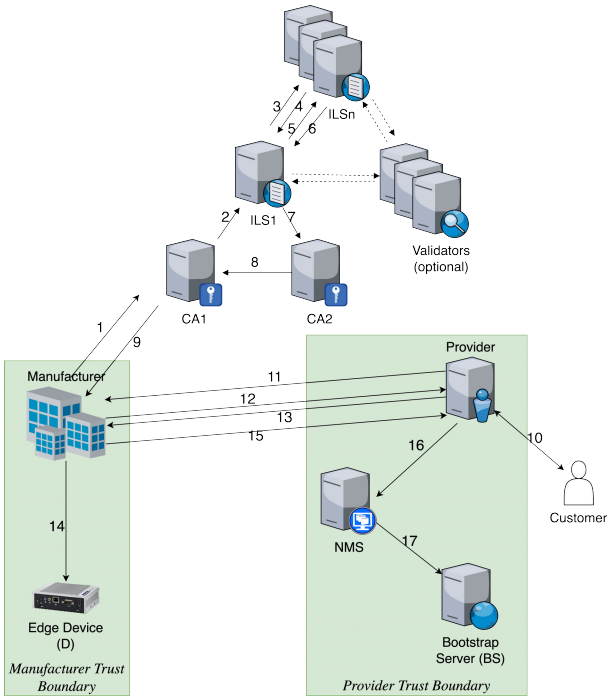
IV. PROPOSED INTEGRATION OF ARPKI AND SZTP

We combine the transparent and accountable ARPKI as certificate management and the secure automated bootstrapping scheme SZTP. ARPKI, with n entities consisting of *CA*s and *ILS*es, guarantees the accountability of certificate issuance to prevent bogus certificates for a Device (D). To simplify discussions, we examine our proposed integration using ARPKI with two *CA*s and one *ILS* ($n = 3$). Figure 3 illustrates the combined architecture of ARPKI and SZTP. This architecture covers *IDevID* Certificate Registration (Figure 3a Steps 1-9), Certificate Update and Revocation (Figure 3a Steps 1-4, 7-9), Certificate Log Validation (Figure 3b Steps 1-5), Credential Exchanges (Figure 3a Steps 11-17), and Bootstrapping (Figure 3c Steps 1-3) processes.

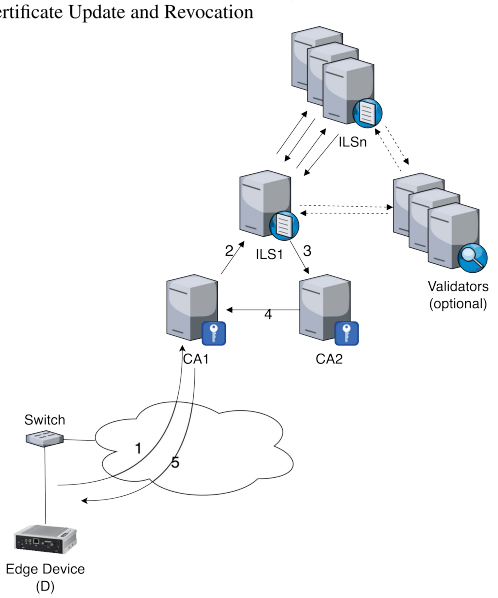
In our architecture, a Customer acts as a Provider's subscriber, who reserves a zero-touch provisioned edge device through the Provider's web page to deploy it in a designated place. The Provider requires the Customer's identity, device specification requested by the Customer, and shipping address designated by the Customer. The Provider sends the shipping address in the *DevOrder* message to the Manufacturer once D is ready. Then, the Manufacturer ships the ordered D to the designated place according to the shipping address.

A. IDevID CERTIFICATE REGISTRATION

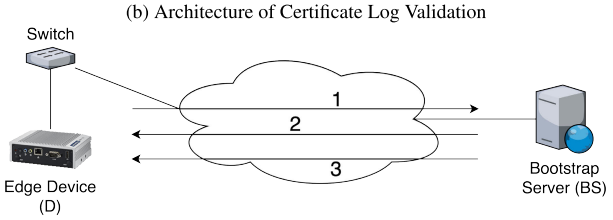
Figure 4 shows the *IDevID* Certificate Registration message flow. Principally the Manufacturer has the policy to register an *IDevID* certificate at any time. For device provisioning



(a) Architecture of $IDevID$ Certificate Registration, Credential Exchanges, and Certificate Update and Revocation



(b) Architecture of Certificate Log Validation



(c) Architecture of Bootstrapping

FIGURE 3. Architecture of ARPKI-SZTP.

TABLE 1. Symbols used in this paper.

Symbol	Description
ID_D	Device's identifier
S_D	Device's specification
Sh_D	Device's shipping address
ShT_D	Device's shipping tracking notification
K_D	Device's public key
K_D^{-1}	Device's private key
SN_D	Device's serial number
$RegReq$	New certificate registration request
$RegResp$	New certificate registration response
C_D	$IDevID$ certificate statement
ACC_D	$IDevID$ certificate acceptance confirmation
$Cert_D$	$IDevID$ certificate
$Cert_{CA1}$	$CA1$'s certificate
$Cert_{CA2}$	$CA2$'s certificate
$Cert_{ILS1}$	$ILS1$'s certificate
K_{CA1}^{-1}	$CA1$'s private key
K_{CA2}^{-1}	$CA2$'s private key
K_{ILS1}^{-1}	$ILS1$'s private key
K_{Man}^{-1}	Manufacturer's private key
K_{Prov}^{-1}	Provider's private key
$List_{BS}$	List of bootstrap server's address
$List_{TABS}$	List of BS's trust anchor certificates
$List_{TAD}$	List of $IDevID$'s trust anchor certificates
CI_D	Conveyed information for device
K_{MCA1}	TLS 1.3-based session key for Manufacturer and $CA1$
K_{L1CA1}	TLS 1.3-based session key for $ILS1$ and $CA1$
K_{L1CA2}	TLS 1.3-based session key for $ILS1$ and $CA2$
K_{CA1CA2}	TLS 1.3-based session key for $CA1$ and $CA2$
K_{L1Ln}	TLS 1.3-based session key for $ILS1$ and $ILSn$
K_{DCA1}	TLS 1.3-based session key for Device D and $CA1$
K_{CP}	TLS 1.3-based session key for Customer and Provider
K_{MP}	TLS 1.3-based session key for Manufacturer and Provider
K_{DBS}	TLS 1.3-based session key for Device D and Bootstrap Server BS
Hdr	TLS 1.3 Record Header
$HSHdr$	TLS 1.3 Handshake Header
Ext	TLS 1.3 Extensions
Tag	TLS 1.3 Authenticated Encryption with Associated Data (AEAD) tag
NST	TLS 1.3 new session ticket
Len	Certificate Length
$zeros$	An arbitrary-length run of zero-valued bytes
IV_{HSD}	Device's handshake initialization vector
IV_{HSBS}	Bootstrap Server's handshake initialization vector
K_{HSD}	Device's handshake key
K_{HSBS}	Bootstrap Server's handshake key
IV_{APD}	Device's application initialization vector
IV_{APBS}	Bootstrap Server's application initialization vector
K_{APD}	Device's application key
K_{APBS}	Bootstrap Server's application key
H	cryptographic hash function
$opad$	block-sized outer padding, consisting of repeated bytes valued 0x5c
$ipad$	block-sized inner padding, consisting of repeated bytes valued 0x36

using the late-binding method, the Manufacturer and Provider can organize an agreement on the combination of device identifier ID_D and specification S_D in advance.

After obtaining the S_D and ID_D, the Manufacturer can register the *IDevID* certificate. The Manufacturer sends *RegReq*, a registration request message of the new *IDevID*

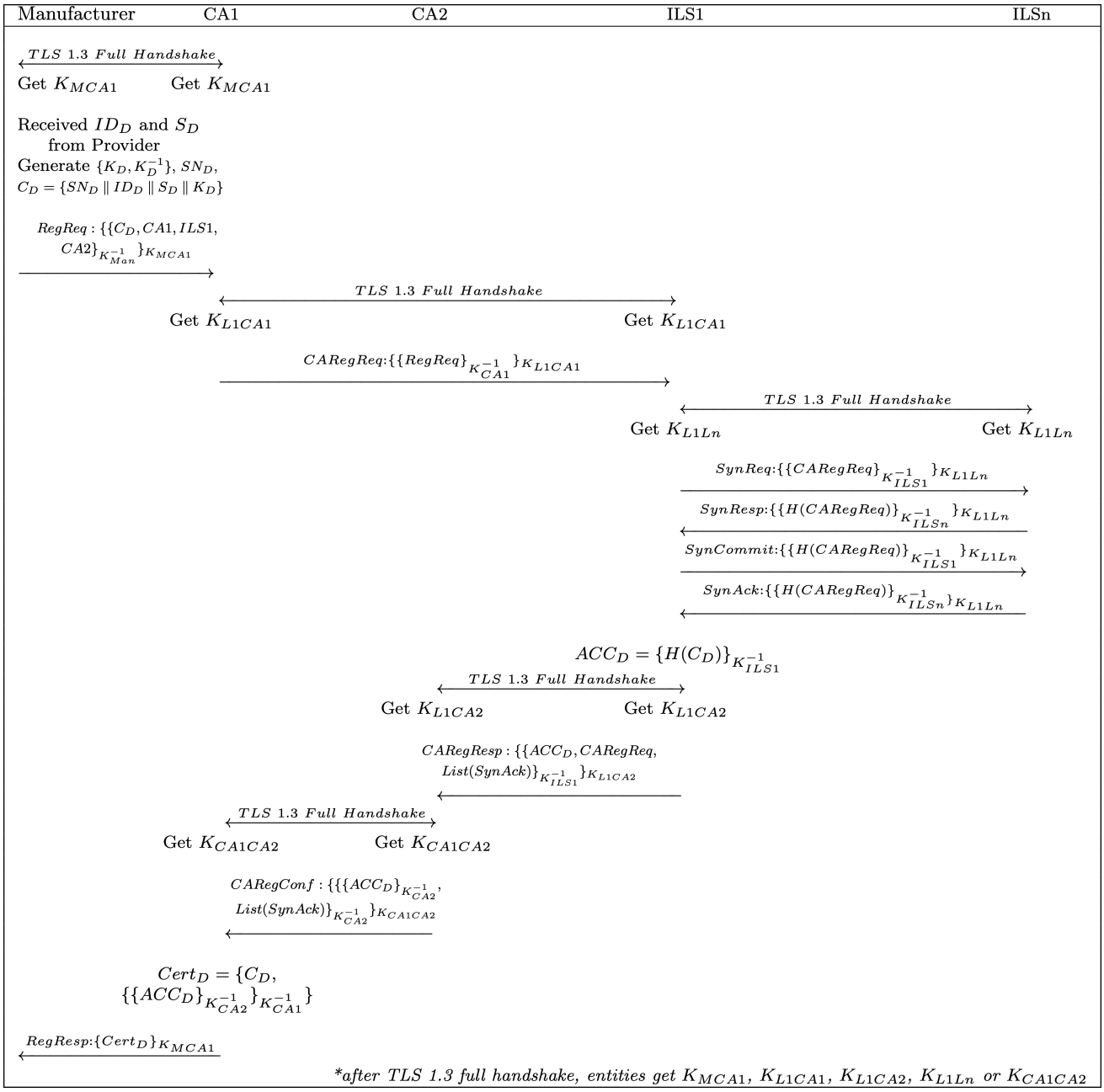


FIGURE 4. ***IDevID*** **certificate registration message flow.**

certificate containing the certificate statement C_D, to $CA1$ (Step 1). $CA1$ receives the registration request from the Manufacturer, then $CA1$ verifies the Manufacturer's signature and forwards the request to $ILS1$ (Step 2). $ILS1$ checks the log of certificates to see whether the requested new certificate has already existed. If no certificate is identical with the new intended certificate, $ILS1$ performs synchronization among *ILS*es. $ILS1$ sends the signed synchronization request message to *ILSn* to ensure that there is only one unique registered certificate for a single domain. The active *ILS*es, which belong to *ILSn*, respond to $ILS1$'s request by verifying the signature and sending signed synchronization response messages containing the hash value of the *CARegReq* message. $ILS1$ sends *SynCommit* messages in response to the *ILSn*'s *SynResp*s. After confirming the correctness of the new certificate registration, the *ILSn* send *SynAck* messages to $ILS1$. These synchronization steps prevent any possible impersonation attacks. After collecting *SynAck*s from a quorum of *ILSn*, $ILS1$ issues the signed certificate of the newly registered certificate (Steps 3-6). $ILS1$ sends the registration response to $CA2$ (Step 7). After verifying the signature of $ILS1$ and ensuring that $ILS1$, $CA1$, and $CA2$ are different entities, $CA2$ sends a signed registration confirmation message to $CA1$ (Step 8). Lastly, $CA1$ signs the

final certificate and sends the *RegResp* message containing the certificate to the Manufacturer (Step 9).

B. CERTIFICATE UPDATE AND REVOCATION

For certain reasons approved by the Provider and the Manufacturer, a valid certificate may become invalid and changed to a new certificate with an updated certificate statement. ARPKI provides the certificate update and revocation procedure. Figure 5 describes a mechanism for updating and canceling a valid certificate. The Manufacturer initiates sending the *UpdateReq* message containing a new certificate statement to $CA1$ (Step 1). Then, $CA1$ forwards the request to $ILS1$ after verifying the Manufacturer's signature (Step 2). $ILS1$ performs the ILS synchronization to validate the new update request (Steps 3-4). After collecting *SynResp*s from $ILSn$, $ILS1$ sends the signed ACC_D, *CAUpdateReq*, and *List*(*SynResp*) to $CA2$ (Step 7). $CA2$ signs the *CAUpdateConf* and sends it to $CA1$ (Step 8). The $CA1$ finalizes the update by signing the message and sending it to the Manufacturer (Step 9).

C. CERTIFICATE LOG VALIDATION

A device D should gather ILS's proof Prf before the timeout of an $ILS1$'s registration response (*RegResp*) or after the $ILS1$ updates the certificate tree. Figure 6 illustrates the Certificate Log Validation message flow. Firstly, D sends the confirmation request to $CA1$ (Step 1). Then, $CA1$ forwards the request to $ILS1$ (Step 2). The $ILS1$ responds to the request by sending the proof Prf and signed root SRt to $CA2$ (Step 3). The $CA2$ signs the message and delivers it to $CA1$ (Step 4). Then, $CA1$ finalizes it by signing it and sending it to D (Step 5).

D. CREDENTIAL EXCHANGES

Figure 7 shows the Credential Exchanges message flow. This flow (Steps 10-17) consists of Enrollment and Ordering Devices (Steps 10-15) and the Owner Stages the Network for Bootstrap (Steps 16-17) workflows of RFC 8572. The Customer Registration and Authentication via the Provider's website initiate the Provider to enroll an edge device to the Manufacturer (Step 10). The prospective Customer is requested to input Customer information, including identity, specification of the desired device, and shipping address. We assume the Customer inputs the data in secure action due to HTTPS-based web service through TLS 1.3 full handshake authentication. Then the Provider generates the device identifier ID_D, specification S_D, and shipping address Sh_D correlating to the Customer's information.

In our proposed scheme, we define additional device identifiers, i.e., device identifier ID_D and device specification S_D, representing the Customer's identity and service requested by the Customer, respectively. We designate these identifiers for convenience in service handling. Two other device identifiers defined in RFC 8572 are *IDevID* and manufacturer-generated device identifiers. *IDevID* indicates the device certificate used for the TLS-level client certificate and the device identity certificate used by the Owner to encrypt bootstrapping data. The manufacturer-generated device identifier indicates the device serial number (in this paper, this identifier is denoted by SN_D).

To perform device enrollment, the Provider sends a *DevEnroll* message to the Manufacturer. On the other hand, the Manufacturer sends the SN_D and *IDevID* certificate's trust anchor to the Provider to be inserted into NMS and BS so that they can recognize the device during the bootstrapping process. Each entity must receive correct and valid identifiers in this Credential Exchange process. Otherwise, an adversary can forge the identifiers to create fake devices.

To reduce the security weaknesses in terms of integrity property, we also define additional variables $H_{DevEnroll}$ and $H_{DevOrder}$ generated by the Provider using Message Authentication Code (MAC) based on HMAC in RFC 2104 [32]. According to RFC 2104, we can define *HMAC* over data *text* with secret key *Key* as $HMAC(Key, text) = H((Key' \oplus opad) \| H((Key' \oplus ipad) \| text))$, where Key' is $H(Key)$ if *Key* length is longer than block size B, otherwise $Key' = \text{ZeroPad}_B(Key) = \{Key \| (B - Key \text{ length}) \text{bytes of } zeros\}$. We employ past session short-term key $K_{MP(t-1)}$ of TLS 1.3 full handshake as the *Key* in our HMAC formula. Since the longest key length of cipher suites available in TLS 1.3 is 32 bytes and the smallest hash function block size available for HMAC is 64 bytes, we use the *Key* appended with zero padding. In this paper, we define $H_{DevEnroll(t)}$ and $H_{DevOrder(t)}$ of current time t as follows.

$$H_{DevEnroll(t)} = HMAC(K_{MP(t-1)}, \quad \{ID_D, S_D\}_{K_{Prov}^{-1}}) \quad (1)$$

$$H_{DevOrder(t)} = HMAC(K_{MP(t-1)}, \{ID_D, List_{BS}, List_{TABS}, Sh_D\}_{K_{Prov}^{-1}}) \quad (2)$$

The Provider sends a device enrollment message *DevEnroll* containing the ID_D, S_D, and $H_{DevEnroll}$ to the Manufacturer (Step 11). The Manufacturer inserts ID_D and S_D in the *IDevID* certificate, which the Manufacturer sends its trust anchor certificates list ($List_{TAD}$) to the Provider as an enrollment response *EnrollResp* message (Step 12). The Provider sends the device order *DevOrder* message containing a list of BS's address $List_{BS}$, a list of the BS's trust anchors $List_{TABS}$, Sh_D, and the ID_D identifier to the Manufacturer (Step 13). The Manufacturer follows the order up by inserting *IDevID* certificate, $List_{BS}$, $List_{TABS}$, and the device's private key K_D^{-1} to finalize D (Step 14). Then, the Manufacturer ships D to the designated address and informs the Provider about the device serial number SN_D and shipping tracking notification ShT_D (Step 15). This step ends the Credential Exchanges between the Provider and the Manufacturer. Then, the Provider prepares the bootstrapping by generating Conveyed Information for D (CI_D), inserting some device's credentials (ID_D, SN_D, and CI_D) to NMS, and installing the bootstrapping-related information (SN_D, CI_D, $List_TAD$) to BS through NMS (Steps 16-17).

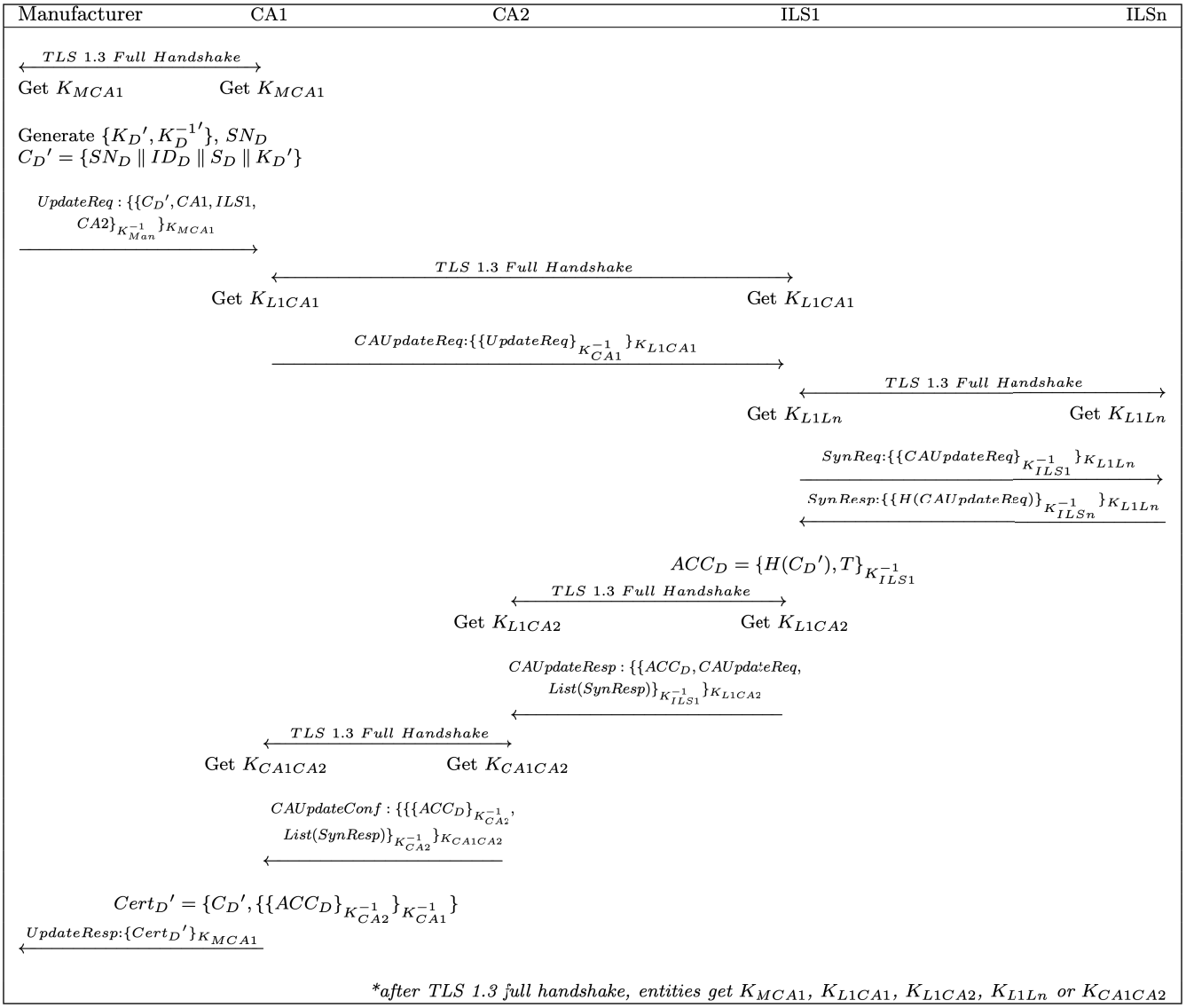


FIGURE 5. Certificate update and revocation message flow.

E. BOOTSTRAPPING

A bootstrapping process sequence immediately runs after D is active and gets an internet connection. Figure 8 describes the message flow between D and BS after TLS 1.3 full handshake finished. Both D and BS get K_{DBS} from TLS 1.3 full handshake procedure (Appendix A). D generates the message's header Hdr and tags Tag according to HTTPS protocol. It fetches the hardware model HW_D, operating system (OS) name OS_D, OS version V_D, and nonce value N_D to construct the bootstrapping request message $AReq$ (Step 1). After receiving the request message, BS generates New Session Ticket NST, then sends it to D (Steps 2). After verifying the request message, BS generates TLS 1.3 related Hdr and Tag, timestamp t_{CI} and reporting level $RLvl$. After fetching CI_D, BS sends an $AResp$ message (Step 3). Finally, D obtains the bootstrapping data from BS and executes it.

V. SECURITY ANALYSIS

We present a security analysis of the ARPKI-SZTP scheme in terms of the integrity and confidentiality of bootstrapping data and the accountability of bootstrapping process. This section consists of three subsections: Formal Analysis, Security Assessment, and Computational Overhead.

The Formal Analysis subsection aims to analyze the integrity and confidentiality of bootstrapping data based on secure mutual authentication of the proposed scheme. First, we analyze the $IDevID$ Certificate Registration to show that the Manufacturer can verify the integrity of the new $IDevID$ certificate based on the signatures of $CA1$, $CA2$, and $ILS1$. Second, we analyze the Credential Exchanges to show that the Manufacturer and Provider can confirm the confidentiality and the integrity of the bootstrapping-related information exchanged between them. Third, we analyze

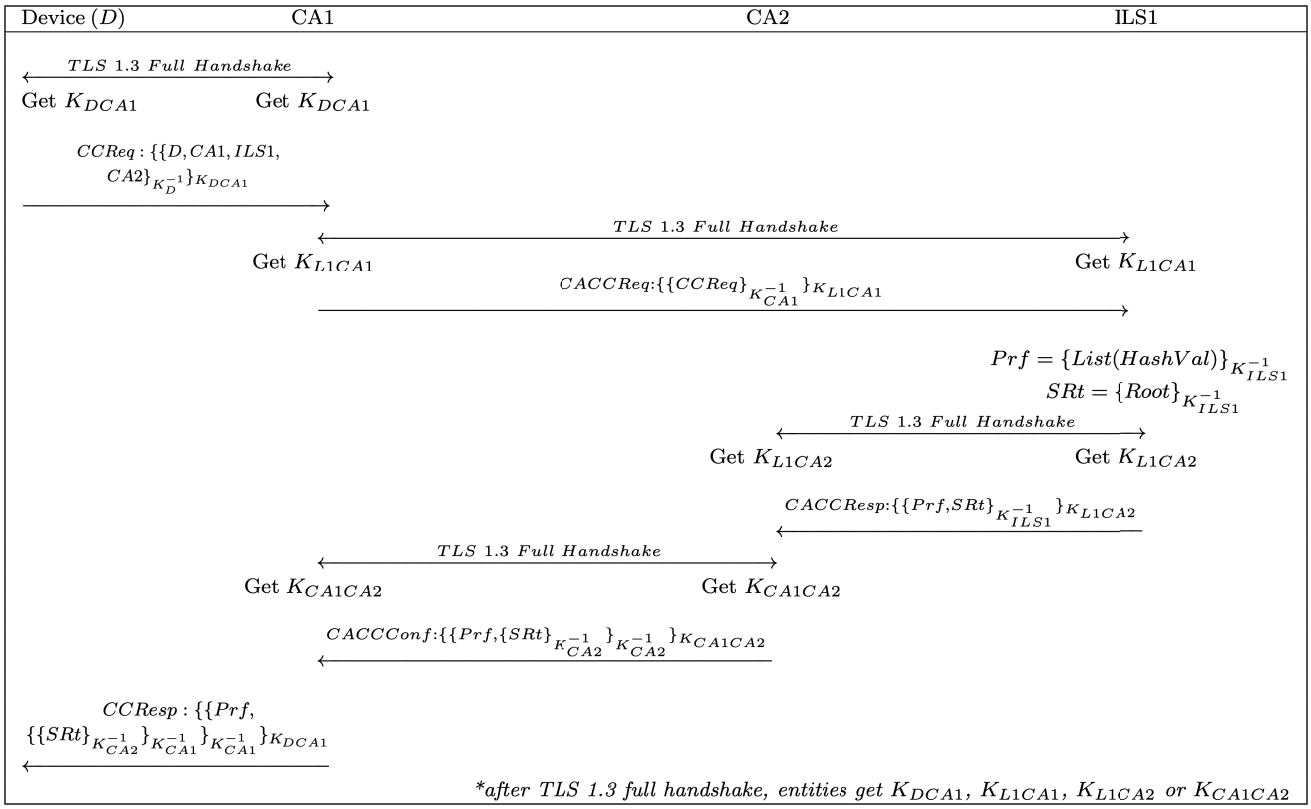


FIGURE 6. Certificate log validation message flow.

Bootstrapping to show that D and BS can authenticate each other by verifying the ARPKI's certificates, and D can securely obtain bootstrapping data from BS.

In the Security Assessment subsection, we employ the Microsoft STRIDE Threat Modeling Tool to identify potential insider threats against accountability. We conduct an informal analysis of the investigation and mitigation of insider threats based on the attack-resilient capability in our proposed scheme. Lastly, in the Computational Overhead subsection, we compare the computational overhead of the proposed ARPKI-SZTP scheme and the Single Trust RCA-SZTP scheme and discuss the computational overhead in correlation with accountability.

A. FORMAL ANALYSIS

We perform formal analysis by using BAN logic [33]. This logic allows us to show step-by-step verification of messages received by each entity regarding the message's signature and encryption. The signature and the encryption are two important components to provide the integrity and confidentiality of information. We use an extended form of BAN Logic to cover modern PKI-based protocols, which use asymmetric authentication, i.e., the extension form by Gaarder and Snekkenes [34], and the improvement by Sufatrio and Yap [35]. However, this logic cannot uncover all possible attacks in cryptographic protocol [36]. The BAN logic and its derivatives assume that all entities in the protocols operate honestly without colluding with other parties. In our formal analysis, we follow the BAN logic and assume that no adversary compromises any entity. The notations and rules we use are listed in Appendix B, and since BAN logic has no notation for HMAC operation [37], we express $HMAC(Key, text)$ as $\langle text \rangle_{Key}$ [38]. To carry out the formal analysis, we define goals, idealized messages, and assumptions, then perform the verification processes shown in Appendix B, C, D, and E. An idealized message represents a message equipped with a formula that states the ''meaning'' of the message exchanged. For example, a message $\{H_{text}\}_{K_{\mathcal{A}}^{-1}}$ from $\mathcal{A}$ to $\mathcal{B}$ can be idealized as $\mathcal{A} \rightarrow \mathcal{B}$: $\{H_{text} : \langle text \rangle_{Key}\}_{K_{\mathcal{A}}^{-1}}$ since H_{text} denotes $HMAC(Key, text)$. Another purpose of the idealized message is to omit contents (e.g., ''$POST$'' message payload from D to BS in Figure 8 and Figure 14) that do not contribute to the recipient's $belief$.

Prior to the verification of the authentication protocol using log-based PKI, we extend the certificate validation rule to adopt the log-based verification. In TLS authentication steps, both entities D and BS send the ILS's proof Prf and signed root $\{\{SRt\}_{K_{CA2}^{-1}}\}_{K_{CA1}^{-1}}$ to each other together with their certificates. Note that we express an idealized form of SRt_Q as $\sigma(\Re((\Theta(t_1^Q, t_2^Q), Root_Q), all), K_{ILS1}^{-1}, K_{CA2}^{-1}, K_{C1}^{-1})$. In this idealized form, $(\Theta(t_1^Q, t_2^Q), Root_Q)$ denotes that ''the Merkle Root Head $Root_Q$ holds in the time interval (t_1, t_2)''. The correspondent $Prf_Q = \sigma(\Re((\Theta(t_1^Q, t_2^Q), List(HashVal_Q)), all), K_{ILS1}^{-1})$ is idealized form of $List(HashVal_Q)$ signed by

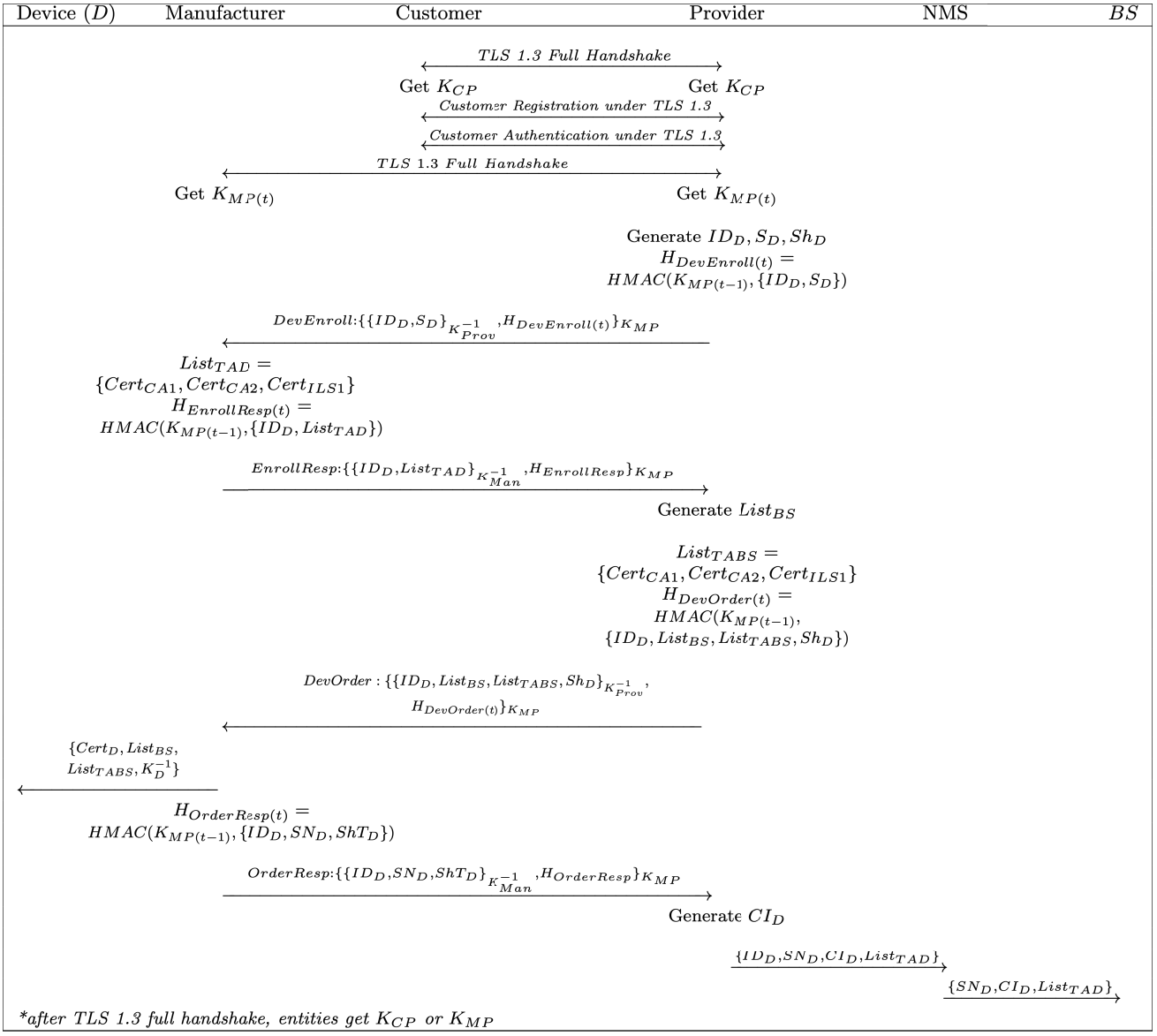


FIGURE 7. Credential exchanges message flow.

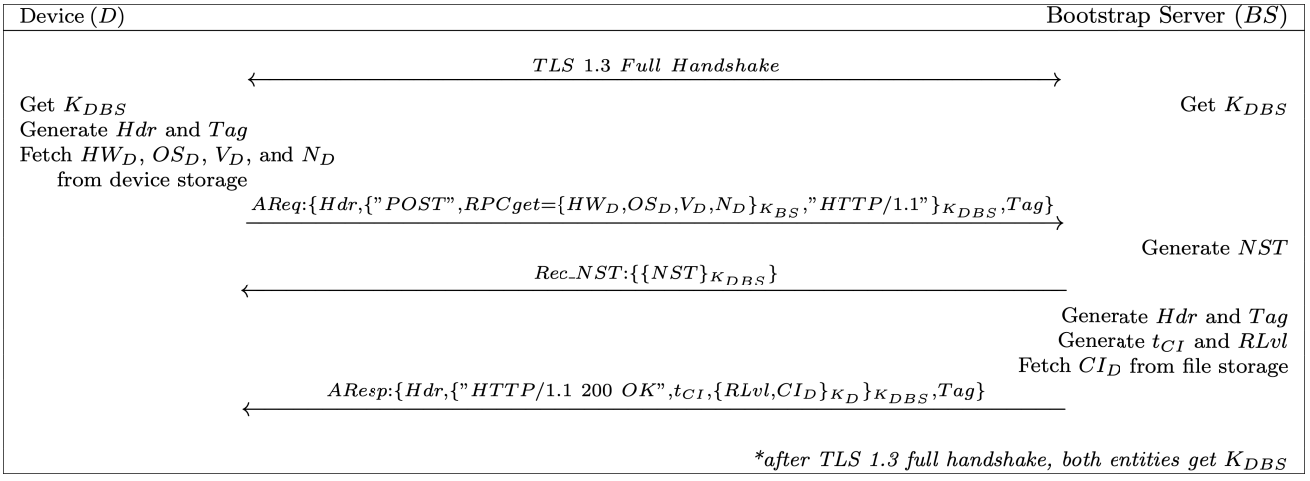


FIGURE 8. Bootstrapping message flow with the simplified version of session key.

$ILS1$ as an issuer who claims that the $List(HashVal_Q)$ is good in the time interval of t_1^Q and t_2^Q.

In the first analysis, we conduct the verification of messages between $CA1$ and Manufacturer in the *IDevID* Certificate Registration messages flow (Appendix C). This analysis aims to show that $CA1$ *believes* a new certificate statement C_D that the Manufacturer requests, and the Manufacturer *believes* that the *IDevID* certificate received from $CA1$ ($Cert_D$) is valid. The verification achieves all two goals. Goal 1 ($CA1 \models C_D$) shows that the $CA1$ can verify the Manufacturer as the issuer of C_D due to the Manufacturer's signature (St_2). Based on this *believes*, $CA1$ together with $CA2$ and ILSes process the registration of the new certificate registration C_D. After finishing the registration, $CA1$ sends the signed certificate $Cert_D$ to the Manufacturer. In Goal 2 ($Man \models Cert_D$), we show that the Manufacturer *believes* the validity of the $Cert_D$. To achieve this goal, the Manufacturer verifies the signatures of $CA1$ (St_9), $CA2$ (St_{10}), and $ILS1$ (St_{11}), and also checks whether the $Cert_D$ is within an active period (St_{12}).

We achieve all four goals in the second analysis of the Credential Exchanges message flow (Appendix D) between the Manufacturer and the Provider in terms of bootstrapping-related information exchanges. Goal 1 ($Man \models \{ID_D, S_D\}$) indicates that the Manufacturer *believes* that the *DevEnroll* message containing valid device's identifier ID_D and specification S_D. To support this statement, we show that after receiving the message, the Manufacturer verifies the encryption (St_1) and the Provider's signature (St_3). Based on the fact that the Manufacturer can verify the validity of the message's signature under the Provider's private key K_{Prov}^{-1}, and the correctness of the $H_{DevEnroll(t)}$, the Manufacturer *believes* that the ID_D and S_D is issued by the honest Provider (St_7). Then, after ensuring the fresh value of ID_D and S_D (St_8), which only the Provider can generate, the Manufacturer *believes* that the ID_D and S_D are valid.

Using the same logic as verification of Goal 1, we achieve Goal 2 ($Prov \models List_{TAD}$), Goal 3 ($Man \models \{List_{BS}, List_{TABS}, Sh_D\}$), and Goal 4 ($Prov \models \{SN_D, ShT_D\}$). Goal 2 indicates that the Provider *believes* that the list of *IDevID*'s trust anchor certificates is valid. This list is used by the bootstrap server *BS* to verify the signatures of the *IDevID* certificate. Goal 3 shows that the Manufacturer *believes* that $List_{BS}$, $List_{TABS}$, and Sh_D are valid and certainly issued by the honest Provider. Lastly, Goal 4 shows that the Provider receives a valid device's serial number SN_D and shipping tracking notification ShT_D from the Manufacturer.

Lastly, based on verification in Appendix E, we achieve all six goals of Bootstrapping as mentioned in Steps St_{24} ($D \models C_{BS}$), St_{29} ($D \models BS \models CV_{BS}$), St_{53} ($BS \models C_D$), St_{58} ($BS \models D \models CV_D$), St_{63} ($BS \models \{HW_D, OS_D, V_D, N_D\}$), and St_{68} ($D \models CI_D$). Goal 1 (St_{24}) shows that D can authenticate the BS's certificate statement C_{BS} by verifying the signatures (St_5–St_7), confirming the validity of $Root_{BS}$ (St_{11}–St_{17}) and $List(HashVal_{BS})$ (St_{19}–St_{21}), and ensuring that the C_{BS} is valid under the active certificate log (St_{22}). After validating the BS's certificate, D also confirms that the corresponding BS has a good private key K_{BS}^{-1} in Goal 2 (St_{26}–St_{29}).

On the other hand, Goal 3 (St_{53}) shows that BS can also authenticate the D's certificate by verifying the signatures (St_{34}–St_{36}), confirming the validity of $Root_D$ (St_{40}–St_{46}) and $List(HashVal_D)$ (St_{48}–St_{50}), and ensuring that the C_D is valid under the active certificate log (St_{51}). BS can also verify that D has a good private key K_D^{-1} as stated in Goal 4 (St_{55}–St_{58}). Finally, Goal 5 (St_{63}) expresses BS *believes* that the bootstrapping request *RPCget* which contains $\{HW_D, OS_D, V_D, N_D\}$ from D is correct due to encryption under $\{IVAP_D, KAP_D\}$ (St_{59}), and Goal 6 (St_{68}) shows D *believes* the encrypted bootstrapping data CI_D from BS is correct due to the encryption under $\{IVAP_{BS}, KAP_{BS}\}$ (St_{64}). In addition, BS sends the encrypted conveyed information $\{CI_D\}_{K_D}$ (St_{65}) to ensure that only the appropriate D can see the content.

B. SECURITY ASSESSMENT

This subsection provides an informal analysis by conducting a security assessment using Microsoft STRIDE Threat Modeling Tool to identify possible threats against our system. We get a list of possible threats for our proposed scheme through threat identification using Microsoft STRIDE. We examine the potential vulnerabilities and confirm the mitigations based on the list. We demonstrate the attack-resilient capability of accountable ARPKI and the accountability of certificates for mitigating the threats against our proposed scheme. As assumed in Section III, the adversary can compromise the long-term private key of some, but not all, entities.

We use Microsoft STRIDE Threat Modeling Tool GA release 7.3.20120.2 to identify threats from internal (insider threats) and external (outsider threats). In Microsoft STRIDE terminology, threats are categorized into six categories, each of which is sub-categorized into several threat types. This tool provides a template of threat categories with several threat types: Spoofing Category (7 threat types), Tampering Category (14 threat types), Repudiation Category (8 threat types), Information Disclosure Category (7 threat types), Denial of Service Category (5 threat types), and Elevation of Privilege Category (6 threat types). Based on these 47 threat types, this tool tests the security of our scheme diagram representing our system architecture for each data flow. This tool provides the potential threat type descriptions for adequate analysis and mitigation.

We make diagrams and identify threats using Microsoft STRIDE. Each diagram covers the minimum required stencils: one or more processes; the directional data flow between external interactors and processes or among the processes themselves; important data stores; an external interactor (e.g., external user); and a trust boundary. Microsoft STRIDE provides two types of boundaries: Arc Boundary (dotted line) and Border Boundary (dotted line rectangle). In our diagrams, we use the Arc Boundary to represent the Internet Boundary and the Border Boundary to represent the Manufacturer and

Provider Trust Boundaries. Data flow crossing the boundary line indicates passes through an insecure channel, while data flow within the rectangle line means the connection occurs through a secure channel. We build the diagrams based on our proposed scheme consisting of *IDevID* Certificate Registration, Credential Exchanges, and Bootstrapping. Each diagram contains data flows that cross the Internet Boundary, i.e., through a public network.

The *IDevID* Certificate Registration diagram shown in Figure 9 consists of five Generic Process stencils expressing Manufacturer, CA1, CA2, ILS1, and ILSn. All data flows are two-way HTTPS types crossing the Internet Boundaries.

In the Credential Exchanges diagram (Figure 10), we apply Generic Process stencil for Provider, NMS, and Manufacturer entities; Browser Client and Customer stencils; Data Store stencils representing Edge Device and Bootstrap Server. The Manufacturer Trust Boundary binds the Manufacturer and Edge Device. The Manufacturer connects to the Provider through the Internet Boundary. The Provider, NMS, and Bootstrap Server belong to the Provider Trust Boundary, to which the Customer can reserve an edge device via web access. The trust boundaries (Manufacturer and Provider Trust Boundaries) preserve the integrity and confidentiality of entities and data flow inside the boundaries. However, once an adversary can compromise the Manufacturer or the Provider, other entities in the same trust boundary will also be compromised. Furthermore, the Reservation, Enroll and Order, and Enrol and Order Resp data flows, which cross the Internet Boundary, become the threat spot for the adversary to attack our system.

Figure 11 illustrates the Bootstrapping diagram consisting of two Generic Process stencils of Edge Device and Bootstrap Server. The mutually authenticated HTTPS data flows connect both entities over the Internet Boundary.

Table 2 shows the number of threats found in all diagrams by the Property, Threat Category, and Diagram, obtained from the Threat List window of Microsoft STRIDE. The Microsoft STRIDE identifies 86 threats and splits them into 49, 23, and 14 threats for *IDevID* Certificate Registration, Credential Exchanges, and Bootstrapping, respectively. Based on the table, we can also show that no possible threat belongs to the Spoofing category due to the utilization of HTTPS communication through internet boundaries. Moreover, Tampering and Information Disclosure threats do not exist in *IDevID* Certificate Registration and Bootstrapping diagrams. This information supports the Formal Analysis in the previous subsection that our scheme achieves the integrity and confidentiality of each message flow due to mutual authentication among entities using TLS 1.3-based HTTPS.

We examine all threats and categorize them into 10 (Appendix F) out of 47 threat types from the Microsoft STRIDE threat template. We judge three out of the 10 identified threat types as insider threats:

- Insider threat-1: Destination claims that it did not receive data from a source outside the trust boundary. This threat belongs to the Repudiation Category.
- Insider threat-2: Source may be able to remotely execute code for Destination. This threat belongs to the Elevation Of Privilege Category.
- Insider threat-3: Destination may be able to impersonate the context of Source in order to gain additional privilege. This threat belongs to the Elevation Of Privilege Category.

This paper assumes that every attack by outsider parties (adversaries) aims to compromise entities by obtaining long-term private keys. The adversary can perform activities that deviate from the proposed protocol by compromising entities. As mentioned in the Adversary Model subsection, we categorize the attacks by compromised entities as insider attacks. Hereafter, we focus on investigating the three insider threats related to the accountability properties.

1) ANALYSIS OF INSIDER THREAT-1

The first possible insider threat is destination repudiation, where the destination claims it does not receive data from a source outside the trust boundary. A destination may claim no receipt of data from a source mainly in the following two cases: messages do not reach the destination due to Denial of Service (DoS) attacks, or the destination becomes dishonest or compromised. Regarding the first case, the DoS attack against the compromised destination results in the inactive destination and does not harm other entities, even if compromised. We do not categorize that case as an insider threat. In case of the DoS attack against the honest destination, it is necessary to apply the mitigation technique to detect and block illegitimate traffic and analyze the network bandwidth. The second case, where the dishonest or compromised destination claims no receipt of data, is related to accountability problems and belongs to the destination repudiation threat. In the following paragraphs, we investigate the second case in all processes of our proposed scheme.

a: INSIDER THREAT-1 IN IDevID CERTIFICATE REGISTRATION

Based on Table 2 and Figure 9, seven Repudiation threats identified in *IDevID* Certificate Registration means destination repudiation threats occur on all data flows in the *IDevID* Certificate Registration Diagram (Figure 9). Three threats occur in data flows among *CA*1, *CA*2, and *ILS*1, two threats occur between *ILS*1 and *ILSn*, and the last two threats occur between Manufacturer and *CA*1.

An attack-resilient mechanism monitors each other and ensures other entities work honestly among *CA*1, *CA*2, and *ILS*1, as described in [39]. In case *ILS*1 becomes compromised or dishonest and claims no message is received from *CA*1. *CA*2, which takes the validator's role in monitoring *ILS*1, can detect the misbehaving activity, then mitigate the failure immediately. If two out of three entities (e.g., *ILS*1 and *CA*2) are compromised, *ILS*1 may not respond to *CARegReq* message from *CA*1, and no misbehavior report from *CA*2. Due to no response from *ILS*1 and *CA*2, *CA*1 monitors

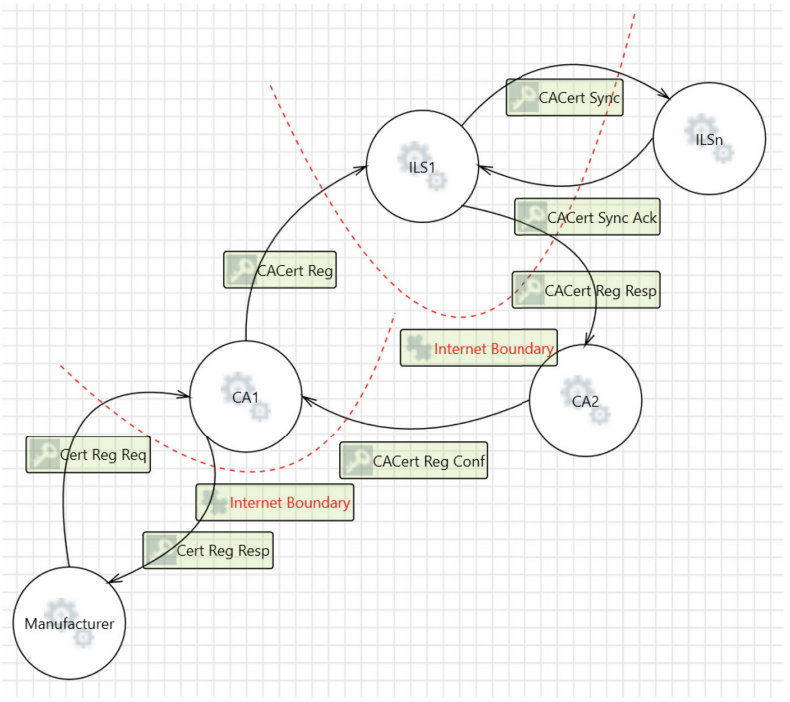


FIGURE 9. ***IDevID*** **certificate registration diagram.**

TABLE 2. **The evaluation result of threat identifications.**

Property	Threat Category	N_T per Diagram			N_T
		$IDevID$ Certificate Registration	Credential Exchanges	Bootstrapping	
Authentication	Spoofing	0	0	0	0
Integrity	Tampering	0	2	0	2
Non-Repudiation	Repudiation	7	3	2	12
Confidentiality	Information Disclosure	0	1	0	1
Availability	Denial Of Service	14	6	4	24
Authorization	Elevation Of Privilege	28	11	8	47
	SUM	49	23	14	86

N_T: Number of threat identifications

*CA*2 and detects the misbehavior of *CA*2, then resolves the problem. Subsequently, *CA*2 can mitigate the *ILS*1's violation.

Regarding the threat of destination repudiation between *ILS*1 and *ILSn*, the number of *ILS*es that respond to synchronization requests *SynReq* from *ILS*1 affects the quorum in reaching consensus. In particular, this consensus requires more than 50% of existing *ILS*es to ensure that only one certificate is registered for a specific domain. Some *ILS*es performing destination repudiation attacks may ignore *SynReq* messages from *ILS*1, and the number of *ILSes* involved in the quorum will be reduced. However, ARPKI requires at least one non-compromised *ILS* to participate in the quorum. Hence, if *ILSn* claims no receipt of *SynReq* messages, it does not interrupt the synchronization steps.

On the other hand, *ILS*1 may repudiate the *SynResp* from *ILSn* after sending the *SynReq* due to being compromised by an adversary. This condition also does not affect the number of quorums to reach a consensus. However, the *ILSn* cannot detect any misbehavior of *ILS*1 due to no authority to validate *ILS*1's activity. The role of monitoring and ensuring the misbehaving activities belongs to the corresponding *CA*2. *CA*2 can determine that *ILS*1 becomes dishonest or compromised when there is a protocol deviation of *ILS*1's activities.

The destination repudiation threat also exists between the Manufacturer and the *CA*1. Misbehaving *CA*1 may claim it does not receive certificate registration request *RegReq* from the Manufacturer. However, as described earlier, *CA*2 and *ILS*1 can determine and mitigate the misbehaving *CA*1. On the other hand, misbehaving Manufacturer may claim it does not receive the *RegResp* message containing the signed certificate from *CA*1. Note that the communication line between the Manufacturer and *CA*1 properly works because the Manufacturer successfully sent the *RegReq* message to *CA*1. In this case, *CA*1 can check the Manufacturer's certificate and whether or not any certificate update request from the Manufacturer during the certificate registration.

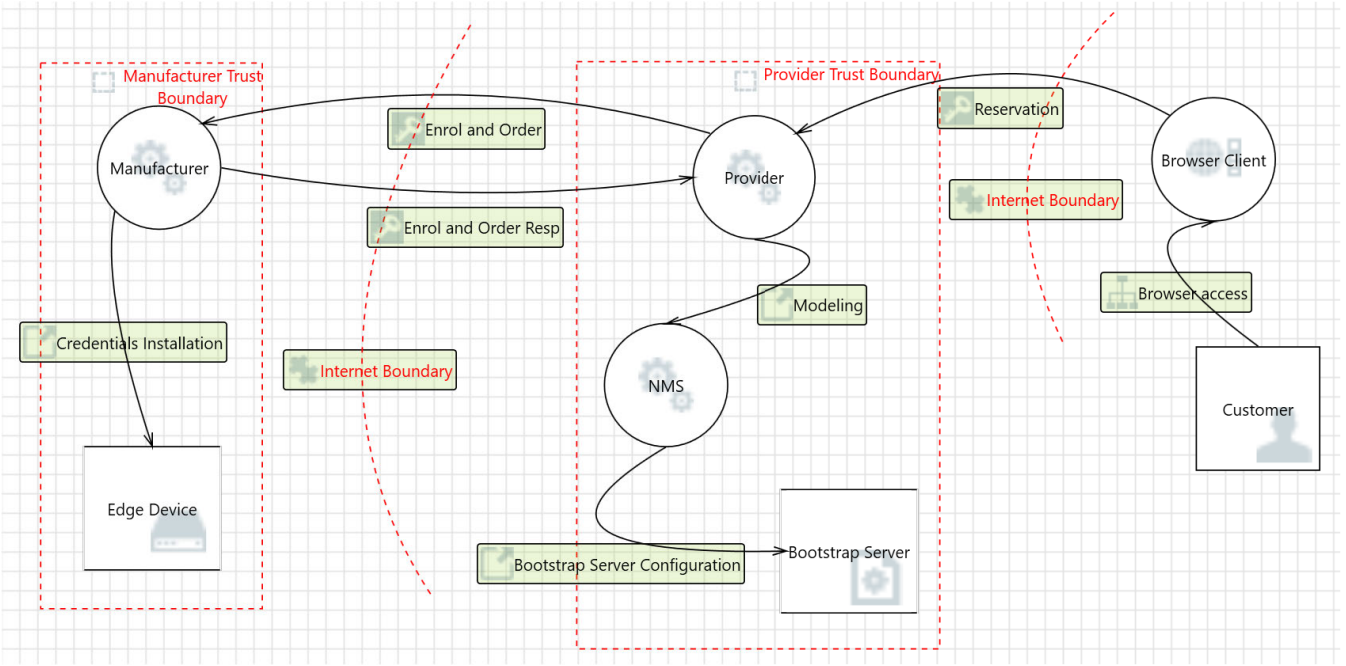


FIGURE 10. Credential exchanges diagram.

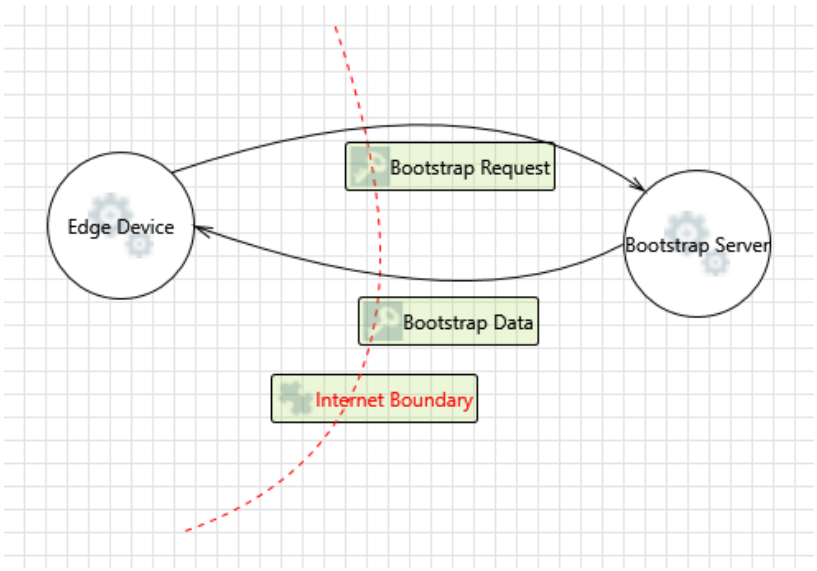


FIGURE 11. Bootstrapping diagram.

If no certificate update request exists, $CA1$ can determine whether the Manufacturer has become dishonest. On the other hand, if there is a certificate update request from the Manufacturer, then there is a possibility that an adversary has compromised the Manufacturer.

b: INSIDER THREAT-1 IN CREDENTIAL EXCHANGES

Based on Table 2 and Figure 10, three Repudiation threats of Credential Exchanges in Table 2 indicate destination repudiation threats are identified in all data flows of the Credential Exchanges Diagram (Figure 10). Two out of three occur between the Provider and the Manufacturer, and the other between the Browser Client and the Provider.

The destination repudiation threat may occur in data flow between the Provider as a source and the Manufacturer as a destination. Before establishing mutual authentication between them, each of them obtains ILS's proof Prf and signed root $\{\{SRt\}_{K_{CA2}^{-1}}\}_{K_{CA1}^{-1}}$ by performing Certificate Log Validation. During TLS 1.3 handshaking (Appendix A), the source and the destination send each other certificates along with the proof and the signed root. Based on these credentials, the source and destination can check whether

their respective certificates are correct or not. Since the ARPKI is accountable and the certificate log maintained by $ILS1$ is publicly verifiable, the source can trust the destination and vice versa, based on the validated certificate.

A Customer accesses the Provider's web page via Browser Client through TLS 1.3-based one-way authentication. The Browser Client can check the validity of the Provider's certificate through the proof and the signed root. Therefore, in this case, the Customer can fully trust the Provider if the Provider's certificate is legitimately contained in the certificate log.

c: INSIDER THREAT-1 IN BOOTSTRAPPING

Based on Table 2 and Figure 11, two Repudiation threats of Bootstrapping mean destination repudiation threats occur in all data flows of the Bootstrapping Diagram (Figure 11).

During the Bootstrapping process, the Edge Device and Bootstrap Server authenticate each other by relying on the proof and the signed root. The trust is established after verifying that the proof and the signed root validate the associated certificate. Prior to establishing mutual authentication between the two, the entities must have a valid certificate with a valid proof Prf and a signed root $\{\{SRt\}_{K_{CA2}^{-1}}\}_{K_{CA1}^{-1}}$. The Edge Device and the Bootstrap Server reach a conclusion that the other entity is trustworthy using the formal verification of **Goal 1** ($D \models C_{BS}$) and **Goal 3** ($BS \models C_D$) of the Bootstrapping process (Appendix E). In **Goal 1** verification, statement St_{22}: $D \models ILS1 \models \Phi(C_{BS})$ can be derived after verifying $h(h(C_{BS}) \parallel List(HashVal_{BS})) = Root_{BS}$. As well as in **Goal 3** verification, statement St_{50}: $BS \models ILS1 \models \Phi(C_D)$ can occur after $h(h(C_D) \parallel List(HashVal_D)) = Root_D$ is achieved. This result gives an evidence that the accountability of certificates is guaranteed in our proposed scheme.

2) ANALYSIS OF INSIDER THREAT-2

The second possible insider threat describes that a compromised source can remotely execute code for a destination that belongs to the Remote Code Execution (RCE) threat. This malicious activity can cause malware execution to the source gaining full access and control over the destination. An adversary usually performs RCE by an automated tool and rarely by manual attempt. The adversary must embed arbitrary code into the Host (in this case, compromised source) and send the code to the Target (destination) for the execution in the Target's machine. An honest Target can easily detect a protocol violation based on a suspicious message from the Host.

On the other hand, to launch the RCE attack successfully, the Host must hide its activities from Target. The adversary intends to insert a malicious program or code into Target's machine using permissible access without being detected by Target. As an external party, the Host requires full trust from Target to use the access to penetrate Target's resources. Basically, Target does not trust any other entities and does not become trust adversaries. Therefore, the adversary is almost impossible to launch a successful RCE attack in our proposed system.

In addition to being aware of the RCE threat, establishing trust among entities is a must for both the message's source and destination. Based on the discussion in the Analysis of insider threat-1 subsubsection, the accountability of certificates is guaranteed in our proposed scheme. Hence, each entity in our system can trust the message's source by relying on the ARPKI's attack resilience and the accountable certificate, which can also prevent the dishonest or compromised source.

3) ANALYSIS OF INSIDER THREAT-3

The third possible insider threat is an authorization problem where the dishonest or compromised destination of the data flow may be able to impersonate the context of the message source to gain additional privilege. To impersonate the context of the message source, an adversary typically requires the legitimate source's private key. We assume the adversary can compromise an entity's long-term private key but cannot reveal other honest entities' long-term private keys. Rather than stealing the private keys, the adversary must obtain a bogus but valid certificate to impersonate the entity successfully. Regarding the ARPKI-based certificate registration, an adversary cannot get a certificate by compromising even $n - 1$ trusted entities. Obtaining a fake certificate is possible only by compromising all n entities of the ARPKI core system. However, this action is difficult to achieve, and ARPKI validators can immediately detect the protocol violation by verifying the transparent certificate log.

In the following discussion, we demonstrate the scenario of impersonation without a bogus but valid message source's certificate. In each scenario, we assume an adversary compromises the message destination's long-term private key and then tries to impersonate the context of the message source towards another entity. The first scenario of impersonation is an adversary compromising a Manufacturer as the message destination of Enrol and Order data flow and then impersonating the context of the Provider towards $CA1$. Such a compromised Manufacturer is denoted by $\overline{Man}$. The $\overline{Man}$ must establish a TLS 1.3-based secure channel to conduct communication with other entities. However, TLS 1.3 provides a Perfect Forward Secrecy (PFS) feature that prevents an adversary from decrypting the previous or future sessions, even though the adversary compromises the long-term private key. This feature requires unique session keys that are generated frequently to protect each conversation.

If $\overline{Man}$ intends to perform session resumption, a short-term new session ticket (NST) from the last session is required in the Client Hello Message. However, the adversary cannot get the previous NST due to its inability to decrypt the last session. The $\overline{Man}$ can try to initiate a new session with $CA1$. However, $CA1$ will still require the session ticket from the last session. Hence, $\overline{Man}$ cannot build TLS 1.3 session with $CA1$.

If the adversary can establish a new session without any session ticket, the adversary intuitively updates the $\overline{Man}$'s certificate. Based on the Certificate Update and Revocation protocol, the $\overline{Man}$ sends an $\overline{UpdateReq}$ containing a $C_{\overline{Man}}{}'$ to $CA1$. $C_{\overline{Man}}{}'$ consists of the current active certificate statement C_{Man} and a new certificate statement $C_{\overline{Man}}$. For an appropriate update, the Manufacturer owner must satisfy the trust requirements of C_{Man} that were previously defined and registered in $ILS1$. Regarding these requirements, $ILS1$ tends to confirm the update request to the legitimate Manufacturer owner. If the owner declines the update request and reports the unrecognized request, $ILS1$ then cancels the $\overline{UpdateReq}$ and detects the misbehavior of $\overline{Man}$.

Instead of updating the certificate, it is enough for the adversary to conduct Certificate Log Validation to get the proof Prf and the signed root $\{\{SRt\}_{K_{CA2}^{-1}}\}_{K_{CA1}^{-1}}$ associated with $Cert_{Man}$. The adversary can directly use the Manufacturer's private key K_{Man}^{-1} to sign the message. Until this step, if the adversary can win all challenges, $\overline{Man}$ can establish communication to $CA1$ to request a new certificate registration.

Prior to generating a fake request of new certificate registration $\overline{RegReq}$, the $\overline{Man}$ generates K_D, K_D^{-1}, $\overline{SN_D}$, and fake values $\overline{ID_D}$, $\overline{S_D}$, and $\overline{C_D} = \{\overline{SN_D} \parallel \overline{ID_D} \parallel \overline{S_D} \parallel K_D\}$. The $\overline{Man}$ then sends $\overline{RegReq} = \{\{\overline{C_D}, CA1, ILS1, CA2\}_{K_{Man}^{-1}}\}_{K_{MCA1}}$ to $CA1$. The ARPKI system processes the request as long as it has not issued the same certificate. $CA1$ cannot authenticate the correctness of $\overline{SN_D}$, $\overline{ID_D}$, and $\overline{S_D}$, thus $CA1$ assumes that the contents of the $\overline{C_D}$ are valid. Until this step, no one detects the misbehaving activity of $\overline{Man}$. $\overline{Man}$ can get a bogus but valid certificate $\overline{Cert_D}$, then installs it together with the previous $List_{BS}$ and $List_{TABS}$ into a fake device $\overline{D}$.

On the other hand, Provider does not possess $\overline{SN_D}$, $\overline{ID_D}$, and $\overline{S_D}$, neither does NMS and Bootstrap Server. Hence, even using a bogus certificate, $\overline{Cert_D}$ cannot get the bootstrapping data. Moreover, Bootstrap Server can detect the misbehaving activity of $\overline{D}$ that tries to get bootstrapping data using fake identifiers. Henceforth, the Bootstrap Server may report the activity to the Provider to further investigate the Manufacturer that provides the bogus certificate to $\overline{D}$.

If $\overline{Man}$ tries to impose Provider to register the $\overline{SN_D}$, $\overline{ID_D}$, and $\overline{S_D}$ to Bootstrap Server, $\overline{Man}$ sends $\overline{EnrollResp} = \{\{\overline{ID_D}, List_{TAD}\}_{K_{\overline{Man}}^{-1}}, \overline{H_{EnrollResp}}\}_{K_{MP}}$ to Provider. However, Provider cannot authenticate $\overline{EnrollResp}$ due to incorrect $\overline{H_{EnrollResp}}$. Moreover, the Provider can detect the misbehaving activity of $\overline{Man}$. Based on this analysis, $\overline{Man}$ cannot impersonate the Provider to produce fake devices with bogus certificates and obtain bootstrapping data.

The second impersonation scenario is when $CA1$ becomes compromised ($\overline{CA1}$) and generates a fake $\overline{CARegReq}$ containing $\overline{RegReq} = \{\overline{C_D}, CA1, ILS1, CA2\}_{\overline{K_{Man}}^{-1}}$, then sends it to $ILS1$. $ILS1$ can easily detect and block the fake request due to an invalid Manufacturer's signature using a fake private key $\overline{K_{Man}}^{-1}$. The same situation will occur if the bogus request is initiated by other ARPKI entities ($CA2$, $ILS1$, or $ILSn$). Thus, every single entity in the ARPKI system is unlikely to perform an impersonation attack in the case of the $IDevID$ Certificate Registration process because of an invalid signature that is detected.

The third impersonation scenario may occur between the Provider and the Manufacturer in the Credential Exchange process. In this case, the possible scenario is the Provider as an adversary impersonating the Customer towards the Manufacturer. Similarly, after compromising the long-term private key of the Provider ($\overline{Prov}$), the adversary tries to break the PFS of TLS 1.3 and performs a Certificate Log Validation for long-term keypair (K_{Prov}, K_{Prov}^{-1}). $\overline{Prov}$ generates random $\overline{ID_D}$, $\overline{S_D}$, $\overline{Sh_D}$ for an arbitrary Customer, and $\overline{H_{DevEnroll(t)}} = HMAC(\overline{K_{MP(t-1)}}, \{\overline{ID_D}, \overline{S_D}\}_{K_{Prov}^{-1}})$. $\overline{Prov}$ uses a fake key $\overline{K_{MP(t-1)}}$ because the adversary cannot obtain the short-term last-session key of the Provider. Then, $\overline{Prov}$ sends a fake device enrollment $\overline{DevEnroll} = \{\{\overline{ID_D}, \overline{S_D}\}_{K_{\overline{Prov}}^{-1}}, \overline{H_{DevEnroll(t)}}\}_{K_{MP}}$ to the Manufacturer. Due to the valid signature of the Provider, the Manufacturer can successfully verify the $\overline{DevEnroll}$ message. However, after checking the message's contents, the honest Manufacturer cannot authenticate the $\overline{H_{DevEnroll(t)}}$ due to the fake last-session key usage. Therefore, the Manufacturer terminates the Credential Exchanges process and detects the misbehavior of $\overline{Prov}$.

In very limited opportunities, the adversary can attempt to register a bogus certificate by compromising all the ILSes and several CAs while maintaining two separate certificate logs (the correct and the bogus logs) under a set of honest CAs. The adversary can forge a legitimate certificate via a group of compromised CAs. At the same time, a manufacturer can still register a certificate for a device using the honest CAs. An improvement by applying gossiping protocol is required to overcome this split-view attack [39]. This protocol enables all entities, including domains, to arbitrarily exchange their knowledge about the certificate log. Hence, they possess a similar view of the certificate log with the expense of the raise of the computational overhead of all entities due to the continuous gossip.

Our scheme achieves a secure bootstrapping process through the certificate issued by the attack-resilient mechanism, mutual authentication using TLS 1.3 full handshake, and consistency checks in the proposed processes. As mentioned in [6], ARPKI's attack resilience capability can provide accountability in the certificate management. The accountability of bootstrapping process is improved by integrating the ARPKI certificate management together with TLS 1.3 mutual authentication into the SZTP framework. We demonstrated this property through the discussion in this subsection.

C. COMPUTATIONAL OVERHEAD

Table 3 compares the computational overhead between the proposed ARPKI-SZTP and Single RCA-based SZTP. We measure the computational overhead, which is associated

with the Decryption ($\mathcal{D}$), Encryption ($\mathcal{E}$), and Hashing ($\mathcal{H}$) operations. We consider the correlations between the number of devices (d), the number of Manufacturers (m), and the number of certificate domains (x) with the measurement. ARPKI application incurs additional costs in the *IDevID* Certificate Registration, Certificate Update and Revocation, and Certificate Log Validation processes. We can observe that the delay caused by computational overhead impacts certification-related processes, not the Bootstrapping processes.

Regarding the *IDevID* Certificate Registration, Certificate Update and Revocation, and Certificate Log Validation processes, our scheme requires up to five times larger computational overhead than Single RCA-based SZTP. An additional $16m\mathcal{H}$ of overhead occurs in the Credential Exchange process. Our proposed scheme does not cause extra delay for the Bootstrapping processes due to certificate pre-installation construction. Regarding the frequency of process occurrence, Certificate Log Validation, Credential Exchanges, and Bootstrapping occur most often. Among these processes, the Certificate Log Validation process costs more than the other two. In terms of accountability, the application of ARPKI, i.e., Certificate Log Validation, guarantees accountable Bootstrapping while increasing the computational overhead of Certificate Log Validation.

VI. CONCLUSION AND FUTURE WORK

This work presents the integration scheme of ARPKI and SZTP to provide accountable bootstrapping for edge devices in a zero-touch fashion with the integrity and confidentiality of bootstrapping data. The edge device can securely bootstrap data from an online trusted bootstrap server due to the ARPKI's certificate-based mutual authentication using TLS 1.3 full handshake. The ARPKI provides trustworthy certificate management in preventing misbehaving activities. The BAN Logic-based analysis proves that our scheme satisfies the integrity and confidentiality of bootstrapping data based on mutual authentication using ARPKI's certificates through the TLS 1.3-based communication channel. ARPKI's validation mechanism can maintain PKI's accountability to provide correct certificates for SZTP's bootstrapping process. In our informal analysis, we discuss various insider threats, including several impersonation attacks which can be prevented by ID management. Our protocol design achieves ID management with the use of HMAC. We also confirm the accountability by using Microsoft STRIDE to identify ten out of 47 possible threat types and conducting an investigation of three insider threats out of the ten potential threats.

We provide the computational overhead calculation as the cost spent to reach a certain level of security. Due to computational overhead, the extra delay occurs in certificate management-related processes. The proposed scheme can maintain accountability despite a slight increase in computational overhead.

We will implement the proposed scheme as a prototype of zero-touch device provisioning with a network-based security improvement for future work. We plan to extend the flexibility, scalability, and lightweight performance properties. And we are considering developing a secure zero-touch solution to support virtual machine orchestration.

APPENDIX A
TLS 1.3 FULL HANDSHAKE

Figure 12 presents a TLS 1.3 Full Handshake process between entities ($\mathcal{A}$) and ($\mathcal{B}$). This process occurs at the beginning of every communication between two parties in our scheme.

APPENDIX B
EXTENDED NOTATION AND RULES OF BAN LOGIC FOR PKI-BASED PROTOCOL

We provide some notations of BAN Logic and the extension for PKI-based protocol.

- $P \equiv X$: *P believes X*, or the principal P believe the annoucement X.
- $P \triangleleft X$: *P sees X* stated when someone has sent a message containing X to principal P.
- $P \mid\sim X$: *P once said X*. The principal P at some time sent a message including the statement X.
- $P \Rightarrow X$: P has *jurisdiction* over X. The principal P is an authority on X and should be trusted on this matter.
- $\#(X)$: The formula X is *fresh*; that is, X has not been sent in a message at any time before the current run of the protocol.
- $P \overset{K}{\leftrightarrow} Q$: P and Q may use the *shared key* K to communicate.
- $\overset{K}{\mapsto} P$: P has K as a *public key*. The matching *private key* (denoted K^{-1}) will never be discovered by any principal except P or a principal trusted by P.
- $P \overset{X}{=} Q$: The formula X is a *secret* known only to P and Q, and possibly to principals trusted by them.
- $\{X\}_K$: X *encrypted* under the key K. When K is a private key K^{-1}, this notation represents a signature of X.
- $\langle X \rangle_Y$: X *combined* with the formula Y.
- $\wp\kappa\ (P, K_P)$: P has associated a good public key K_P
- $\Pi\left(K_P^{-1}\right)$: P has a good private key K_P^{-1}
- $\sigma\left(X, K_P^{-1}\right)$: X is signed by P's private key K_P^{-1}
- $\left(\Theta\left(t_1^P, t_2^P\right), C_P\right)$: certificate statement C_P holds in the time interval $\left(t_1^P, t_2^P\right)$
- C_P: certificate statement $(\wp\kappa\ (P, K_P)\ , \Pi\left(K_P^{-1}\right))$ about P
- $\Phi(C_P)$: certificate statement C_P remains valid

Based on the logical postulates of the BAN Logic of authentication, we provide the rules we use in formal analysis.

Rule 1: Message-meaning rule for shared secret key:

$$\frac{P \equiv Q \overset{K}{\leftrightarrow} P, P \triangleleft \{X\}_K}{P \equiv Q \mid\sim X} \tag{3}$$

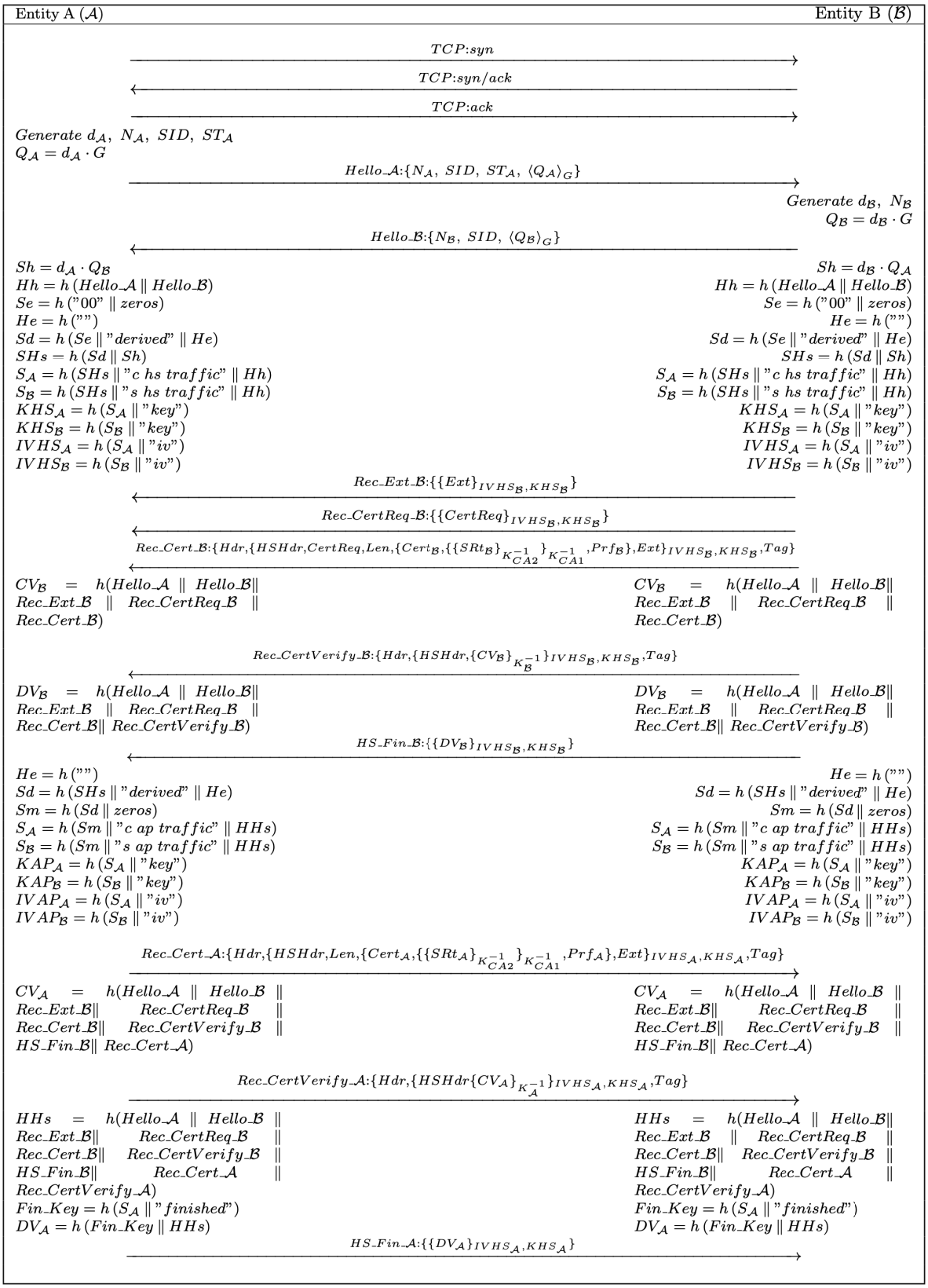


FIGURE 12. TLS 1.3 full handshake.

TABLE 3. Computational overhead.

Process	Scheme	
	Proposed ARPKI-SZTP	Single-Trust RCA-SZTP
$IDevID$ Certificate Registration	$20(\sum_{i=0}^{m} d_i)\mathcal{D} + 23(\sum_{i=0}^{m} d_i)\mathcal{E} + 8(\sum_{i=0}^{m} d_i)\mathcal{H}$	$4(\sum_{i=0}^{m} d_i)\mathcal{D} + 4(\sum_{i=0}^{m} d_i)\mathcal{E} + 2(\sum_{i=0}^{m} d_i)\mathcal{H}$
Certificate Update and Revocation	$16(\sum_{i=0}^{m} d_i)\mathcal{D} + 19(\sum_{i=0}^{m} d_i)\mathcal{E} + 4(\sum_{i=0}^{m} d_i)\mathcal{H}$	$4(\sum_{i=0}^{m} d_i)\mathcal{D} + 4(\sum_{i=0}^{m} d_i)\mathcal{E} + 2(\sum_{i=0}^{m} d_i)\mathcal{H}$
Certificate (Log) Validation	$9x\mathcal{D} + 14x\mathcal{E}$	$2x\mathcal{D} + 2x\mathcal{E}$
Credetial Exchange	$8(\sum_{i=0}^{m} d_i)(\mathcal{D} + \mathcal{E}) + 16m\mathcal{H}$	$8(\sum_{i=0}^{m} d_i)(\mathcal{D} + \mathcal{E})$
Bootstrapping	$6d(\mathcal{D} + \mathcal{E})$	$6d(\mathcal{D} + \mathcal{E})$

d: Number of devices; m: Number of manufacturers; x:Number of certificate domains; $\mathcal{D}$: Decryption/signing computational overhead; $\mathcal{E}$: Encryption/signature verification computational overhead; $\mathcal{H}$: Hashing computational overhead

P is said to *believes* Q, if P *believes* that K is shared with Q and P *sees* X is encrypted under K.

Rule 2: Message-meaning rule for shared secret formula:

$$\frac{P \models Q \overset{Y}{\rightleftharpoons} P, P \triangleleft \langle X \rangle_Y}{P \models Q \mid\sim X} \tag{4}$$

Rule 3: Message-meaning rule for signed message under private key:

$$\frac{P \models \wp\kappa\left(Q, K_Q\right), P \models \Pi\left(K_Q^{-1}\right), P \triangleleft \sigma\left(\Re(X,P), K_Q^{-1}\right)}{P \models Q \mid\sim X} \tag{5}$$

$P \triangleleft \sigma(\Re(X, P), K_Q^{-1})$ means P *sees* message X sent to P and signed by principal Q using the private key K_Q^{-1}.

Rule 4: Message-meaning rule for encryption message under public key:

$$\frac{P \models \wp\kappa(P, K_P), P \models \Pi(K_P^{-1}), P \triangleleft \{\mathcal{S}(X,Q)\}_{K_P}}{P \models Q \mid\sim X} \tag{6}$$

The third premise construction introduces the notion of the "stated sender" of a message. $\mathcal{S}(X, Q)$ means "message X together with Q as the stated sender of the message".

Rule 5: Session-key rule:

$$\frac{P \models \#(X), P \models Q \models X}{P \models Q \overset{K}{\longleftrightarrow} P} \tag{7}$$

If P and Q is said to *believe* X and value X is fresh, then P *believes* the session key K with Q.

Rule 6: Nonce-verification rule:

$$\frac{P \models \#(X), P \models Q \mid\sim X}{P \models Q \models X} \tag{8}$$

If P *believes* that X is expressed recently (freshness) and P *believes* that Q *once said* X, P *believes* that Q *believes* X.

Rule 7: Jurisdiction rule:

$$\frac{P \models Q \Rightarrow X, P \models Q \models X}{P \models X} \tag{9}$$

If P *believes* that Q has jurisdiction over X, and P *believes* that Q *believes* a message X, P also *believes* X.

Rule 8: See-rule:

$$\frac{P \triangleleft (X, Y)}{P \triangleleft X} \tag{10}$$

$$\frac{P \triangleleft \langle X \rangle_Y}{P \triangleleft X} \tag{11}$$

$$\frac{P \models Q \overset{K}{\longleftrightarrow} P, P \triangleleft \{X\}_K}{P \triangleleft X} \tag{12}$$

$$\frac{P \models \overset{K}{\mapsto} P, P \triangleleft \{X\}_K}{P \triangleleft X} \tag{13}$$

$$\frac{P \models \overset{K}{\mapsto} Q, P \triangleleft \{X\}_{K^{-1}}}{P \triangleleft X} \tag{14}$$

Rule 9: All recipient see-rule:

$$\frac{P \triangleleft \sigma\left(\Re(X, all), K_Q^{-1}\right)}{P \triangleleft \sigma\left(\Re(X,P), K_Q^{-1}\right)} \tag{15}$$

Rule 10: Freshness:

$$\frac{P \models \#(X)}{P \models \#(X,Y)} \tag{16}$$

If one part is known to be *fresh*, the entire formula must be *fresh*.

Rule 11: Certificate definition:

$$Cert_P = \sigma(\Re((\Theta(t_1^P, t_2^P), \wp\kappa(P, K_P), \Pi(K_P^{-1})), all), K_I^{-1}) \tag{17}$$

Certificate of P ($Cert_P$) contains information for all recipients that the P's public key K_P holds in the time interval (t_1^P, t_2^P), and within that time, P owns the corresponding private key K_P^{-1} secretly. Within the time interval (t_1^P, t_2^P), the signature of a principal I, as the certificate issuer, is also valid.

Rule 12: Certificate validation:

$$\frac{P \models I \mid\sim \left(\Theta\left(t_1^Q, t_2^Q\right), C_Q\right), P \models I \models \Delta\left(t_1^Q, t_2^Q\right), P \models I \models \Phi(C_Q)}{P \models I \models C_Q} \tag{18}$$

This Certificate validation rule derives a belief on a certificate $Cert_R$. $\Delta(t_1^Q, t_2^Q)$ denotes "good interval time" (t_1^Q) and (t_2^Q).

$P \models I \models \Phi(C_Q)$ expresses that P *believes* that I still *believes* that the uttered certificate statement C_Q remains valid.

APPENDIX C
VERIFICATION OF *IDevID* CERTIFICATE REGISTRATION

We conduct a verification of *IDevID* certificate registration message flow (Figure 4) between Manufacturer (*Man*) and *CA*1. We do not include all messages related to certificate registration with the assumption that all entities operate the protocol honestly, explained in the FORMAL ANALYSIS subsection. By verifying the initial and final messages between Man and *CA*1, we can achieve the goal of the logical analysis as follows.

Goal 1. $CA1 \models C_D$
Goal 2. $Man \models Cert_D$

Then, we idealized the communication messages of our proposed protocol between *Man* and *CA*1 as follows.

M_1: $Man \rightarrow CA1$: $\{\{C_D\}_{K_{Man}^{-1}}\}_{K_{MCA1}}$
M_2: $CA1 \rightarrow Man$: $\{Cert_D\}_{K_{MCA1}}$

In the original message flow in Figure 4, *RegReq* contains not only certificate statement C_D but also the destination nodes of the ARPKI core system. In the idealized messages, we remove the content of destination nodes because it does not contribute to the *CA*1's *belief*. Then, we define the verification assumptions as follows.

A_1: $CA1 \models Man \overset{K_{MCA1}}{\longleftrightarrow} CA1$
A_2: $Man \models CA1 \overset{K_{MCA1}}{\longleftrightarrow} Man$
A_3: $CA1 \models C_{Man}$
A_4: $Man \models C_{ILS1}$
A_5: $Man \models \overset{K_{CA1}}{\longmapsto} CA1$
A_6: $Man \models \overset{K_{CA2}}{\longmapsto} CA2$
A_7: $CA1 \models \#(C_D)$
A_8: $CA1 \models Man \Rightarrow C_D$
A_9: $Man \models ILS1 \models \Delta(t_1^D, t_2^D)$
A_{10}: $Man \models ILS1 \models \Phi(C_D)$

Lastly, we construct the verification states as follows.
From M_1 we get

St_1: $CA1 \triangleleft \{\{C_D\}_{K_{Man}^{-1}}\}_{K_{MCA1}}$

By combining St_1 and A_1, then applying statement (12) of **Rule 8**, we can get

St_2: $CA1 \triangleleft \{C_D\}_{K_{Man}^{-1}}$

We can construct idealized form of St_2 as

$CA1 \triangleleft \sigma(\Re(C_D, CA1), K_{Man}^{-1})$

By combining St_2 and A_3, then applying **Rule 3**, we can get

St_3: $CA1 \models Man \mid\sim C_D$

By combining St_3 and A_7, then applying **Rule 6**, we can get

St_4: $CA1 \models Man \models C_D$

By combining St_4 and A_8, then applying **Rule 7**, we can get

St_5: $CA1 \models C_D$ (**Goal 1**)

From M_2 we get

St_6: $Man \triangleleft \{Cert_D\}_{K_{MCA1}}$

By combining St_6 and A_2, then applying statement (12) of **Rule 8**, we can get

St_7: $Man \triangleleft Cert_D$

By applying **Rule 11** on St_7, we get

St_8: $Man \triangleleft \sigma(\Re((\Theta(t_1^D, t_2^D), \wp\kappa(D, K_D), \Pi(K_D^{-1})), all), K_{ILS1}^{-1}, K_{CA2}^{-1}, K_{CA1}^{-1})$

Since C_D denotes as $\wp\kappa(D, K_D), \Pi(K_D^{-1}))$, we can construct St_8 as

$Man \triangleleft \sigma(\Re((\Theta(t_1^D, t_2^D), C_D), all), K_{ILS1}^{-1}, K_{CA2}^{-1}, K_{CA1}^{-1})$

By applying **Rule 9** on St_8, we can get

St_9: $Man \triangleleft \sigma(\Re((\Theta(t_1^D, t_2^D), C_D), Man), K_{ILS1}^{-1}, K_{CA2}^{-1}, K_{CA1}^{-1})$

By combining St_9 and A_5, then applying statement (14) of **Rule 8**, we can get

St_{10}: $Man \triangleleft \sigma(\Re((\Theta(t_1^D, t_2^D), C_D), Man), K_{ILS1}^{-1}, K_{CA2}^{-1})$

By combining St_{10} and A_6, then applying statement (14) of **Rule 8**, we can get

St_{11}: $Man \triangleleft \sigma(\Re((\Theta(t_1^D, t_2^D), C_D), Man), K_{ILS1}^{-1})$

By combining St_{11} and A_4, then applying **Rule 3**, we can get

St_{12}: $Man \models ILS1 \mid\sim (\Theta(t_1^D, t_2^D), C_D)$

By combining St_{12}, A_9, and A_{10}, then applying **Rule 12**, we can get

St_{13}: $Man \models ILS1 \models C_D$

Statement St_{13} derives

St_{14}: $Man \models Cert_D$ (**Goal 2**)

APPENDIX D
VERIFICATION OF CREDENTIAL EXCHANGES

We conduct a verification of credential exchanges message flow (Figure 7) between Manufacturer (*Man*) and Provider (*Prov*). First, we state the analysis goals as follows.

Goal 1. $Man \models \{ID_D, S_D\}$
Goal 2. $Prov \models List_{TAD}$
Goal 3. $Man \models \{List_{BS}, List_{TABS}, Sh_D\}$
Goal 4. $Prov \models \{SN_D, ShT_D\}$

We idealized the communication messages of our proposed protocol to ease the analysis between *Man* and *Prov* as mentioned in Figure 13. Then, we define the verification assumptions as follows.

A_1: $Man \models Prov \overset{K_{MP}}{\longleftrightarrow} Man$
A_2: $Prov \models Man \overset{K_{MP}}{\longleftrightarrow} Prov$
A_3: $Man \models \overset{K_{Prov}}{\longmapsto} Prov$
A_4: $Prov \models \overset{K_{Man}}{\longmapsto} Man$
A_5: $Man \models Prov \overset{K_{MP(t-1)}}{\leftrightharpoons} Man$
A_6: $Prov \models Man \overset{K_{MP(t-1)}}{\leftrightharpoons} Prov$
A_7: $Man \models \#(ID_D, SN_D)$
A_8: $Prov \models \#(ID_D, SN_D)$
A_9: $Man \models Prov \Rightarrow \{ID_D, S_D, Sh_D\}$
A_{10}: $Prov \models Man \Rightarrow \{SN_D, List_{TAD}\}$

Finally, we construct the verification states as follows.
From M_1 we get

St_1: $Man \triangleleft \{\{ID_D, S_D\}_{K_{Prov}^{-1}}, H_{DevEnroll(t)}\}_{K_{MP}}$

By combining St_1 and A_1, then applying statement (12) of **Rule 8**, we can get

St_2: $Man \triangleleft \{\{ID_D, S_D\}_{K_{Prov}^{-1}}, H_{DevEnroll(t)}\}$

By applying statement (10) of **Rule 8** on St_2, we can get

M_1: $Prov \rightarrow Man$: $\{\{ID_D, S_D\}_{K^{-1}_{Prov}}, H_{DevEnroll(t)} : \langle ID_D, S_D\rangle_{K_{MP(t-1)}}\}_{K_{MP}}$
M_2: $Man \rightarrow Prov$: $\{\{ID_D, List_{TAD}\}_{K^{-1}_{Man}}, H_{EnrollResp(t)} : \langle ID_D, List_{TAD}\rangle_{K_{MP(t-1)}}\}_{K_{MP}}$
M_3: $Prov \rightarrow Man$: $\{\{ID_D, List_{BS}, List_{TABS}, Sh_D\}_{K^{-1}_{Prov}}, H_{DevOrder(t)} : \langle ID_D, List_{BS}, List_{TABS}, Sh_D\rangle_{K_{MP(t-1)}}\}_{K_{MP}}$
M_4: $Man \rightarrow Prov$: $\{\{ID_D, SN_D, ShT_D\}_{K^{-1}_{Man}}, H_{OrderResp(t)} : \langle ID_D, SN_D, ShT_D\rangle_{K_{MP(t-1)}}\}_{K_{MP}}$

FIGURE 13. Idealized message flow of credential exchanges.

St_3: $Man \triangleleft \{ID_D, S_D\}_{K^{-1}_{Prov}}$ and
St_4: $Man \triangleleft H_{DevEnroll(t)}$
By combining St_3 and A_3, then applying statement (14) of **Rule 8**, we can get
St_5: $Man \triangleleft \{ID_D, S_D\}$
Since $H_{DevEnroll(t)} = HMAC(K_{MP(t-1)}, \{ID_D, S_D\})$, we can construct St_4 as
St_6: $Man \triangleleft \langle\{ID_D, S_D\}\rangle_{K_{MP(t-1)}}$
By combining A_5 and St_6, then applying **Rule 2**, we can get
St_7: $Man \models Prov \mid\sim \{ID_D, S_D\}$
By applying **Rule 10** on A_7, we get
St_8: $Man \models \#(ID_D, S_D)$
By combining St_7 and St_8, then applying **Rule 6**, we can get
St_9: $Man \models Prov \models \{ID_D, S_D\}$
By combining St_9 and A_9, then applying **Rule 7**, we can get
St_{10}: $Man \models \{ID_D, S_D\}$ (**Goal 1**)
From M_2 we get
St_{11}: $Prov \triangleleft \{\{ID_D, List_{TAD}\}_{K^{-1}_{Man}}, H_{EnrollResp(t)}\}_{K_{MP}}$
By combining St_{11} and A_2, then applying statement (12) of **Rule 8**, we can get
St_{12}: $Prov \triangleleft \{\{ID_D, List_{TAD}\}_{K^{-1}_{Man}}, H_{EnrollResp(t)}\}$
By applying statement (10) of **Rule 8** on St_{12}, we can get
St_{13}: $Prov \triangleleft \{ID_D, List_{TAD}\}_{K^{-1}_{Man}}$, and
St_{14}: $Prov \triangleleft H_{EnrollResp(t)}$
By combining St_{13} and A_4, then applying statement (14) of **Rule 8**, we can get
St_{15}: $Prov \triangleleft \{ID_D, List_{TAD}\}$
Since $H_{EnrollResp(t)} = HMAC(K_{MP(t-1)}, \{ID_D, List_{TAD}\})$, we can construct St_{14} as
St_{16}: $Prov \triangleleft \langle\{ID_D, List_{TAD}\}\rangle_{K_{MP(t-1)}}$
By combining A_6 and St_{16}, then applying **Rule 2**, we can get
St_{17}: $Prov \models Man \mid\sim \{ID_D, List_{TAD}\}$
By applying **Rule 10** on A_8, we get
St_{18}: $Prov \models \#(ID_D, List_{TAD})$
By combining St_{17} and St_{18}, then applying **Rule 6**, we can get
St_{19}: $Prov \models Man \models \{ID_D, List_{TAD}\}$
By combining St_{19} and A_{10}, then applying **Rule 7**, we can get
St_{20}: $Prov \models \{ID_D, List_{TAD}\}$
By breaking conjunction of St_{20}, we can get
St_{21}: $Prov \models List_{TAD}$ (**Goal 2**)
From M_3 we get
St_{22}: $Man \triangleleft \{\{ID_D, List_{BS}, List_{TABS}, Sh_D\}_{K^{-1}_{Prov}}, H_{DevOrder(t)}\}_{K_{MP}}$
By combining St_{22} and A_1, then applying statement (12) of **Rule 8**, we can get
St_{23}: $Man \triangleleft \{\{ID_D, List_{BS}, List_{TABS}, Sh_D\}_{K^{-1}_{Prov}}, H_{DevOrder(t)}\}$
By applying statement (10) of **Rule 8** on St_{23}, we can get
St_{24}: $Man \triangleleft \{ID_D, List_{BS}, List_{TABS}, Sh_D\}_{K^{-1}_{Prov}}$ and
St_{25}: $Man \triangleleft H_{DevOrder(t)}$
By combining St_{24} and A_3, then applying statement (14) of **Rule 8**, we can get
St_{26}: $Man \triangleleft \{ID_D, List_{BS}, List_{TABS}, Sh_D\}$
Since $H_{DevOrder(t)} = HMAC(K_{MP(t-1)}, \{ID_D, List_{BS}, List_{TABS}, Sh_D\})$, we can construct St_{25} as
St_{27}: $Man \triangleleft \langle\{ID_D, List_{BS}, List_{TABS}, Sh_D\}\rangle_{K_{MP(t-1)}}$
By combining A_5 and St_{27}, then applying **Rule 2**, we can get
St_{28}: $Man \models Prov \mid\sim \{ID_D, List_{BS}, List_{TABS}, Sh_D\}$
By applying **Rule 10** on A_7, we get
St_{29}: $Man \models \#(ID_D, List_{BS}, List_{TABS}, Sh_D)$
By combining St_{28} and St_{29}, then applying **Rule 6**, we can get
St_{30}: $Man \models Prov \models \{ID_D, List_{BS}, List_{TABS}, Sh_D\}$
By combining St_{30} and A_9, then applying **Rule 7**, we can get
St_{31}: $Man \models \{ID_D, List_{BS}, List_{TABS}, Sh_D\}$
By breaking conjunction of St_{31}, we can get
St_{32}: $Prov \models \{List_{BS}, List_{TABS}, Sh_D\}$ (**Goal 3**)
From M_4 we get
St_{33}: $Prov \triangleleft \{\{ID_D, SN_D, ShT_D\}_{K^{-1}_{Man}}, H_{OrderResp(t)}\}_{K_{MP}}$
By combining St_{33} and A_2, then applying statement (12) of **Rule 8**, we can get
St_{34}: $Prov \triangleleft \{\{ID_D, SN_D, ShT_D\}_{K^{-1}_{Man}}, H_{OrderResp(t)}\}$
By applying statement (10) of **Rule 8** on St_{34}, we can get
St_{35}: $Prov \triangleleft \{ID_D, SN_D, ShT_D\}_{K^{-1}_{Man}}$ and
St_{36}: $Prov \triangleleft H_{OrderResp(t)}$
By combining St_{35} and A_4, then applying statement (14) of **Rule 8**, we can get
St_{37}: $Prov \triangleleft \{ID_D, SN_D, ShT_D\}$
Since $H_{OrderResp(t)} = HMAC(K_{MP(t-1)}, \{ID_D, SN_D, ShT_D\})$, we can construct St_{36} as
St_{38}: $Prov \triangleleft \langle\{ID_D, SN_D, ShT_D\}\rangle_{K_{MP(t-1)}}$
By combining A_6 and St_{38}, then applying **Rule 2**, we can get
St_{39}: $Prov \models Man \mid\sim \{ID_D, SN_D, ShT_D\}$
By applying **Rule 10** on A_8, we get
St_{40}: $Prov \models \#(ID_D, SN_D, ShT_D)$
By combining St_{39} and St_{40}, then applying **Rule 6**, we can get
St_{41}: $Prov \models Man \models \{ID_D, SN_D, ShT_D\}$
By combining St_{41} and A_{10}, then applying **Rule 7**, we can get
St_{42}: $Prov \models \{ID_D, SN_D, ShT_D\}$
By breaking conjunction of St_{42}, we can get
St_{43}: $Prov \models \{SN_D, ShT_D\}$ (**Goal 4**)

APPENDIX E
VERIFICATION OF BOOSTRAPPING

We conduct a verification of message flow (Figure 14) between device D and bootstrap server BS. First, we state the analysis goals as follows.

Goal 1. $D \models C_{BS}$
Goal 2. $D \models BS \models CV_{BS}$
Goal 3. $BS \models C_D$
Goal 4. $BS \models D \models CV_D$
Goal 5. $BS \models \{HW_D, OS_D, V_D, N_D\}$
Goal 6. $D \models CI_D$

Then, we idealized the communication messages of our proposed protocol to ease the analysis between D and BS as mentioned in Figure 15. Note that we include the certificate record messages and certificate verify record messages of TLS 1.3 full handshake (as $M1$—$M4$). In this verification, we concentrate on showing the authentication of certificates and bootstrapping data, so we remove other message payloads. We define the verification assumptions as follows.

Assumption

A_1: $D \models BS \xleftrightarrow{IVHS_{BS}, KHS_{BS}} D$
A_2: $BS \models D \xleftrightarrow{IVHS_D, KHS_D} BS$
A_3: $D \models BS \xleftrightarrow{IVAP_{BS}, KAP_{BS}} D$
A_4: $BS \models D \xleftrightarrow{IVAP_D, KAP_D} BS$
A_5: $D \models \overset{K_{CA1}}{\longmapsto} CA1$
A_6: $BS \models \overset{K_{CA1}}{\longmapsto} CA1$
A_7: $D \models \overset{K_{CA2}}{\longmapsto} CA2$
A_8: $BS \models \overset{K_{CA2}}{\longmapsto} CA2$
A_9: $D \models C_{ILS1}$
A_{10}: $BS \models C_{ILS1}$
A_{11}: $D \models \#(t_1^{BS}, t_2^{BS})$
A_{12}: $BS \models \#(t_1^{D}, t_2^{D})$
A_{13}: $D \models ILS1 \models \Phi(Root_{BS}, List(HashVal_{BS}))$
A_{14}: $BS \models ILS1 \models \Phi(Root_D, List(HashVal_D))$
A_{15}: $D \models ILS1 \Rightarrow C_{BS}$
A_{16}: $BS \models ILS1 \Rightarrow C_D$
A_{17}: $D \models C_D$
A_{18}: $D \models \#(CI_D)$
A_{19}: $D \models BS \Rightarrow CI_D$
A_{20}: $BS \models \#(HW_D, OS_D, V_D, N_D)$
A_{21}: $BS \models D \Rightarrow \{HW_D, OS_D, V_D, N_D\}$

Finally, we construct the verification states as follows.

From M_1 we get

St_1: $D \triangleleft \{Cert_{BS}, \{\{SRt_{BS}\}_{K_{CA2}^{-1}}\}_{K_{CA1}^{-1}}, Prf_{BS}\}_{IVHS_{BS}, KHS_{BS}}$

By combining St_1 and A_1, then applying statement (12) of **Rule 8**, we can get

St_2: $D \triangleleft \{Cert_{BS}, \{\{SRt_{BS}\}_{K_{CA2}^{-1}}\}_{K_{CA1}^{-1}}, Prf_{BS}\}$

By applying statement (10) of **Rule 8** on St_2, we can get

St_3: $D \triangleleft Cert_{BS}$

By applying **Rule 11** on St_3, we get

St_4: $D \triangleleft \sigma(\Re((\Theta(t_1^{BS}, t_2^{BS}), \wp\kappa(BS, K_{BS}), \Pi(K_{BS}^{-1})), all), K_{ILS1}^{-1}, K_{CA2}^{-1}, K_{CA1}^{-1})$

Since C_{BS} denotes as $\wp\kappa(BS, K_{BS}), \Pi(K_{BS}^{-1})$, we can formulate St_4 as

$D \triangleleft \sigma(\Re((\Theta(t_1^{BS}, t_2^{BS}), C_{BS}), all), K_{ILS1}^{-1}, K_{CA2}^{-1}, K_{CA1}^{-1})$

By applying **Rule 9** on St_4, we can get

St_5: $D \triangleleft \sigma(\Re((\Theta(t_1^{BS}, t_2^{BS}), C_{BS}), D), K_{ILS1}^{-1}, K_{CA2}^{-1}, K_{CA1}^{-1})$

By combining St_5 and A_5, then applying statement (14) of **Rule 8**, we can get

St_6: $D \triangleleft \sigma(\Re((\Theta(t_1^{BS}, t_2^{BS}), C_{BS}), D), K_{ILS1}^{-1}, K_{CA2}^{-1})$

By combiningSt_6 and A_7, then applying statement (14) of **Rule 8**, we can get

St_7: $D \triangleleft \sigma(\Re((\Theta(t_1^{BS}, t_2^{BS}), C_{BS}), D), K_{ILS1}^{-1})$

By combining St_7 and A_9, then applying **Rule 3**, we can get

St_8: $D \models ILS1 \mid\sim (\Theta(t_1^{BS}, t_2^{BS}), C_{BS})$

By breaking the conjunctions of St_8, we get

St_9: $D \models ILS1 \mid\sim C_{BS}$ (**Sub-Goal 1.1**)

By applying statement (10) of **Rule 8** on St_2, we can get

St_{10}: $D \triangleleft \{\{SRt_{BS}\}_{K_{CA2}^{-1}}\}_{K_{CA1}^{-1}}$

Since $SRt_{BS} = \{Root_{BS}\}_{K_{ILS1}^{-1}}$ and it is a signed message, we can construct St_{10} as

$D \triangleleft \sigma(\Re((\Theta(t_1^{BS}, t_2^{BS}), Root_{BS}), all), K_{ILS1}^{-1}, K_{CA2}^{-1}, K_{CA1}^{-1})$

By applying **Rule 9** on St_{10}, we can get

St_{11}: $D \triangleleft \sigma(\Re((\Theta(t_1^{BS}, t_2^{BS}), Root_{BS}), D), K_{ILS1}^{-1}, K_{CA2}^{-1}, K_{CA1}^{-1})$

By combining St_{11} and A_5, then applying statement (14) of **Rule 8**, we can get

St_{12}: $D \triangleleft \sigma(\Re((\Theta(t_1^{BS}, t_2^{BS}), Root_{BS}), D), K_{ILS1}^{-1}, K_{CA2}^{-1})$

By combining St_{12} and A_7, then applying statement (14) of **Rule 8**, we can get

St_{13}: $D \triangleleft \sigma(\Re((\Theta(t_1^{BS}, t_2^{BS}), Root_{BS}), D), K_{ILS1}^{-1})$

By combining St_{13} and A_9, then applying **Rule 3**, we can get

St_{14}: $D \models ILS1 \mid\sim (\Theta(t_1^{BS}, t_2^{BS}), Root_{BS})$

By breaking conjunctions on St_{14}, we can get

St_{15}: $D \models ILS1 \mid\sim \Delta(t_1^{BS}, t_2^{BS})$

By combining St_{15} and A_{11}, then applying **Rule 6**, we can get

St_{16}: $D \models ILS1 \models \Delta(t_1^{BS}, t_2^{BS})$

By combining St_{14}, St_{16}, and A_{13}, then applying **Rule 12**, we can get

St_{17}: $D \models ILS1 \models Root_{BS}$ (**Sub-Goal 1.2**)

By applying statement (10) of **Rule 8** on St_2, we can get

St_{18}: $D \triangleleft Prf_{BS}$

Since Prf_{BS} is a signed message of $\{List(HashVal_{BS})\}_{K_{ILS1}^{-1}}$, we can formulate St_{18} as

$D \triangleleft \sigma(\Re((\Theta(t_1^{BS}, t_2^{BS}), List(HashVal_{BS})), all), K_{ILS1}^{-1})$

By applying **Rule 9** on St_{18}, we can get

St_{19}: $D \triangleleft \sigma(\Re((\Theta(t_1^{BS}, t_2^{BS}), List(HashVal_{BS})), D), K_{ILS1}^{-1})$

By combining St_{19} and A_9, then applying **Rule 3**, we can get

St_{20}: $D \models ILS1 \mid\sim (\Theta(t_1^{BS}, t_2^{BS}), List(HashVal_{BS}))$

By combining St_{16}, St_{20}, and A_{13}, then applying **Rule 12**, we can get

St_{21}: $D \models ILS1 \models List(HashVal_{BS})$ (**Sub-Goal 1.3**)

By considering St_9, St_{17}, and St_{21}, if and only if $h(h(C_{BS}) \parallel List(HashVal_{BS})) = Root_{BS}$ holds, we can construct

St_{22}: $D \models ILS1 \models \Phi(C_{BS})$

By combining St_8, St_{15}, and St_{22}, then applying **Rule 12**, we can get

St_{23}: $D \models ILS1 \models C_{BS}$ By combining St_{23} and A_{15}, then

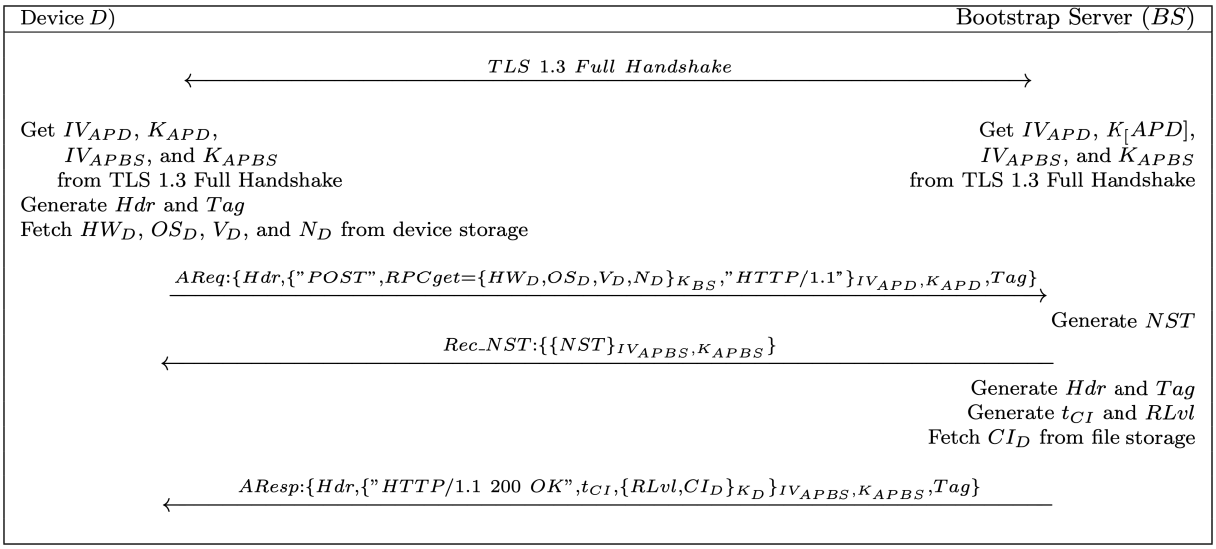


FIGURE 14. Bootstrapping message flow with the original version of session key.

$$
\begin{aligned}
&M_1\colon BS \rightarrow D\colon \{Cert_{BS}, \{\{SRt_{BS}\}_{K_{CA2}^{-1}}\}_{K_{CA1}^{-1}}, Prf_{BS}\}_{IVHS_{BS},KHS_{BS}}\\
&M_2\colon BS \rightarrow D\colon \{\{CV_{BS}\}_{K_{BS}^{-1}}\}_{IVHS_{BS},KHS_{BS}}\\
&M_3\colon D \rightarrow BS\colon \{Cert_{D}, \{\{SRt_{D}\}_{K_{CA2}^{-1}}\}_{K_{CA1}^{-1}}, Prf_{D}\}_{IVHS_{D},KHS_{D}}\\
&M_4\colon D \rightarrow BS\colon \{\{CV_{D}\}_{K_{D}^{-1}}\}_{IVHS_{D},KHS_{D}}\\
&M_5\colon D \rightarrow BS\colon \{\{HW_D, OS_D, V_D, N_D\}_{K_{BS}}\}_{IVAP_D,KAP_D}\\
&M_6\colon BS \rightarrow D\colon \{t_{CI}, \{RLvl, CI_D\}_{K_D}\}_{IVAP_{BS},KAP_{BS}}
\end{aligned}
$$

FIGURE 15. Idealized message flow of Bootstrapping.

applying **Rule 7**, we can get

St_{24}: $D \models C_{BS}$ (**Goal 1**)

By defining C_{BS} as $\wp\kappa\ (BS, K_{BS})$, $\Pi\left(K_{BS}^{-1}\right)$, we can formulate St_{24} as

$D \models \wp\kappa\ (BS, K_{BS})$ and $D \models \Pi\left(K_{BS}^{-1}\right)$

Then, from M_2 we get

St_{25}: $D \triangleleft \{\{CV_{BS}\}_{K_{BS}^{-1}}\}_{IVHS_{BS},KHS_{BS}}$

By combining St_{25} and A_1, then applying statement (12) of **Rule 8**, we can get

St_{26}: $D \triangleleft \{CV_{BS}\}_{K_{BS}^{-1}}$

We can construct idealized form of St_{26} as

$D \triangleleft \sigma(\Re(CV_{BS}, D), K_{BS}^{-1})$

By combining St_{24} and St_{26}, then applying **Rule 3**, we can get

St_{27}: $D \models BS \mid\sim CV_{BS}$

Note that each D and BS construct the message context CV_{BS} from previous message flow of TLS 1.3 full handshake process. By comparing the CV_{BS} from BS and its own CV_{BS}, D cannot authenticate BS if the value is not identical. Otherwise,

St_{28}: $D \models \#(CV_{BS})$

By combining St_{27} and St_{28}, then applying **Rule 6**, we can get

St_{29}: $D \models BS \models CV_{BS}$ (**Goal 2**)

Then, from M_3 we get

St_{30}: $BS \triangleleft \{Cert_D, \{\{SRt_D\}_{K_{CA2}^{-1}}\}_{K_{CA1}^{-1}}, Prf_D\}_{IVHS_D,KHS_D}$

By combining St_{30} and A_2, then applying statement (12) of **Rule 8**, we can get

St_{31}: $BS \triangleleft \{Cert_D, \{\{SRt_D\}_{K_{CA2}^{-1}}\}_{K_{CA1}^{-1}}, Prf_D\}$

By applying statement (10) of **Rule 8** on St_{31}, we can get

St_{32}: $BS \triangleleft Cert_D$

By applying **Rule 11** on St_{32}, we get

St_{33}: $BS \triangleleft \sigma(\Re((\Theta(t_1^D, t_2^D), \wp\kappa(D, K_D), \Pi(K_D^{-1})), all), K_{ILS1}^{-1}, K_{CA2}^{-1}, K_{CA1}^{-1})$

Since C_D denotes as $\wp\kappa(D, K_D)$, $\Pi(K_D^{-1})$, we can formulate St_{33} as

$BS \triangleleft \sigma(\Re((\Theta(t_1^D, t_2^D), C_D), all), K_{ILS1}^{-1}, K_{CA2}^{-1}, K_{CA1}^{-1})$

By applying **Rule 9** on St_33, we can get

St_{34}: $BS \triangleleft \sigma(\Re((\Theta(t_1^D, t_2^D), C_D), BS), K_{ILS1}^{-1}, K_{CA2}^{-1}, K_{CA1}^{-1})$

By combining St_{34} and A_6, then applying statement (14) of **Rule 8**, we can get

St_{35}: $BS \triangleleft \sigma(\Re((\Theta(t_1^D, t_2^D), C_D), BS), K_{ILS1}^{-1}, K_{CA2}^{-1})$

By combining St_{35} and A_8, then applying statement (14) of **Rule 8**, we can get

St_{36}: $BS \triangleleft \sigma(\Re((\Theta(t_1^D, t_2^D), C_D), BS), K_{ILS1}^{-1})$

By combining St_{36} and A_{10}, then applying **Rule 3**, we can get

St_{37}: $BS \models ILS1 \mid\sim (\Theta(t_1^D, t_2^D), C_D)$

By breaking the conjunctions of St_{37}, we get

TABLE 4. Identified possible threat types.

Threat Category	Threat Type Description
Tampering	Attackers who can send a series of packets or messages may be able to overlap data. For example, packet 1 may be 100 bytes starting at offset 0. Packet 2 may be 100 bytes starting at offset 25. Packet 2 will overwrite 75 bytes of packet 1. Ensure you reassemble data before filtering it, and ensure you explicitly handle these sorts of cases.
	Packets or messages without sequence numbers or timestamps can be captured and replayed in a wide variety of ways. Implement or utilize an existing communication protocol that supports anti-replay techniques (investigate sequence numbers before timers) and strong integrity.
Repudiation	Destination claims that it did not receive data from a source outside the trust boundary. Consider using logging or auditing to record the source, time, and summary of the received data.
Information Disclosure	Custom authentication schemes are susceptible to common weaknesses such as weak credential change management, credential equivalence, easily guessable credentials, null credentials, downgrade authentication or a weak credential change management system. Consider the impact and potential mitigations for your custom authentication scheme.
Denial of Service	An external agent interrupts data flowing across a trust boundary in either direction.
	Destination crashes, halts, stops or runs slowly; in all cases violating an availability metric.
Elevation Of Privilege	An attacker may pass data into Destination in order to change the flow of program execution within Destination to the attacker's choosing.
	Source may be able to remotely execute code for Destination.
	Cross-site request forgery (CSRF or XSRF) is a type of attack in which an attacker forces a user's browser to make a forged request to a vulnerable site by exploiting an existing trust relationship between the browser and the vulnerable web site. In a simple scenario, a user is logged in to web site A using a cookie as a credential. The other browses to web site B. Web site B returns a page with a hidden form that posts to web site A. Since the browser will carry the user's cookie to web site A, web site B now can take any action on web site A, for example, adding an admin to an account. The attack can be used to exploit any requests that the browser automatically authenticates, e.g. by session cookie, integrated authentication, IP whitelisting. The attack can be carried out in many ways such as by luring the victim to a site under control of the attacker, getting the user to click a link in a phishing email, or hacking a reputable web site that the victim will visit. The issue can only be resolved on the server side by requiring that all authenticated state-changing requests include an additional piece of secret payload (canary or CSRF token) which is known only to the legitimate web site and the browser and which is protected in transit through SSL/TLS. See the Forgery Protection property on the flow stencil for a list of mitigations.
	Destination may be able to impersonate the context of Source in order to gain additional privilege.

St_{38}: $BS \models ILS1 \sim C_D$ (**Sub-Goal 3.1**)

By applying statement (10) of **Rule 8** on St_{31}, we can get

St_{39}: $BS \triangleleft \{\{SRt_D\}_{K_{CA2}^{-1}}\}_{K_{CA1}^{-1}}$

Since $SRt_D = \{Root_D\}_{K_{ILS1}^{-1}}$ and it is a signed message, we can formulate St_{39} as

$BS \triangleleft \sigma(\Re((\Theta(t_1^D, t_2^D), Root_D), all), K_{ILS1}^{-1}, K_{CA2}^{-1}, K_{CA1}^{-1})$

By applying **Rule 9** on St_{39}, we can get

St_{40}: $BS \triangleleft \sigma(\Re((\Theta(t_1^D, t_2^D), Root_D), BS), K_{ILS1}^{-1}, K_{CA2}^{-1}, K_{CA1}^{-1})$

By combining St_{40} and A_6, then applying statement (14) of **Rule 8**, we can get

St_{41}: $BS \triangleleft \sigma(\Re((\Theta(t_1^D, t_2^D), Root_D), BS), K_{ILS1}^{-1}, K_{CA2}^{-1})$

By combining St_{41} and A_8, then applying statement (14) of **Rule 8**, we can get

St_{42}: $BS \triangleleft \sigma(\Re((\Theta(t_1^D, t_2^D), Root_D), BS), K_{ILS1}^{-1})$

By combining St_{42} and A_{10}, then applying **Rule 3**, we can get

St_{43}: $BS \models ILS1 \sim (\Theta(t_1^D, t_2^D), Root_D)$

By breaking conjunctions on St_{43}, we can get

St_{44}: $BS \models ILS1 \sim \Delta(t_1^D, t_2^D)$

By combining St_{44} and A_{12}, then applying **Rule 6**, we can get

St_{45}: $BS \models ILS1 \models \Delta(t_1^D, t_2^D)$

By combining St_{43}, St_{45}, and A_{14}, then applying **Rule 12**, we can get

St_{46}: $BS \models ILS1 \models Root_D$ (**Sub-Goal 3.2**)

By applying statement (10) of **Rule 8** on St_{31}, we can get

St_{47}: $BS \triangleleft Prf_D$

Since Prf_D is a signed message of $\{List(HashVal_D)\}_{K_{ILS1}^{-1}}$, we can formulate St_{47} as

$BS \triangleleft \sigma(\Re((\Theta(t_1^D, t_2^D), List(HashVal_D)), all), K_{ILS1}^{-1})$

By applying **Rule 9** on St_{47}, we can get

St_{48}: $BS \triangleleft \sigma(\Re((\Theta(t_1^D, t_2^D), List(HashVal_D)), BS), K_{ILS1}^{-1})$

By combining St_{48} and A_{10}, then applying **Rule 3**, we can get

St_{49}: $BS \models ILS1 \sim (\Theta(t_1^D, t_2^D), List(HashVal_D))$

By combining St_{47}, St_{49}, and A_{14}, then applying **Rule 12**, we can get

St_{50}: $BS \models ILS1 \models List(HashVal_D)$ (**Sub-Goal 3.3**)

By considering St_{38}, St_{46}, and St_{50}, if and only if $h(h(C_D) \parallel List(HashVal_D)) = Root_D$ holds, we can construct

St_{51}: $BS \models ILS1 \models \Phi(C_D)$

By combining St_{37}, St_{45}, and St_{51}, then applying **Rule 12**, we can get

St_{52}: $BS \models ILS1 \models C_D$

By combining St_{52} and A_{16}, then applying **Rule 7**, we can get

St_{53}: $BS \models C_D$ (**Goal 3**)

By defining C_D as $\wp\kappa\,(D, K_D)$, $\Pi\left(K_D^{-1}\right)$, we can formulate St_{53} as

$BS \models \wp\kappa\,(D, K_D)$ and $BS \models \Pi\left(K_D^{-1}\right)$

Then, from M_4 we get

St_{54}: $BS \triangleleft \{\{CV_D\}_{K_D^{-1}}\}_{IVHS_D,KHS_D}$

By combining St_{54} and A_2, then applying statement (12) of **Rule 8**, we can get

St_{55}: $BS \triangleleft \{CV_D\}_{K_D^{-1}}$

We can construct idealized form of St_{55} as

$BS \triangleleft \sigma(\mathfrak{R}(CV_D, BS), K_D^{-1})$

By combining St_{53} and St_{55}, then applying **Rule 3**, we get

St_{56}: $BS \models D \mid\sim CV_D$

Note that each D and BS construct the message context CV_D from previous message flow of TLS 1.3 full handshake process. By comparing the CV_D from D and its own CV_D, BS cannot authenticate D if the value is not identical. Otherwise,

St_{57}: $BS \models \#(CV_D)$

By combining St_{56} and St_{57}, then applying **Rule 6**, we can get

St_{58}: $BS \models D \models CV_D$ (**Goal 4**)

Then, from M_5 we get

St_{59}: $BS \triangleleft \{\{HW_D, OS_D, V_D, N_D\}_{K_{BS}}\}_{IVAP_D,KAP_D}$

By combining St_{59} and A_4, then applying statement (12) of **Rule 8**, we can get

St_{60}: $BS \triangleleft \{HW_D, OS_D, V_D, N_D\}_{K_{BS}}$

Since $\{HW_D, OS_D, V_D, N_D\}_{K_{BS}}$ is a message encrypted by D using public key K_{BS}, we can formulate St_{60} as

$BS \triangleleft \{\mathcal{S}(\{HW_D, OS_D, V_D, N_D\}, D)\}_{K_{BS}}$

By applying **Rule 4** on St_{60}, we can get

St_{61}: $BS \models D \mid\sim \{HW_D, OS_D, V_D, N_D\}$

By combining St_{61} and A_{20}, then applying **Rule 6**, we can get

St_{62}: $BS \models D \models RPCget$

By combining St_{62} and A_{21}, then applying **Rule 7**, we can get

St_{63}: $BS \models \{HW_D, OS_D, V_D, N_D\}$ (**Goal 5**)

Then, from M_6 we get

St_{64}: $D \triangleleft \{\{CI_D\}_{K_D}\}_{IVAP_{BS},KAP_{BS}}$

By combining St_{64} and A_3, then applying statement (12) of **Rule 8**, we can get

St_{65}: $D \triangleleft \{CI_D\}_{K_D}$

Since $\{CI_D\}_{K_D}$ is a message encrypted by BS using public key K_D, we can formulate St_{65} as

$D \triangleleft \{S(CI_D, BS)\}_{K_D}$

By combining St_{65} and A_{17}, then applying **Rule 4**, we can get

St_{66}: $D \models BS \mid\sim CI_D$

By combining St_{66} and A_{18}, then applying **Rule 6**, we can get

St_{67}: $D \models BS \models CI_D$

By combining St_{67} and A_{19}, then applying **Rule 7**, we can get

St_{68}: $D \models CI_D$ (**Goal 6**)

APPENDIX F
IDENTIFIED THREAT TYPES OF PROPOSED SCHEME

Table 4 contains the description of threat types that potentially harm our proposed scheme based on the Microsoft STRIDE Threat Modeling Tool.

REFERENCES

[1] L. Piccirillo, "Zero-touch disasters. KNP analysis for real-world CWMP deployments," M.S. thesis, School Ind. Inf. Eng., Politecnico di Milano, Milan, Italy, 2020. [Online]. Available: http://hdl.handle.net/10589/167099

[2] R. Kunnemann, I. Esiyok, and M. Backes, "Automated verification of accountability in security protocols," in *Proc. IEEE 32nd Comput. Secur. Found. Symp. (CSF)*, Jun. 2019, pp. 397–39716, doi: 10.1109/CSF.2019.00034.

[3] B. Laurie, "Certificate transparency," *Commun. ACM*, vol. 57, no. 10, pp. 40–46, Sep. 2014, doi: 10.1145/2659897.

[4] J. Sanchez-Gomez, D. Garcia-Carrillo, R. Marin-Perez, and A. Skarmeta, "Secure authentication and credential establishment in narrowband IoT and 5G," *Sensors*, vol. 20, no. 3, p. 882, Feb. 2020, doi: 10.3390/s20030882.

[5] A. Boudguiga, N. Bouzerna, L. Granboulan, A. Olivereau, F. Quesnel, A. Roger, and R. Sirdey, "Towards better availability and accountability for IoT updates by means of a blockchain," in *Proc. IEEE Eur. Symp. Secur. Privacy Workshops (EuroS&PW)*, Apr. 2017, pp. 50–58, doi: 10.1109/EuroSPW.2017.50.

[6] D. Basin, C. Cremers, T. H.-J. Kim, A. Perrig, R. Sasse, and P. Szalachowski, "ARPKI: Attack resilient public-key infrastructure," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2014, pp. 382–393, doi: 10.1145/2660267.2660298.

[7] S. Khan, L. Zhu, Z. Zhang, M. A. Rahim, K. Khan, and M. Li, "Attack-resilient TLS certificate transparency," *IEEE Access*, vol. 8, pp. 98958–98973, 2020, doi: 10.1109/ACCESS.2020.2996997.

[8] K. Watsen, I. Farrer, and M. Abrahamsson, *Secure Zero Touch Provisioning (SZTP)*, document RFC 8572, Apr. 2019. [Online]. Available: https://www.rfc-editor.org/info/rfc8572

[9] J. Höglund, S. Lindemer, M. Furuhed, and S. Raza, "PKI4IoT: Towards public key infrastructure for the Internet of Things," *Comput. Secur.*, vol. 89, Feb. 2020, Art. no. 101658, doi: 10.1016/j.cose.2019.101658.

[10] S. Maksuti, A. Bicaku, M. Zsilak, I. Ivkic, B. Peceli, G. Singler, K. Kovacs, M. Tauber, and J. Delsing, "Automated and secure onboarding for system of systems," *IEEE Access*, vol. 9, pp. 111095–111113, 2021, doi: 10.1109/ACCESS.2021.3102280.

[11] B. Pularikkal, S. Patil, S. Anantha, and S. Chakraborty, "Blockchain based Wi-Fi onboarding simplification, identity management and device profiling for IoT devices in enterprise networks," *Tech. Discl. Commons*, no. 1222, pp. 1–6, Jun. 2018. [Online]. Available: https://www.tdcommons.org/dpubs_series/1222

[12] N. Yousefnezhad, A. Malhi, and K. Främling, "Security in product lifecycle of IoT devices: A survey," *J. Netw. Comput. Appl.*, vol. 171, Dec. 2020, Art. no. 102779, doi: 10.1016/j.jnca.2020.102779.

[13] S. Belattaf, M. Mohammedi, M. Omar, and R. Aoudjit, "Reliable and adaptive distributed public-key management infrastructure for the Internet of Things," *Wireless Pers. Commun.*, vol. 120, no. 1, pp. 113–137, Sep. 2021, doi: 10.1007/s11277-021-08437-9.

[14] Intel Corporation. (Nov. 2019). *Zero-Touch Provisioning for Edge Devices and Software-Defined Networks*. Intel Corporation. Accessed: May 22, 2022. [Online]. Available: https://networkbuilders.intel.com/solutionslibrary/zero-touch-provisioning-for-edge-devices-and-software-defined-networks

[15] M. Carrer, L. LaChance, E. Asanghanwa, and J. Kohn. (Apr. 2021). *Securing the IoT Begins With Zero-Touch Provisioning at Scale*. Infineon Technologies AG. Accessed: Oct. 18, 2021. [Online]. Available: https://azure.microsoft.com/en-us/resources/secure-iot-begins-with-zero-touch-provisioning-at-scale/

[16] S. Boire, T. Akgün, P. Ginzboorg, P. Laitinen, S. Tamrakar, and T. Aura, "Credential provisioning and device configuration with EAP," in *Proc. 19th ACM Int. Symp. Mobility Manage. Wireless Access*, Nov. 2021, pp. 87–96, doi: 10.1145/3479241.3486705.

[17] P. Sousa, L. Magalhães, J. Resende, R. Martins, and L. Antunes, "Provisioning, authentication and secure communications for IoT devices on FIWARE," *Sensors*, vol. 21, no. 17, p. 5898, Sep. 2021, doi: 10.3390/s21175898.

[18] (Jul. 2020). *Trusted Platform Module (TPM): Trusted Computing Group (TPM)*. [Online]. Available: https://trustedcomputinggroup.org/work-groups/trusted-platform-module/

[19] D. Moghimi, B. Sunar, T. Eisenbarth, and N. Heninger, "TPM-FAIL: TPM meets timing and lattice attacks," in *Proc. 29th USENIX Secur. Symp. (USENIX Secur.)*, Aug. 2020, pp. 2057–2073. [Online]. Available: https://www.usenix.org/conference/usenixsecurity20/presentation/moghimi-tpm

[20] T. H.-J. Kim, L.-S. Huang, A. Perring, C. Jackson, and V. Gligor, "Accountable key infrastructure (AKI): A proposal for a public-key validation infrastructure," in *Proc. 22nd Int. Conf. World Wide Web (WWW)*, 2013, pp. 679–690, doi: 10.1145/2488388.2488448.

[21] P. Szalachowski, S. Matsumoto, and A. Perrig, "PoliCert: Secure and flexible TLS certificate management," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2014, pp. 406–417, doi: 10.1145/2660267.2660355.
[22] S. Khan, Z. Zhang, L. Zhu, M. Li, Q. G. K. Safi, and X. Chen, "Accountable and transparent TLS certificate management: An alternate public-key infrastructure with verifiable trusted parties," *Secur. Commun. Netw.*, vol. 2018, pp. 1–16, Jul. 2018, doi: 10.1155/2018/8527010.
[23] M. Y. Kubilay, M. S. Kiraz, and H. A. Mantar, "CertLedger: A new PKI model with certificate transparency based on blockchain," *Comput. Secur.*, vol. 85, pp. 333–352, Aug. 2019, doi: 10.1016/j.cose.2019.05.013.
[24] Z. Wang, J. Lin, Q. Cai, Q. Wang, D. Zha, and J. Jing, "Blockchain-based certificate transparency and revocation transparency," *IEEE Trans. Depend. Secure Comput.*, vol. 19, no. 1, pp. 681–697, Jan. 2022, doi: 10.1109/TDSC.2020.2983022.
[25] J. Chen, S. Yao, Q. Yuan, K. He, S. Ji, and R. Du, "Certchain: Public and efficient certificate audit based on blockchain for TLS connections," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, Apr. 2018, pp. 2060–2068, doi: 10.1109/INFOCOM.2018.8486344.
[26] M. Chase and S. Meiklejohn, "Transparency overlays and applications," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2016, pp. 168–179, doi: 10.1145/2976749.2978404.
[27] M. Al-Bassam and S. Meiklejohn, "Contour: A practical system for binary transparency," in *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, vol. 11025. Cham, Switzerland: Springer, 2018, pp. 94–110, doi: 10.1007/978-3-030-00305-0_8.
[28] B. Dowling, M. Fischlin, F. Günther, and D. Stebila, "A cryptographic analysis of the TLS 1.3 handshake protocol candidates," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2015, pp. 1197–1210, doi: 10.1145/2810103.2813653.
[29] G. Arfaoui, X. Bultel, P.-A. Fouque, A. Nedelcu, and C. Onete, "The privacy of the TLS 1.3 protocol," Cryptol. ePrint Arch., Tech. Rep. 2019/749, 2019. [Online]. Available: https://eprint.iacr.org/2019/749
[30] J. Hoyland, "An analysis of TLS 1.3 and its use in composite protocols," Ph.D. dissertation, Dept. Math., Roy. Holloway, Univ. London, Egham, U.K., 2018. [Online]. Available: https://core.ac.uk/download/pdf/195282008.pdf
[31] H. Lee, D. Kim, and Y. Kwon, "TLS 1.3 in practice: How TLS 1.3 contributes to the internet," in *Proc. Web Conf.*, Apr. 2021, pp. 70–79, doi: 10.1145/3442381.3450057.
[32] H. Krawczyk, M. Bellare, and R. Canetti, *HMAC: Keyed-Hashing for Message Authentication*, document RFC 2104, Feb. 1997. [Online]. Available: https://www.rfc-editor.org/info/rfc2104
[33] M. Burrows, M. Abadi, and R. Needham, "A logic of authentication," *ACM Trans. Comput. Syst.*, vol. 8, no. 1, pp. 18–36, 1990, doi: 10.1145/77648.77649.
[34] K. Gaarder and E. Snekkenes, "Applying a formal analysis technique to the CCITT X.509 strong two-way authentication protocol," *J. Cryptol.*, vol. 3, no. 2, pp. 81–98, Jan. 1991, doi: 10.1007/BF00196790.
[35] R. H. C. Yap, "Extending BAN logic for reasoning with modern PKI-based protocols," in *Proc. IFIP Int. Conf. Netw. Parallel Comput.*, Oct. 2008, pp. 190–197, doi: 10.1109/NPC.2008.86.
[36] J. Srinivas, S. Mukhopadhyay, and D. Mishra, "Secure and efficient user authentication scheme for multi-gateway wireless sensor networks," *Ad Hoc Netw.*, vol. 54, pp. 147–169, Jan. 2017, doi: 10.1016/j.adhoc.2016.11.002.
[37] W. Teepe, "On BAN logic and hash functions or: How an unjustified inference rule causes problems," *Auton. Agents Multi-Agent Syst.*, vol. 19, no. 1, pp. 76–88 2009, doi: 10.1007/s10458-008-9063-8.
[38] J. Zhou, M. Ma, Y. Feng, and T. N. Nguyen, "A symmetric key-based pre-authentication protocol for secure handover in mobile Wimax networks," *J. Supercomput.*, vol. 72, no. 7, pp. 2734–2751, Jul. 2016, doi: 10.1007/s11227-015-1581-y.
[39] D. Basin, C. Cremers, T. H.-J. Kim, A. Perrig, R. Sasse, and P. Szalachowski, "Design, analysis, and implementation of ARPKI: An attack-resilient public-key infrastructure," *IEEE Trans. Depend. Secure Comput.*, vol. 15, no. 3, pp. 393–408, May/Jun. 2018, doi: 10.1109/TDSC.2016.2601610.

DANU DWI SANJOYO received the B.Eng. and M.Eng. degrees in telecommunication engineering from Telkom University, Bandung, Indonesia, in 2010 and 2013, respectively. He is currently pursuing the Ph.D. degree in information security with the Division of Electrical Engineering and Computer Science, Graduate School of Natural Science and Technology, Kanazawa University, Japan. Since 2014, he has been a Lecturer at the School of Electrical Engineering, Telkom University. His research interests include information security, cryptography, automated networks, network orchestration, software-defined networking, traffic engineering, data center, and embedded systems.

MASAHIRO MAMBO (Member, IEEE) received the B.Eng. degree from Kanazawa University, Kanazawa, Japan, in 1988, and the M.S.Eng. and Dr.Eng. degrees in electronic engineering from the Tokyo Institute of Technology, Tokyo, Japan, in 1990 and 1993, respectively. In 2011, after working with the Japan Advanced Institute of Science and Technology, Tohoku University, Sendai, Japan, and the University of Tsukuba, Tsukuba, Japan, he joined Kanazawa University, where he is currently a Professor with the Faculty of Electrical, Information and Communication Engineering, Institute of Science and Engineering. His research interests include information security, software protection, and privacy protection. He has served as the Co-Editor-in-Chief for the *International Journal of Information Security*, the Steering Committee Chair for the International Conference on Information Security, and the Chair for the Technical Committee on Information Security in Engineering Sciences Society of the Institute of Electronics, Information and Communication Engineers (ISEC in ESS of IEICE).