University of Passau
Faculty of Computer Science and Mathematics
Chair for IT Security

Master Thesis in Informatics

Secure Bootstrapping and Post-Compromise Security in IoT

submitted by

Hossam Hamed

1. Examiner: Prof. Dr. Joachim Posegga
2. Examiner: Prof. Dr. Jorge Cuéllar
Date: March 31, 2022

Contents

Abstract

Please write a short abstract summarizing your work.

Acknowledgments

I would first like to thank ...

List of Figures

List of Tables

Acronyms

CSR Certificate Signing Request

MAC Message Authentication Code

OFMC Open-source Fixed-point Model Checker

SMP Socialist Millionaires' Protocol

TESLA Timed Efficient Stream Loss-tolerant Authentication

PKI Public Key Infrastructure

ABC Automated Border Control

DAS Document Authentication System

BVS Biometric Verification System

CSI Central Systems Interface

BGMS Border Guard Maintenance System

VMS Visa Management System

RTP Registered Traveler Program

EEMS Entry-Exit Management System

TCP Transmission Control Protocol

UDP User Datagram Protocol

IoT Internet of Things

TLS Transport Layer Security

DTLS Datagram Transport Layer Security

IETF Internet Engineering Task Force

CoAP Constrained Application Protocol

REST Representational state transfer

IDevID Initial Device Identifier

LDevID Locally Significant Device Identifier

P2P Peer-to-Peer

BRSKI Bootstrapping Remote Secure Key Infrastructure

SZTP Secure Zero Touch Provisioning

OOB Out-Of-Band

EAP-TLS Extensible Authentication Protocol TLS

GBA Generic Bootstrapping Architecture

EST Enrollment over Secure Transport

CA Certification Authority

MASA Manufacturer Authorized Signing Authority

RA Registration Authority

EE End Entity

DoS Denial of Service

SCRAM Salted Challenge Response Authentication Mechanism

TOFU Trust on first use

NEA Network Endpoint Assessment

CMP Certificate Managment Protocol

NMS Network Managment System

RESTCONF Representational State Transfer Configuration

OTR Off-the-Record

RKE ratcheted key exchange

NIST National Institute of Standards and Technology

EC Elliptic Curve

ECC Elliptic Curve Cryptography

ECDH Elliptic Curve Diffie-Hellmann

SIDH Supersingular isogeny Diffie-Hellman

KEM Key Encapsulation Mechanism

AKE authenticated key exchange

X3DH Extended Triple Diffie-Hellmann

OTP One Time Pre-key

IK Identity Key

0-RTT Zero Round Trip Time

KDF Key Derivation Function

DH Diffie-Hellmann

RK Root Key

CK Chain Key

HK Header Key

NHK Next Header Key

1 Introduction

Internet of Things (IoT) is a relatively new concept that already has applications in many domains, creating new ways of interactions between humans and small devices, referred to as smart devices. Applications like Smart Airports, Transportation, Home Automation, and many more are being revolutionized by this technology [1]. IoT is the deployment of network-connected constrained devices that interact with the physical environment by employing sensors to collect data, managing other systems, controlling actuators, as well as communicating with one another. IoT devices are classified into several classes depending on their degree of computing resources and power usage. Depending on their degree, they may be assumed to not be able to process sophisticated or even conventional cryptographic operations. In many use cases, the IoT devices handle critical and private data. This imposes the requirement for data integrity and authenticity guarantees, and -because of privacy- in many cases also confidentiality.

IoT device life cycle and nature of IoT communication

1.1 Research Questions and Contributions

research questions: motivate the desire for zero-touch bootstrapping and future secrecy.

The contributions of the thesis are the following:

- We conduct a comparative study between two secure bootstrapping protocols, Bootstrapping Remote Secure Key Infrastructure (BRSKI) and Secure Zero Touch Provisioning (SZTP).
- We give an overview over the X3DH protocol and the double ratchet algorithm. In addition to present a formal verification of the X3DH using Open-source Fixed-point Model Checker (OFMC). Moreover, we reflect on stand of those protocols in the post-quantum setting.
- We provide a demo implementation for X3DH and the double ratchet algorithm using Python.
- Although not strictly relevant, we present a comparison table between a set of certificate enrollment protocols in appendix B.

1.2 Structure of the Thesis

The structure of this thesis is as follows: In chapter 2, we reflect on principles relevant to the understanding of the work presented in further chapters. In addition to other works

related to our thesis. Next we propose real-world use cases in chapter 3 where the protocols to be proposed can be applied. In chapter 4, we give an overview of the two promising automated bootstrapping protocols: BRSKI and SZTP. Moreover, we present a comparative study between them. Chapter 5 discusses post-compromise security, and how it is achievable in an asynchronous environment through the use of X3DH and double ratcheting. In addition, we discuss the formal verification of the protocols and their stand in the world of quantum computing. In chapter 6 we present our demo implementation of the protocols discussed in chapter 4. Furthermore, chapter 7 discusses our findings and reflects the represented protocols on the use cases. Finally, chapter 8 concludes our work and introduces suggestions for future work.

2 Background

This chapter gives an overview over concepts and related required for understanding the protocols and algorithms discussed in further chapters of the thesis.

2.1 Preliminaries

2.1.1 Cryptography Concepts Overview

The practice and study of mechanisms for secure communication is known as cryptography. Secure communication in digital communications is the assurance of message privacy between communicating parties, even when the communication channel is untrusted, and adversaries are present. Cryptography is the process of creating and evaluating protocols with the prime objective of assuring data confidentiality, integrity, and authenticity between communicating parties. The fundamental basis of cryptography are mathematical theory and computer science practice. Cryptographic algorithms are built around assumptions about computational difficulty. While it is theoretically conceivable to break such algorithms in a well-designed system, any adversary would find it infeasible in practice.

Encryption is the operation of converting information (*cleartext*) using cryptographic algorithms into a form (*ciphertext*) that is unintelligible to the public or an adversary. The ciphertext can be restored to its original form by its intended recipient who owns the necessary cryptographic material to perform the decryption process. Encryption is a prominent and commonly utilized method to ensure confidentiality.

On the other hand, integrity is the assurance that data has not been intercepted and modified by an adversary or corrupted in transit. Mainly, the approach to ensure data integrity is through mathematically compute a digital fingerprint for the data via hash functions. Subsequently, the digital fingerprint is attached to the data and sent to their designated party. The receiver verifies that the fingerprint corresponds to the data received, and thus the data integrity is proven.

Cryptography is split into two main cryptosystems: Symmetric and Asymmetric cryptography.

2.1.1.1 Symmetric Cryptography

Symmetric cryptography, aka shared secret cryptography, relies in its operations on a private key that is shared between the communicating parties. In symmetric encryption, the shared secret is used for both encryption and decryption operations. Schemes for symmetric encryption are divided into two main categories according to how data is encrypted: Block ciphers

and Stream ciphers. Block ciphers encrypt a single fixed-size block of data at a time. Because block ciphers utilize deterministic algorithms, a plaintext block will always encrypt to the same ciphertext when the same key is used. Block ciphers can operate in several different modes, including CBC and CTR. The AES algorithm is an example of a well-known secure block cipher. A stream cipher is a symmetric key cipher in which each cleartext bit is XORed with a corresponding keystream bit one bit at a time. The output stream is generated based on an internal state that is hidden and changes while the encryption functions. An arbitrarily lengthy stream of secret key material is used to establish that internal state at first. The key is constantly changing by implementing a form of feedback mechanism. Stream ciphers, in general, operate faster than block ciphers and have less hardware complexity. RC4 is an example for a widely used stream cipher.

Integrity, as well as authenticity, in symmetric cryptography are achieved through Message Authentication Codes (MACs). MAC algorithms are realized through key-based hash functions, e.g. HMAC. Therefore, MAC algorithms inherit several properties from hash functions. For example, They accept messages of arbitrary size and output a MAC of a fixed size, relative to the algorithm used. A MAC is difficult to forge or to find a collision for. Moreover, a MAC is permanently tied to the specific message it is created for. The MAC is sent along with the message it verifies to the other party, which in turn recomputes the MAC of the received message using the shared secret and compares the output of its algorithm to the MAC it received. If both MACs match, then the message is authentic and tamper-free. Otherwise, the message should be discarded.

2.1.1.2 Asymmetric Cryptography

Asymmetric cryptography, aka public key cryptography, is established upon the use of two different keys, a public key and a private key. Although the keys are different, they are mathematically related. Nevertheless, the key pair are constructed in a fashion that it is infeasible to compute one from the other. The key pair generation relies on computational complexity of hard mathematical functions, such as the discrete logarithm problem. The most common practice is that a party creates its own key pair. The public key is published to public peers; however, the private key remains a secret to its owner. In an Alice and Bob notation, if Alice wants to encrypt a message to Bob, Alice uses Bob's public key for encryption. Bob can decrypt the received message using his private key. A famouse example for a public key encryption algorithm is RSA. Moreover, the key pair can be used in a key establishment protocol like DH.

To guarantee message integrity and authenticity, digital signatures are used to proof that a message is tamper-free. Digital signatures are analogous to MACs in symmetric cryptography. In contrast to encryption, when Alice wants to sign a message that will be sent to Bob, Alice uses its private key in the signature algorithm to produce a signature. The signature is attached to the message and sent to Bob. Bob uses Alice's public key to verify the signature of the received message. Two of the most used digital signature methods are RSA and DSA.

2.1.2 Digital Certificates

In the realm of public key cryptography, an End Entity (EE) in possession of a key pair is able to encrypt, decrypt and sign data to and from other entities. This satisfies the confidentiality and integrity requirements of communication security. However, this is not sufficient to achieve the authentication and authorization goals certificates, which are a requirement in several use cases. Authentication is the assurance that an entity is in fact who it claims to be, while authorization is the assurance that an entity has the privilege to communicate with a resource. Without authentication guarantees, a malicious third party can impersonate any other party in a communication channel or manipulate the messages between two communicating parties. Moreover, the intrusion cannot be detected. To address this problem, digital certificates were introduced.

A digital certificate (aka public key certificate) is an electronic document that cryptographically binds a public key to the identity of an EE. The certificate is cryptographically signed by a trusted third party that issues the certificate. In addition to including the public key and the digital signature, the certificate also contains information about the public key, the identity of its owner, and information about the certificate issuer. The party that wishes to authenticate itself presents its digital certificate to its counterpart. To validate a certificate, the receiving party has to trust the certificate issuer. If the issuer is trusted, then its public key is used to verify the certificate's signature and as a result trust the presented certificate.

To be able to use digital certificates in a scalable environment like the internet, a trusted large scale infrastructure that allows for the generation, storage, and distribution and revocation of certificates is required. Although not the only one, the X.509 standard [2] is the most commonly used for specifying digital certificates format and Public Key Infrastructure (PKI). It is an product of International Telecommunication Union (ITU). X.509 certificates are widely used in many internet protocols, including TLS. Therefore, X.509 is considered a pillar of network security. Currently, there are three versions of X.509, the most recent of which is simply known as X.509 version 3. In addition to the certificate identifiers and attributes, optional extensions have been added to the newest version, which can be tagged as critical and thus must be handled by the recipient, or can be ignored otherwise. In the X.509 PKI, a Certification Authority (CA) is an entity responsible for handling end entities request to generate a certificate for their public keys from the PKI. The requests are known as Certificate Signing Request (CSR). The CA verifies the identity of the EE and checks the attributes of the CSR subsequently decides whether to issue and sign a certificate to the requester or not. In the X.509 PKI, the CA is a trusted third party and the issued certificate is dubbed an *EE certificate*. Nevertheless, the CA needs to have a certificate to identify itself. In a PKI there are several CAs where they are organized in a hierarchical architecture. At the top level, there are the *Root CA* which have self-signed certificates as they are the highest authority. Root CAs issue certificates to CAs in the same level (cross certification) or in lower levels. Lower level CAs are named *intermediate CAs* which can issue certificates to end entities. Though end entities cannot use their certificates to issue other certificates. Another component of a PKI is the Registration Authority (RA). A CA can delegate some of its functionality to a RA. A RA lies between the CA and the EE, in addition, it is usually in proximity to the EE.

PKI provide methods for certificate revocation. Naturally, a certificate is only valid for a limited time due to various constrains. There are numerous critical reasons why its validity must be terminated sooner than allotted due to various threats, and hence the certificate must be revoked. For example, compromise of EE or a trusted CA private key. Wohlmacher

[3] provides a survey study of revocation methods that gives a good overview of the main revocation methods.

2.1.3 Bootstrapping

The term "Bootstrapping" is used in a spectrum of contexts including Computing, Law, Finance[1]. It is inspired from the late idiom "to pull oneself up by one's bootstraps" which means to succeed or elevate yourself without any outside help. In the context of Networking, bootstrapping is an initial procedure between pledges (unconfigured devices) which intend to communicate with a network for the first time. Its goal is to provide the device with the required information that enables it to establish subsequent secure communication channels with the desired network. Such information can be certificates, configurations, and metadata. More on bootstrapping in chapter 4.

2.1.4 Bootstrapping Voucher Artifact

While bootstrapping, the pledge must be able to authenticate the network attempting to take control of it, hence assigning ownership is critical for bootstrapping techniques. Defined in [5], a voucher is a signed document that identifies the cryptographic identity of the domain that a pledge should trust. The goal of the voucher is to allow for a authorize a zero-touch imprinting of the pledge on the registrar of a domain. The voucher is signed by a manufacturer's Manufacturer Authorized Signing Authority (MASA). The voucher artifact is a JSON formatted document which is included and signed in a CMS structure [6]. The artifact data model is formally described by a YANG [7] module in [5, Section 5.3]. Essentially, the main purpose of a voucher is to securely convey a certificate to the pledge with which it can authenticate the owner's domain registrar. The said certificate is found in the "pinned-domain-cert" attribute of the voucher.

Vouchers can be classified into two types: nonced and nonceless vouchers. Nonced vouchers contain the same nonce specified by the pledge in its submitted voucher request. Nonces provide mitigation against replay attacks as they are not reusable. On the other hand, Nonceless vouchers may be reusable. Their validity is dependent on their lifetime which may vary. A voucher's lifetime is specified by its "expires-on" attribute. Since voucher are non-revocable artifacts, the specification recommends short-lived vouchers rather than long-lived vouchers with the possibility to renew a voucher by reissuing it. The specification points out that the renewal process should be a lightweight process, as it ostensibly only updates the voucher's validity period. There may also exist nonceless vouchers without expiration times, however, they are advised against as they provide their bearer with extreme control over the pledge. Lastly, some pledges may not have the capability of understanding time, such pledges must not use vouchers with time constrains.

2.1.5 Certificate Enrollment

[1] *Bootstrapping*. URL: `https://en.wikipedia.org/wiki/Bootstrapping` (visited on 08/08/2021).

Certificate enrollment is the process where an EE requests and obtains a digital certificate from a PKI. The process starts by the EE submitting a CSR to the CA or RA of the PKI which the EE intends to obtain a digital certificate from. Several protocols [8, 9, 10, 11] were proposed to address certificate enrollment.

For certificate enrollment protocols, it is mandatory for EE to authenticate messages from entities. This is achieved by installing a root CA certificate on the EE that enables it to authenticate the PKI entities. Nevertheless, provisioning such certificate to an EE is out-of-band of certificate enrollment protocols, but it is achievable through bootstrapping protocols. On the other hand, the PKI may authenticate EEs using certificates they already posses or through private shared secrets.

Some protocols, like [10] require the EE to generate their own key pair locally, but some protocols, like [8, 9], provide the possibility to generate the key pair for the EE during certification and transfer the keys to the EE during the process. In either case, the EE has to provide proof of possession of the private key, possibly through digital signature, decryption, or challenge-response. Messages in certificate enrollment protocols have to be protected either by digital signatures or MACs.

Certificate enrollment protocols generally realize a set of basic functionalities. These functionalities include certificate enrollment, certificate update and renewal, and certificate revocation. However, how each protocol implements a functionality can differ in its features. For instance, [11] allows EEs and/or RAs to revoke EEs certificates, while [10] allows only CAs to revoke certificates. Different certificate enrollment protocols offer different functionalities. Some are only concerned with communication between EEs and CA/RA, while others offer intra-PKI entities communications. For example, [9] realizes cross-certification between CAs.

2.1.6 Protocol Formal Verification

Formal verification is the application of mathematical procedures to confirm that a design complies with some clearly specified notion of functional correctness. In the lack of formal mechanisms for verification, security flaws may go unnoticed. On the other hand, formal verification approaches give a method for analyzing complex protocol in detail and prove the absence of large classes of attacks in a systematic fashion. Automated verification tools are programs built around a specific approach of formal methods with the goal of providing security guarantees for protocol execution employing the underlying cryptographic primitives, cryptographic protocols, and network systems. They help uncover protocol situations or vulnerabilities that contradict the intuitive assumptions underlying the construction of a protocol since they can analyze a large number of scenarios based on rigorous mathematical notions.

Symbolic model checking is a protocol verification technique for determining if a protocol's finite-state model fits a set of requirements. Symbolic protocol analyzers effectively identify attacks, but they provide poorer security assurances than standard cryptographic proofs that account for probabilistic and computational concerns since they treat cryptographic constructions as perfect black boxes. OFMC [12] is a symbolic analyzer for security protocols. The AVISPA Intermediate Format IF [13] is OFMC's native input language. AnB [14] is an intuitive Alice-and-Bob-style language that OFMC also supports. OFMC automatically

converts AnB to IF. It performs protocol falsification and bounded session verification by exploring the transition system resulting from an IF specification. OFMC successfully employs two primary techniques: the lazy invader and constraint differentiation. The lazy invader is a symbolic representation of the Dolev-Yao intruder [15] that functions in a demand-driven manner. Constraint differentiation is a broad search-reduction approach that combines the lazy intruder with partial-order reduction principles.

2.1.7 Key Derivation Function (KDF)

A KDF is a cryptographic algorithm that generates keying material that cryptographic algorithms can use. The function requires two sorts of input: a secret value, such as a key or a password, and other data. The core of a KDF is often built using a pseudorandom function, such as Keyed cryptographic hash functions. A key derivation function iterates an n-bit pseudorandom function and concatenates the outputs until L bits of keying material are generated. Each output receives a distinct value with each cycle, thus if one key is compromised, the risk is isolated to that key while preceding keys remain secure. National Institute of Standards and Technology (NIST) proposed recommended techniques to use pseudorandom-based KDFs in [16].

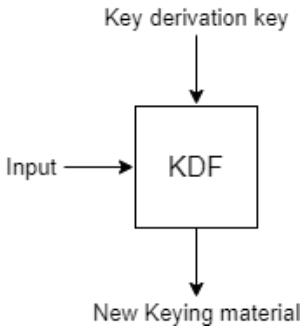

Figure 2.1: Key derivation function.

2.2 Related Work

2.2.1 Key Continuity Management

Peter Gutmann in [17] presented key continuity as mechanism for assuring that the entity with whom a user is currently communicating is the same as the one they were previously communicating with. Assuming the key resembles the identity of its user, the foundation for key management via key continuity is that once an entity has a remote entity's trustable key, it can validate later the remote entity's identity by confirming that they are still using the same key. The security mechanism relies on how long the key has been in service by its owner. The longer the key is witnessed to have been used, the more it is trusted by other entities. The research also proposed that an external authority manage the key continuity information. A key continuity external authority keeps track of how long a particular key has

been used by a specific cryptographic service and responds to inquiries with that information. The document discusses key continuity in several security protocol use cases.

However, the document does not precisely discuss authentication of keys at first use. Rather, the author has mentioned the possibility of users following the Trust on first use (TOFU) approach, or out-of-band means, for example, comparing key fingerprints over the phone. Moreover, He did not rule out utilizing digital certificates to verify public keys; nevertheless, he deemed it a more tedious, expensive, and manual process than TOFU.

2.2.2 Off-the-Record Protocol

Consider the following scenario: Alice and Bob are alone in a room. Unless they are recorded, no one can hear what they are talking to one other. Therefore, no one knows what they are talking about until Alice and Bob tell them, and no one, not even Alice and Bob, can verify that what they are saying is true. Off-the-Record (OTR) [18] aims to achieve the same form of secrecy in the field of instant messaging. The OTR protocol was first released in October 2004 by cryptographers Ian Goldberg and Nikita Borisov. At the time of writing, OTR is at version 3 [19]. OTR provides a set of essential security features: Encryption, Authentication, Deniability, and forward secrecy.

Abstractly, the protocol flow goes as follows. At first Alice and Bob establish a session encryption key through a DH key exchange. Even though each party has established a shared secret, neither has a guarantee about authentication, i.e., no party is sure that the other party is whom it claims to be as a man-in-the-middle attack is possible. Each party owns a long-term public key for identity authentication. These keys are utilized discretely between the communicating parties to prove their identity to each other without sacrificing deniability to third parties. Alice and Bob generate signatures using their long-term private keys, enveloped in messages encrypted using the computed session key. Moreover, HMAC signatures are generated for messages to guarantee integrity as well as authenticity. Alice and Bob generate symmetric signing keys by passing the shared encryption key through hash functions. These signing keys are used to generate HMAC signatures. Next, parties can verify the signature received and consequently have the assurance of the party's identity at the other end of the channel. Assuming there was a man-in-the-middle, he would not be able to forge a signature for either party, and thus, signature verification would fail. Despite the use of digital signatures, they are used within an encrypted channel, and no other party can verify that both parties were communicating. After a message has been received and successfully decrypted, the sender publishes the message signing keys, and both parties delete their encryption keys and start over with the DH key exchange to generate a new shared secret. Publishing signing keys enables outsiders to forge messages, enhancing the deniability feature for Alice and Bob. On the other hand, deletion of encryption keys ensures forward secrecy. Lastly, during the exchange of data messages, either Alice or Bob may employ the Socialist Millionaires' Protocol (SMP) [20] to detect impersonation or man-in-the-middle attacks.

2.2.3 SoK: Secure Messaging

Unger et al. [21] presented a comprehensive survey study where they used a methodology they've established to analyze and systematize current secure messaging solutions. Their

survey valuably contributes towards an open standard for secure messaging by combining the most promising secure messaging features. The study looks at secure messaging solutions from academic research as well as real-world deployments, which identify innovative and promising ways that have already been implemented but are not covered in academic literature. The presented framework focuses on evaluating three main principles of the surveyed solutions: trust establishment, conversation security, and transport privacy. For each, they evaluate the security, usability, and ease-of-adoption properties. However, security is the primary aspect relevant to our work.

Trust establishment is defined as the procedure through which users ensure that they are communicating with the intended parties, i.e., a combination of long-term key exchange and long-term key authentication. While, transport privacy is concerned with preserving the privacy of users by obscuring metadata of messages during transit, such as the sender, receiver, and which conversation the message belongs to. Nevertheless, the aspect most relevant to our work is conversation security. It relates to protecting messages' security and privacy; and comprises the methods used to encrypt communications, the associated data, and the cryptographic algorithms employed. In addition to confidentiality, authentication, and integrity, discussed earlier, below are additional security properties. The following properties are relevant to two-party communication only as group communication is out of our scope.

- *Participant Consistency:* When one honest party accepts a message, all other honest parties are assured to have the same view of the participant list.
- *Destination Validation:* When an honest party accepts a message, they may verify that they were included in the message's intended recipients list.
- *Anonymity Preserving:* The underlying transport privacy architecture's anonymity characteristics are not jeopardized by the message exchange protocol.
- *Speaker Consistency:* The order of messages transmitted by each participant is agreed upon by all participants. During the protocol, or after each message is transmitted, a protocol may execute consistency checks on blocks of messages.
- *Causality Preserving:* It is possible to prevent showing a message before messages that are causally related to it in the implementation.
- *Global Transcript:* All of the messages are displayed in the same order to all of the participants. It's worth noting that this presupposes speaker consistency.
- *Deniability:* In some secure communications protocol use cases, deniability is a desired property. It refers to the inability of others to verify that a particular individual transmitted the data. However, if Bob receives a message from Alice, he can be confident that Alice sent it, but he cannot prove it to anybody else. Anyone can forge messages after a conversation to make them look like they came from them. However, if it is during a conversation, participants can rest assured that the messages they exchange are authentic and have not been modified by an intruder. Secure messaging protocols that offer deniability can assure the user that anyone can forge messages on their behalf after a conversation has ended, but not during a conversation. Deniability may be realized by conversation security protocols in a variety of forms. The authors define the following deniability-related features.

  - *Message Unlinkability:* If a judge believes a participant authored one message in a conversation, that does not mean they authored all of the messages.
  - *Message Repudiation:* Under the assumption that the judge does not have access to the accused participant's long-term secret keys, provided a conversation transcript, and all cryptographic keys, including session keys, there is no proof that any individual user authored a given message.
  - *Participation Repudiation:* There is no proof that the honest participant was in a conversation with any of the other participants, given the conversation transcript and all cryptographic key material for all but one accused participant.

- *Forward* and *Future secrecy* are discussed later in chapter 5

Their study concluded a number of outcomes. First, the usability and adoption of trust establishment solutions with solid security and privacy guarantees are low. However, other hybrid approaches that have not been thoroughly examined in the academic literature may give better trade-offs in reality. Second, most of the stated conversation security properties are not mutually exclusive; nonetheless, combining protocol designs has considerable potential for improvement. For two-party conversation security, the most outstanding and promising solution of the bunch was the per-message ratcheting with resilience for out-of-order messages combined with deniable key exchange protocols, as implemented in Axolotl (now called the double ratchet algorithm), can be employed today at the cost of additional implementation complexity with no significant impact on user experience. Finally, transport privacy remains a challenging problem as it is difficult to solve without paying significant performance penalties. Among the analyzed solutions, no suggested approaches provided strong transport privacy properties against global adversaries while also remaining practical.

2.2.4 The TESLA Broadcast Authentication Protocol

Source authentication and integrity are among the most challenging aspects of protecting broadcast communication. In other words, receivers of broadcast data can verify that the received data originates from the claimed source and the data has not been modified en route. This difficulty is exacerbated by mutually untrustworthy receivers and unreliable communication settings in which the sender does not retransmit dropped packets. While many broadcast networks can effectively convey data to several recipients, they also make it easy for a malicious user to impersonate the sender and insert broadcast packets. This is known as a packet injection attack. Participants must use a broadcast authentication protocol to resist the attack, allowing receivers to verify that the alleged sender indeed transmitted the received packet. Nevertheless, using asymmetric cryptography to sign each message is a high overhead for time and bandwidth. Similarly, simply using MACs does not solve the problem as any receiver with the shared secret can impersonate the sender and forge data.

Perrig et al. [22] aimed to tackle the problem by proposing the TESLA protocol. TESLA inflicts low overhead for communication and computation while generating and verifying authentic information. Moreover, it is scalable to large numbers of receivers and tolerates packet loss. Nevertheless, TESLA requires the sender and the receivers to be at least loosely time-synchronized as well as the receiver or the sender is required to buffer some messages. Loosely time synchronization is that a receiver does not need to know the difference between

the sender and the receiver's time, however, it is only interested in an upper bound on the sender's time, or also know as the maximum time synchronization error. TESLA also relies on one-way chains which are repeated use of one-way hash functions. Similar in principle to KDF chains explained in section 2.1.7.

In principle, the sender in TESLA generates a chain from an initial random secret key s_ℓ by iteratively applying the one-way hash function till the desired chain length is achieved. All intermediate keys generated are stored securely as well as the final key s_0. The final key is referred to as the commitment key. The sender divides the timeline into intervals while assigning each key to an interval. A key is used to generate MACs for the messages sent during the specified time interval. However, keys are used in an opposite order to how they were generated, as depicted in figure 2.2. During an interval, the sender uses the key to generate MACs but key remains secret to the sender through out that interval. Meaning that receivers are unable to verify the MAC during that interval and also no malicious entity is able to forge a valid message. After the interval has passed, the sender publishes the now expired key for receiver to verify messages whose MAC was generated using that key. Receivers also verify the correctness of the disclosed key by inputting it into the one-way chain. If after a number of iterations the commitment key is reached s_0, then the key belongs to the expected chain and is valid.

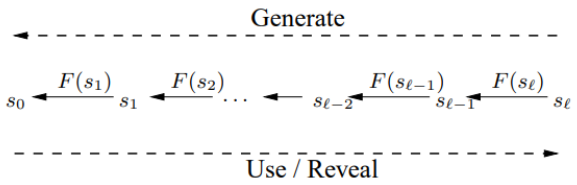

Figure 2.2: TESLA keys generation and usage [22].

Nevertheless, before authenticating messages with TESLA a receiver must be loosely time synchronized with the sender, know the disclosure schedule of keys, and receive an authenticated key of the one-way key chain. The required information can be authentically obtained via digitally signed broadcast message, or over unicast with each receiver.

Despite the security guarantees of the innovative approach, TESLA might not be suitable for some use cases due to the following limitations.

- There is an initialization phase in which the first element of the hash chain must be transmitted to all recipients. In this step, authentication is accomplished using standard asymmetric cryptography. Furthermore, a new chain must be formed once the hash chain is depleted.
- The verification of a message or a set of messages is only possible once the next is received.

2.2.5 A Survey of Key Bootstrapping Protocols

Malik et al. [23] provide a comprehensive survey study for a set of well-developed bootstrapping protocols in the context of IoT. They discuss the role of security and, in particular,

key bootstrapping protocols at different layers of the IoT architecture. Next, they suggest a taxonomy for classifying key bootstrapping protocols, as shown in figure 2.3. The considered protocols are classified under the mentioned taxonomy and analyzed with the focus primarily on bootstrapping protocols based on asymmetric key distribution schemes. Lastly, they compare the surveyed key management schemes based on security and functionality features: Mutual Authentication, Forward Secrecy, resilience to key compromise impersonation, and resilience to replay attacks.

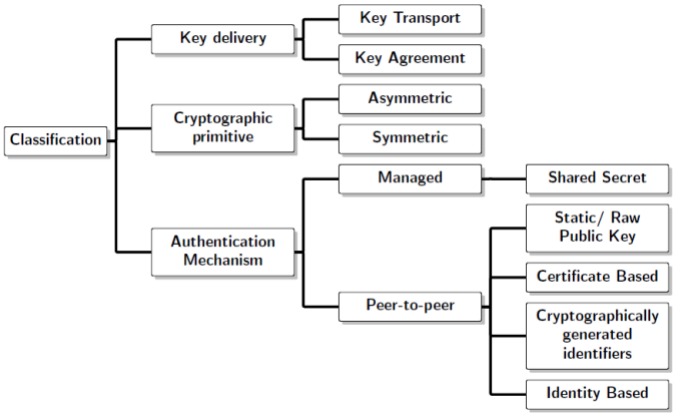

Figure 2.3: Taxonomy for classifying bootstrapping protocols [23].

3 Use Cases

3.1 Automated Border Control (ABC)

International passenger traffic has been monitored to continuously increase over the years. Since air transport is one of the most convenient form of long distance transport among passengers, the above claim is indicated by the Airports Council International's Passenger Traffic Summary[2]. Despite the current shock due to the Covid-19 pandemic to the air transport industry, the aviation industry has shown resilience over the past decades and is predicted to recover and further grow by 4% over the course of the next 20 years[3] international border crossing points are forecasted to face more traffic than they already are. Implying the requirement for a solution to the Border control process to mitigate longer queues and waiting times, or the need to hire and train more personnel to conduct the task.

Bio-metric passports are widely issued by more than 150 countries and regions, as of 2020[4]. Such passports contain a contactless smart chip that stores the passport holder's bio-metric information to authenticate his identity at passport control stations. The chips may contain one or more of the following bio-metric data: Iris, Fingerprint, and/or Facial features. In recent years, Automated Border Control (ABC) systems, or E-Gates, started rolling out in airports leveraging the wide deployment of bio-metric passports. They intend to automate the process and improve the management and control of travel flows, serving passengers in a shorter time and reduce queues at airports. Resulting in benefits to the Aircraft operators, Airports, passengers, and Governments [27].

Labati et al. [28] represented two types of communication performed at an E-Gate: communication between systems interconnected within the E-Gate, and communication with External systems. Internally, an E-Gate is composed of 4 subsystems: Document Authentication System (DAS), Biometric Verification System (BVS), Central Systems Interface (CSI), and Border Guard Maintenance System (BGMS). Externally, the E-gate is part of a larger border control infrastructure. It communicates with external systems to query for additional information to verify the traveler's eligibility to be granted access to cross the border. An E-Gate can query the databases of the following external systems: Visa Management System (VMS), Registered Traveler Program (RTP), and Entry-Exit Management System (EEMS). Figure 3.1 from [28] presents an illustrative overview of the E-Gate logical architecture. Essentially, the data transfer between the above systems, internally and externally, must be secure as

[2] *2017 Passenger Summary - Annual Traffic Data.* Jan. 2019. URL: https://aci.aero/data-centre/annual-traffic-data/passengers/2017-passenger-summary-annual-traffic-data/.

[3] Boeing. *Commercial Market Outlet 2020 – 2039.* Tech. rep. 2020. URL: https://www.boeing.com/commercial/market/commercial-market-outlook/.

[4] ReadID. *Which countries have ePassports?* 2020. URL: https://www.readid.com/blog/countries-epassports (visited on 08/07/2021).

it includes personal data, bio-metrics, and decision making information that affect a state's national security.

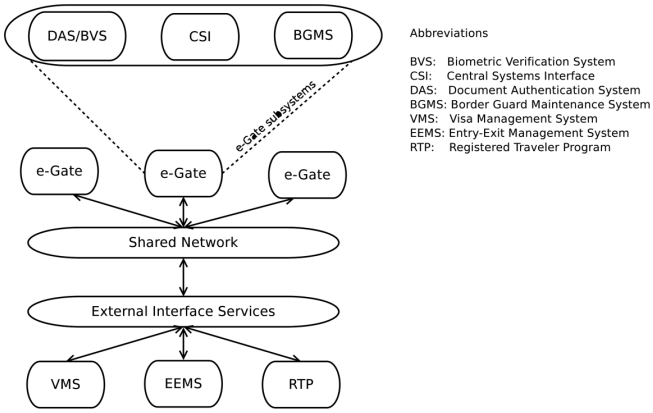

Figure 3.1: ABC Logical Architecture overview [28]

The workflow of the ABC systems is similar to the one depicted in Figure 3.2. At first, a citizen starts by scanning his passport data page through the gate's scanner. The scanner communicates the image to the Border control system (BCS) which in turn runs its optical data recognition software to extract the data from the scanned data page and verify the document security features. Next, the E-Gate proceeds by reading the embedded electronic chip in the passport. Due to the sensitivity of the bio-metric data, they are not transferred. However, the non-sensitive data stored on the chip, which is equivalent to the data already scanned, is sent to the BCS to cross-check the passport data page. The citizen's facial features and bio-metrics are scanned by the E-Gate to authenticate the citizen against the chip's data. Finally, If the BCS verifies the passport, the E-Gate verifies the bio-metric features, and no match was found in the BCS's watchlists, then the citizen is allowed to pass the border checkpoint.

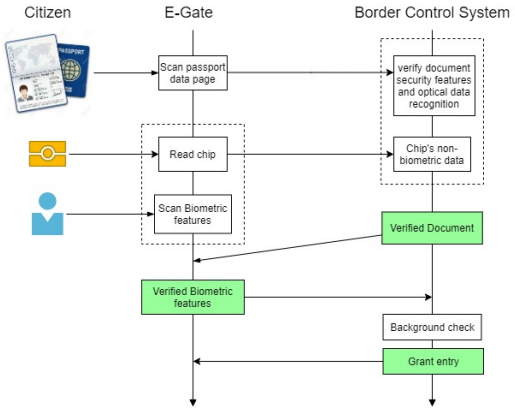

Figure 3.2: Workflow of ABC.

3.2 Industrial IoT

Industry 4.0, or the Fourth Industrial Revolution, refers to the 21st century's swift advances in technology, industry, and societal patterns and processes. Industry 4.0 is the concept of using automation and data exchange in manufacturing. Nine pillars of technologies make up Industry 4.0; among them are the Industrial Internet of Things (IIoT) and Cybersecurity. These technologies are used to create a "smart factory" where sensors, machines, systems, and humans communicate with each other in order to control and monitor progress throughout the manufacturing process digitally.

IIoT is a method of digitally transforming manufacturing. It relies on a network of sensors to gather essential production data, then leverages linked systems and infrastructure to transform that data into beneficial insights about the effectiveness of the industrial processes. Likewise, Cybersecurity is a vital foundation of IIoT. Industry 4.0 and IoT demand linking together independent devices and systems from possibly different vendors while one device's action is based on the output of another device. This translates to a substantial increase in connectivity directly related to an increase in the risk of potential cyber-attacks. IoT devices in the industrial realm interact and generate an enormous amount of data in different aspects. For example, employees' wearable devices to share information, alerts, and monitor their health status, smart production line machines, inventory environment sensors, and connected transportation vehicles. The data within IIoT environments is critically sensitive as it directly affects employees', customers', and manufacturers' privacy, manufacturer's operations and market competitiveness, and functional safety. A relevant incident that highlights the importance of cybersecurity in the industrial field is the infamous Stuxnet attack[5]. It is an attack against an Iranian uranium enrichment plant where the attack technically manipulated the enrichment centrifuges causing them to fail. Therefore, it is crucial for industrial information systems and manufacturing processes against cyber threats as a security breach can affect multiple areas, from supply chain to operations.

3.3 V2X Communication

- define v2v
- architectural overview of v2v
- use case messages
- secure communication (Tesal, etc...)

[5] https://www.wired.com/2014/11/countdown-to-zero-day-stuxnet/

4 Secure Bootstrapping

Integrity and confidentiality of information flow between the two ends of the communication are are what defines end to end secure channels. Authentication, whether unilateral or mutual, is another fundamental aspect to achieve channel security. Existing protocols, such as Transport Layer Security (TLS), can achieve End-to-end security between parties. However, analogous protocols rely digital certificates and credentials which are managed by local or third party PKI. Therefore to ensure correct and secure execution of protocols certificates must be securely provisioned to their corresponding identities. Typically at the manufacturing phase, the manufacturer usually installs globally unique manufacturer provided identifiers known as the Initial Device Identifier (IDevID) [29]. Its main use is for identity verification purposes and it should not be used to enforce data integrity nor confidentiality. Upon successful completion of the bootstrapping process, the device should posses identifiers that allows for subsequent establishment of secure channels with the network domain. An identifier in this set of identifiers is known as Locally Significant Device Identifier (LDevID) [29].

Bootstrapping approaches differ in the degree of manual user involvement and the amount of information on the device which must be pre-configured by the manufacturer. The survey by Sethi et al. [30] provides a classification for Bootstrapping mechanisms into general categories: Managed methods, Peer-to-Peer (P2P) and Ad-hoc methods, Opportunistic methods, and Hybrid methods.

- *Managed methods*: These bootstrapping approaches count on pre-established credentials and trust anchors for authentication and security. The required initial information and cryptographic material can be acquired either at manufacture time in the factory, or through Out-Of-Band (OOB) means, e.g. using a smart card or a USB. Examples of this method are Extensible Authentication Protocol TLS (EAP-TLS) [31] and Generic Bootstrapping Architecture (GBA) [32].
- *P2P methods*: Contrary to managed methods, these bootstrapping methods do not rely on any pre-established cryptographic material or information. Instead, the bootstrapping protocol results in credentials being established for subsequent secure communication. Typically, resulting credentials are authenticated using an OOB channel. This category of methods may be utilized if the manufacturer is incapable or untrustworthy to generate the desired credential.
- *Opportunistic methods*: Unlike previous methods where the authenticity of the initially presented identity is verified, approaches which fall under this category rely on verification of the continuity of the initial identity provided. Bergmann et al. [33] have developed a secure bootstrapping mechanism that is an example of this category.
- *Hybrid methods*: A wide range of deployed approaches use components from both managed and P2P methods. Such approaches are categorized as hybrid methods.

This chapter discusses two voucher-based bootstrapping protocols that aim at being zero touch protocols by eliminating the user interference. These protocols are BRSKI [34] and SZTP [35]. Both protocols fall under the managed methods category.

4.1 Bootstrapping Remote Secure Key Infrastructure (BRSKI)

BRSKI is a product of the ANIMA working group of the Internet Engineering Task Force (IETF). It is an automated bootstrapping protocol that enables an unconfigured device to discover and securely join an unfamiliar network domain it is installed in. BRSKI results in the device device acquiring an X.509 root certificate to authenticate the network domains' elements and establish consecutive secure channels. Moreover, the device can use the obtained certificate to perform further certificate enrollment protocols, like Enrollment over Secure Transport (EST) [8]. BRSKI is capable of realizing a large scale of thousands of devices in a risk prone environment. For example, customer devices provided by ISPs which are directly shipped to the customers.

4.1.1 Architecture Overview

The environment of BRSKI is composed of three general entities: the network domain, the pledge, and the manufacturer services. Figure 4.1 shows an overview of the protocols architecture.

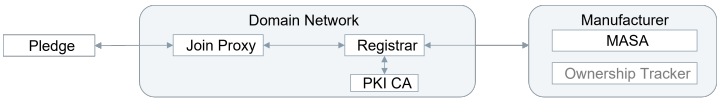

Figure 4.1: BRSKI Architecture.

The network domain is the domain of the alleged new owner of the device, i.e the network that is expecting the device to be connected to it. The domain is a network of entities who share a common local trust anchor. It incorporates a PKI to govern the issuance of digital certificates to provide unique digital identifiers for clients and establish end-to-end security.

A join proxy is another component of the domain. It helps with the discovery of pledges that intend to join the network. In addition, it is responsible for discovering the domain registrar(s) and determining the proxy mechanisms supported by the registrar and utilizing the lowest impact mechanism. Pledge discovery methods can be classified into passive and active methods. GRASP flooding [36] is a pledge passive discovery method for autonomic networks [37]. It is short for GeneRic Autonomic Signaling Protocol. It is used for signaling between autonomic service agents. GRASP provides discovery, flooding, synchronization, and negotiation functionalities for the technical objectives through respective GRASP messages. Pledge discovery via GRASP multicast flooding is the normative and mandatory method for BRSKI. On the other hand, DNS-based Service Discovery [38] over Multicast DNS [39] as well as DHCP [40] are pledge active discovery methods. They can be used as secondary discovery methods in parallel to GRASP. Moreover, A proxy provides HTTPS connectivity

and forwards messages without examination between a pledge and a registrar in the network, and without interfering with the protocol messages.

A pledge is an unconfigured device attempting to join the network domain. Its goal is to be securely bootstrapped in a zero-touch fashion. To achieve this goal, the pledge establishes a TLS connection with one or more of the domain's registrars through the domain's proxy. It is necessary for the pledge and the registrar to establish mutual authentication. A manufacturer installed IDevID is used for pledge authentication to the domain's registrar. It is installed during the manufacturing process and includes certificates signed by the manufacturer and unique identifiers that represent the pledge, in addition to, the pledge unique serial number given by the manufacturer. It is recommended that The provided certificates are used for authentication with the registrar and the signing of voucher requests. The unique serial number is used in vouchers and voucher requests to ensure linkability.

A registrar is an element of the domain that is responsible to carry out the bootstrap process for the pledge. Also, it can be considered as the RA for the domain's PKI. A domain can have one or more registrars which all have to be recognized by the domain proxy. If the pledge is capable to concurrently connect to multiple registrars, it is advisable to do so as this protects against a malicious proxy attempting a Denial of Service (DoS) attack like Slowloris.

A manufacturer is the entity that produced the device and set up its initial configuration (IDevID). It provides two distinct services: the MASA service and ownership tracking and validation. MASA can be a third party service that signs the vouchers issued for the bootstrapping process. It is also responsible for providing a repository for audit-log information of bootstrapping events. The service is contacted each time a pledge performs a zero-touch bootstrap in an attempt to enroll into a domain. It takes the decision whether or not issue the voucher according to the MASA policy. Voucher issuance could be done blindly at the lowest security level or it could be tightly bound to the sales channel that verifies the actual ownership of the domain. Hence, the manufacturer can provide protection against stolen devices or illegitimate resale of devices by declining voucher issuance to the suspected pledge.

Ownership tracking and validation is an optional manufacturer service. It is supposed to log all claim attempts and to know which device is owned by which domain and provide such information to registrars. A verified log entry indicates that the pledge was issued a voucher as a result of positive verification of ownership.

4.1.2 Protocol Details

This section describes the message sequence of BRSKI illustrated in figure 4.2 and elaborates on content of the exchanged messages. The numbering sequence referenced through this section refers to the message numbers in figure 4.2.

- *Message 1a, 1b (Discovery phase):* It is the first phase of the protocol where the pledge identifies the domain proxy. This can be performed by a pledge initiated mechanism as in message (1a) or via a proxy initiated mechanism as in message (1b). After successful discovery, the pledge can address a domain registrar through the proxy. A proxy does not assume any specific TLS version.

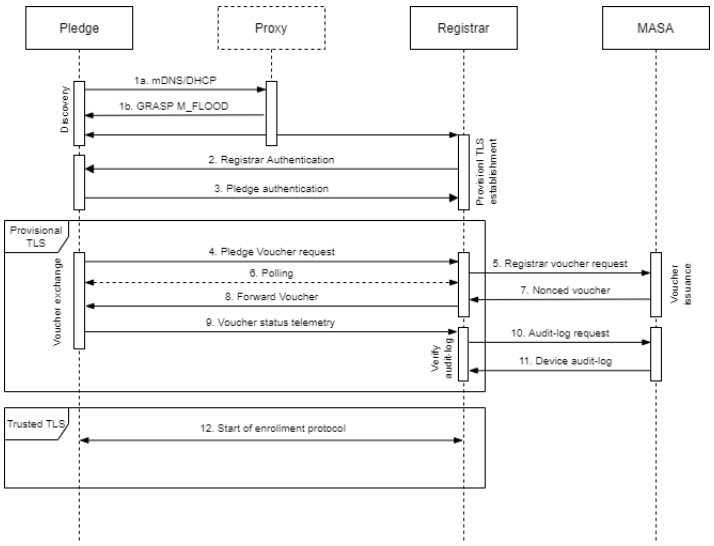

Figure 4.2: A successful BRSKI protocol run.

- *Message 2, 3 (Provisional TLS establishment):* The pledge attempts to establish a TLS channel with each discovered registrar to ensure End-to-End security. The pledge must not use any TLS version lower than TLS 1.2, while TLS 1.3 is the encouraged version to be used. To establish the channel, mutual authentication has to be performed. At first, in message (2), the pledge receives the registrar's server certificate. However, the pledge does not posses any trust anchors to verify it yet. Therefore, the pledge accepts the registrar's certificate provisionally. Next in message (3), the pledge is authenticated via the installed IDevID. The registrar must be able to verify the provided certificate, however the distribution of the trust anchors for this task is out-of-scope of BRSKI. Meaning, the information received by the pledge must be untrusted, although it is in a TLS channel, till a trusted trust anchor to verify the certificate is received.
- *Message 4:* Having established a secure provisional TLS channel, the pledge initiates the voucher exchange by sending a pledge voucher request to the registrar in message (4). The request must contain a unique nonce per bootstrapping attempt to protect against replay attacks. Also, the request contains the 'proximity-registrar-cert' and the pledge serial number. The 'proximity-registrar-cert' is the EE certificate of the registrar which is also used to establish the provisional TLS channel. Pledge serial number is a manufacturer defined unique identifier for each device. It is different from the IDevID certificate serial number. Since not all devices have a real-time clock, depending on the device capabilities, the request is recommended to have the 'created-on' value. Finally, the request must be signed using the pledge's IDevID certificate.

  The registrar authorizes the pledge based on the authenticated information presented in the pledge's IDevID and the registrar's policy. The policy can be either to allow any device from a specific vendor, to allow any device of a specific type, or to allow a specific type of devices from a specific vendor.
- *BRSKI-MASA TLS channel:* The registrar initiates a TLS 1.2 or newer channel with the MASA where all subsequent communication between the two parties occur within

this secure channel. The MASA URL is obtained from the pledge IDevID, as mentioned earlier. To authenticate the MASA, the registrar should be configurable with trust anchors on a per vendor MASA basis as part of the sales process. Moreover, the registrar should also support client authentication mechanisms such as TLS client certificate, HTTP Basic, Digest, or Salted Challenge Response Authentication Mechanism (SCRAM); however TLS Client Certificate based authentication is the recommended method.

- *Message 5:* After obtaining the pledge's voucher request, the registrar constructs a registrar voucher request that is sent to the MASA to obtain a voucher for the pledge. The registrar voucher request is a JSON document that is signed using a CMS structure. The JSON document encapsulates the pledge voucher request CMS object that was sent to the registrar and is referred to as 'prior-signed-voucher-request'. Moreover, the request contains the 'created-on' field which holds the timestamp the request was formed on. In addition, it consists of other fields which relate to the pledge request like the pledge serial number, the nonce used produced by the pledge and used in the pledge request, and 'idevid-issuer' field which holds the issuer value of the pledge IDevID certificate. The registrar includes some certificates in the registrar voucher request CMS object as well. Those certificates are used by the MASA to be pinned into the voucher to be later used by the pledge as a trust anchor for authenticating the domain registrar. Therefore, the certificates enclosed by the registrar in the request have to be part of the chain it wishes the MASA to pin in the voucher. Hence the specificity of the attached certificates is considerably significant. A 'pinned-domain-cert' can be as specific as the registrar's TLS EE certificate. On the other hand, if it is as general as a public webPKI CA it could permit any entity that possess a certificate issued by that authority to claim ownership of the device.

  On the other hand, the pledge might not be available at the time of deployment to send a pledge voucher request, or the registrar speculates to not being able to reach the MASA at the time of deployment where the pledge will be available. Such use cases justify the need for nonceless registrar voucher request. In these cases, the previous message (4) would not exist. To formulate this request, the registrar has to acquire the pledge's serial number and IDevID issuer, however, they are obtained through out-of-band means. Subsequently, the nonce field of the request is omitted.

- *Phase 6 (Polling)*: Before processing the pledge's request, the registrar may send the pledge an HTTP 202 response message which indicates that the request received earlier has been accepted for processing however processing is not yet complete. This response initiates a polling phase between the pledge and the registrar. A "Retry-After" field is specified within the headers of the registrar's response that indicates the minimum time for the pledge to wait before asking for a response for the voucher request sent earlier. After the specified waiting time, the pledge polls the response by resending the exact same request and must not change the nonce nor sign a new voucher request. If the pledge is simultaneously trying to bootstrap itself with several registrars of the network, it can be overwhelming for the pledge to keep track of all the "Retry-After" times. Therefore, a pledge may ignore the specified interval and follow a hard-coded "Retry-After" interval. A pledge should be able to hold the retry state for a maximum of 4 days.

- *Message 7:* upon receiving the voucher request, the MASA performs a set of checks to decide weather to issue the requested voucher. Given the fact that vouchers have a short lifetime, the request may be from a registrar that has been issued a voucher previously, i.e a voucher renewal request. In this case, the request should be automatically authorized by the MASA.

  The MASA extracts the certificate chain attached in the signed CMS object. If the domain CA is unknown to the MASA it is considered as a temporary trust anchor as the intention is not to authenticate the message rather to establish consistency of the domain PKI. According to the MASA's policy, it decides which certificate of the chain supplied by the registrar it chooses to pin. It may be the farthest certificate of the chain, or it may be as close as the EE TLS certificate of the registrar. If revocation information is available for that certificate, it must be checked by the MASA to prevent issuance of new or renewed vouchers to unauthorized registrars. Next, the CMS signature is validated using the domain's CA extracted from the voucher request. Also, the signing certificate is verified to contain the 'id-kp-cmcRA' Extended Key Usage. This ensures that the signer is an entity that is authorized to be a registrar of the domain. Hence, assures domains that a MASA only accepts requests from domain registrars.

  In case of nonceless requests, It is mandatory for the MASA to authenticate the registrar. The decision to issue a noncless voucher is taken according to the MASA policy that is out of scope.

  In case of nonced voucher requests, the MASA verifies that the 'prior-signed-voucher-request', enclosed in the registrar request, contains a 'proximity-registrar-cert' that is coherent to the certificate used to sign the registrar voucher request. Moreover, the nonce is verified to be consistent between the registrar voucher request and the 'prior-signed-voucher-request'.

  Subsequent to a successful validation of the request, the MASA responds with an issued voucher in message (6). Any issued voucher by the MASA is recorded in the audit-log. Otherwise if a problem occurs, a response with the appropriate http signaling as described in [34]. For example, a 403 status code response if the voucher request is not signed correctly, or a 406 status code response if the requested voucher type or algorithms cannot be issued due to the MASA's awareness that such pledge is not capable of processing them.

- *Message 8:* The registrar evaluates the received voucher solely for transparency and future audit-log verification. The received voucher is forwarded to the pledge without any interference or modification from the registrar.

- *Message 9:* After the pledge successfully receives a voucher, the pledge must indicate its status regarding the voucher to the domain. This occurs by sending a status message to the registrar. The pledge decides weather to accept the voucher or not through the voucher validation process. If acceptable, the message should contain the version of BRSKI and a boolean status field to indicate the acceptance status. In case of an unacceptable voucher or a failure, the pledge is expected to fail gracefully. The message should contain a Reason field with a string commenting on the cause. Nevertheless, the Reason should not be excessively descriptive as it may be sent to an unauthenticated and potentially malicious registrar.

Bearing the voucher, the pledge verifies its validity. It verifies the signature using the manufacturer installed MASA trust anchor. It verifies also that the serial number enclosed in the voucher matches its own. For nonced vouchers, the pledge verifies the voucher nonce corresponds to the nonce it sent earlier in the voucher request. However nonceless vouchers can be accepted according to pledge local policy. The pledge can be configured to always accept nonceless vouchers to realize the use case where the MASA is unreachable at the time of pledge deployment.

A pledge could be operating in other similar security reduced mode that skip voucher validation in favor of offline or emergency touch-based deployment bootstrapping procedures. For example, TOFU or physical presence methods such as the use of serial console or depressing a physical button during bootstrapping. However, TOFU must not be available unless a hardware-assisted Network Endpoint Assessment (NEA) is supported. Meanwhile, it is only recommended for other methods of skipping voucher validation. This recommendation serves as a prevention against unintended use of offline methods when autonomic methods fail or are unavailable.

Upon successful verification of the voucher, the voucher's pinned-domain-cert should be considered by the pledge as a trust anchor. The current provisional TLS connection between the pledge and the registrar is evaluated using the obtained trust anchor. The pledge verifies the registrar's TLS server certificate using the trust anchor's public key. If the registrar's credentials could be verified, either by directly matching the server certificate or through verifying a higher certificate in its chain, the pledge trust the TLS connection and it is not considered provisional any further.

- *Message 10:* After receiving the pledge status telemetry message, the registrar requests the MASA audit-log from the MASA. The log data helps the registrar make a knowledgeable decision regarding further proceeding of the bootstrapping process. The decision making criteria is based upon the security requirements of the registrar domain. Hence, the criteria is out of the protocol's scope. The request content is the exact same registrar voucher-request sent earlier to the MASA, but is directed through a different URI specific for requesting the audit log, which is "/.well-known/brski/requestauditlog". Reusing the same message minimizes the required cryptographic and message operations on both ends. The registrar may reuse the cached voucher request and the MASA may take advantage of its internal state to correlate the message with the already verified request averting additional operations.
- *Message 11:* A MASA can infer the proper pledge log to be prepared from the "idevid-issuer" and the "serial-number" information included in the received request of the previous message. Instead of immediately responding with the audit-log, the MASA can a HTTP 201 "Created" response with a URL in the "Location" header field redirecting to actual audit-log. The response log is a JSON format document consisting of all the log entries associated with the pledge. Nevertheless, a MASA that sends out URLS has to ensure they are unpredictable to avoid enumeration attacks against device audit-logs.

  The log format structure consists of several entries: "version", "events", and "truncation". "version" is an integer value representing the log format version. "events" is an array of event objects that are associated to the device. Each of the event objects is comprised of a set of entries. The "date" entry represents the event's timestamp in the

format according to [41]. The “domainID” is a unique identifier for the domain’s registrar that encodes the pinned-domain-cert’s SubjectKeyIdentifier or SPKI fingerprint in base64. A “nonce”, if exists, is a base64 encoding of the same nonce used in the voucher request and issued voucher. If it is a nonceless voucher, then the field should preferably be set to null rather than omitting it. The “assertion” field indicates the level of verification with which the MASA issued the voucher. It can have one of three values: “verified”, “logged”, and “proximity”; the latter being the one supported by this protocol. “truncated” field shows the number of event truncations for the specified domainID. Lastly, since audit-logs can be arbitrarily large, duplicated or old entries may be truncated as an optimization for the log structure. The “truncation” entry contains meta-information about truncated entries such as “nonced duplicates”, “nonced duplicates”, and “arbitrary”.

On the registrar side, the received audit-log is vetted for discrepancies and unexpected behavior like the pledge previously imprinting to an unexpected domain or whether a certain domain possesses a nonceless voucher and can reset the device anytime. If the registrar’s audit-log verification is successful, then the bootstrapping process is complete.

- *Message 12:* At this point, the pledge has a trust anchor allowing it to verify the registrar, as well as a trusted TLS channel between them. Therefore the environment is suitable to start a certificate enrollment protocol after which the pledge obtains digital certificates that authenticate it to the domain and authorize it to utilize the relevant domain services.

  BRSKI is described as an extension for EST that provides automated proposal instead of the originally manual authentication method that relies on the intervention of a human user. Hence it is recommended for a pledge to use EST following BRSKI as a certificate enrollment protocol as it is considered a harmonious integration.

  Nonetheless, the succeeding certificate enrollment protocol is not limited to EST, however, a variety of certificate enrollment protocols can be used. Using EST is an example of a pull model where the EST server is the protocol initiating party. As an example of the push model architecture, Certificate Managment Protocol (CMP) can be used as the certificate enrollment protocol since the EE is the initiating party of the protocol.

4.2 Secure Zero Touch Provisioning (SZTP)

SZTP is as well an outcome of the IETF group. It introduces a bootstrapping technique for devices in initial factory state. It aims at securely supplying networking devices with bootstrapping data without further actions required other than physical placement and connecting the power and network cables. SZTP is primarily concerned with physical devices, however, it has the potential to be extended to logical entities like virtual machines. The protocol is not limited to provisioning trust anchors to the device but rather can update the device’s boot, commit an initial configuration, and execute arbitrary scripts to handle auxiliary requirements.

4.2.1 Architecture Overview

The protocol's architecture is divided into three abstract entities: the owner, the manufacturer, and the device. Figure 4.3 illustrates the protocol's architecture.

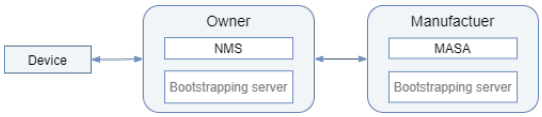

Figure 4.3: SZTP Architecture.

The owner is used to refer to the person or organization that owns the device. Throughout the protocol, the owner term abstractly represents the owner network domain and its sub-entities without explicitly referring to them. An example of the owner's sub-entities are the domain CA and the domain registrar. An owner possesses an *Owner Certificate* which is an X.509 certificate that identifies the owner's identity and binds it to its public key. In addition to validating the owner's identity, the certificate is used to by other entities to validate the owner's digital signature over received artifacts. A Network Managment System (NMS) is assumed to be part of every owner's structure. A NMS is a software used by network administrators to monitor software and hardware nodes in a network and logs data from those nodes for reporting. The bootstrapping process introduces newly admitted devices to the NMS.

The manufacturer is the entity that produced the device. The term is also used to refer to entities that the manufacturer delegates functions to. A MASA is an example of a third party manufacturer delegate entity. Each manufacturer is supposed to operate a MASA or most commonly delegate its functionality to a third party. The MASA is responsible for generating the voucher required by the device to authenticate the owner and bootstrap the device. Each of the manufacturer services and its related entities has its own certificate for authentication and signing. The manufacturer preserves a list of trusted voucher-signing authorities and well-known bootstrap servers. The shipped devices from the manufacturer are pre-configured with the list of trust anchors along with their own IDevID credentials. Owners are assumed to trust the same pre-configured trust anchors as the ones maintained by the manufacturer.

The protocol defines three artifacts that are exchanged through out the protocol: Conveyed information, ownership voucher, and owner certificate. Those artifacts provide the device with all the required information needed for bootstrapping. The conveyed information is divided between redirect information and onboarding information. Bootstrap servers are Representational State Transfer Configuration (RESTCONF) servers that provide bootstrapping artifacts to devices. Depending on the type of information a server provides, they could split into two types of servers: Redirect servers and Onboarding servers. Redirect servers only returns redirect information to clients while onboarding servers only return onboarding information. However, servers are not limited to mentioned categories only. There may exist a server that can provide both types on conveyed information.

Redirect information redirects a device to another bootstrapping server. Redirect information encodes a list of bootstrap servers hostnames and an optional trust anchor certificate that the device can use to authenticate each bootstrap server with. Depending on the source,

redirect information may be trusted or untrusted. It is trusted whenever obtained via a secure connection to a trusted bootstrap server or whenever it is signed by the device's owners. Trusted redirect information is useful for enabling a device to establish a secure connection to a specified bootstrap server. Untrusted redirect information is useful for directing a device to a bootstrap server where signed data has been staged for it to obtain. Redirection acts as a guide for device to discover the bootstrapping servers capable of providing onboarding information.

Onboarding information is a bundle of data required for a device to perform the bootstrapping process into the owner's network. It includes information about the boot image a device is required to be running, an initial configuration a device must apply, and scripts to address arbitrary needs which the device must successfully execute. Onboarding information must be obtained from a trusted source, either through a secure connection to a trusted bootstrap source or the information obtained is signed by the device's owner.

Devices can obtain the needed bootstrapping data from various sources according to the owner's deployed infrastructure. However, The concern is focused on sources that can supply devices with the information in an automated manner. Here are examples of touchless bootstrapping sources.

- A DNS server can be utilized as a form of a bootstrapping server. It is an attractive method for environments which already employ a DNS infrastructure. It does necessarily require communication with an internet-accessible third party DNS service. Using DNS as bootstrapping server is considered a touchless bootstrapping as it does not involve any external interaction. However, a DNS server is not a trusted source of bootstrapping information, even if DNSSEC [42] is used to authenticate the DNS records. The reason being that the device cannot verify if the returned domain belongs to its rightful owner. Therefore, the returned DNS records must either be signed for later verification by the device or the records must be processed provisionally.

  Devices supporting DNS server as a bootstrap server are offered two prioritized types of queries to the server: device-specific and device independent queries. Queries must first be attempted using multicast DNS before unicast DNS. A DNS response for the device-specific query can encode the three artifacts into the TXT records but the response must be signed. However, by DNS conventions, the size of signed data is large. Therefore, it is implausible for the signed onboarding information to fit the UDP-based DNS packet size. Thus, if signed onboarding is to be sent over DNS, it is expected to be transported over Transmission Control Protocol (TCP) so as to handle the DNS TXT record size. Otherwise, only redirect information shall be returned. On the other hand, DNS response for device-independent queries does not support returning onboarding information because the DNS server is incapable of returning signed data in response to the device-independent query. Accordingly, only redirect information shall be returned.

- Similar to DNS servers, a DHCP server is another untrusted bootstrapping data source. It can be leveraged by deployments that already employ a DHCP infrastructure. However, a DHCP server is a limited bootstrapping source due to its lack of ability to transmit enough data to hold signed bootstrapping data. Nevertheless, a DHCP server can return to a device unsigned redirect information.

- Lastly, a specific bootstrap server can be deployed to provide bootstrapping data and

receive data from devices. It is defined as a RESTCONF server that implements the YANG module described in [35, Section 7]. Moreover, it may be using TLS which may eliminate the requirement to sign the transmitted bootstrapping data if the bootstrap server is trusted. Otherwise, if the bootstrap server is not trusted by the device, the conveyed information must be signed or processed provisionally. Whether a bootstrap server is trusted or not depends on the device's knowledge of the server's trust anchor. A bootstrap server exposes two endpoints to communicate with devices. Namely, the "get-bootstrapping-data" for devices to request bootstrapping data and the "report-progress" for devices to report their bootstrapping process status back to the bootstrap server, such as warnings, errors, and the result of the process. A bootstrap server may be hosted by the owner or is an internet accessible server hosted by the manufacturer.

4.2.2 Protocol Details

SZTP is focused on the enrollment process between the device and its owner. However, before initiating SZTP, the owner and the manufacturer must complete a pre-protocol phase where the owner orders the devices and enrolls itself to the manufacturer where they exchange vital information required for the owner to be able to run SZTP. This phase can be referred to as the Device Ordering and Owner Enrollment phase. At first, the owner orders the needed devices from the manufacturer. In turn, the manufacturer always provides the owner with the trust anchor it will need to verify the IDevID certificates the devices use. If the manufacturer offers an internet-based bootstrap server, the manufacturer may also provide the owner with the necessary credentials to access the hosted bootstrap server to configure it. Consequently, the owner configures its NMS with the information obtained from manufacturer. If a remote bootstrap server credentials were obtained, the NMS configures an owner trust anchor certificate onto the bootstrap server that is to be used as the 'pinned-domain-cert' of the voucher at the time of dynamically generating the enrollment voucher. After the manufacturer concludes the order and the devices are shipped, it may send the owner the serial numbers of the devices, other meta information, or even nonceless ownership vouchers. The owner may configure its NMS with the received information to prepare its network infrastructure to enroll the expected devices.

Eventually, the owner is in possession of the devices and physically sets them up in its environment. As soon as the unconfigured device is booted for the first time, it checks if its factory default configuration enables SZTP bootstrapping. If enabled, the device initiates the SZTP protocol and goes through a series of phases to bootstrap into the owner network infrastructure. Figure 4.4 mentions the said phases which are discussed below.

Figure 4.4: SZTP protocol phases.

a) *Discovery:* First of all, the device discovers and enumerates all the available sources of bootstrapping information. A legitimate discoverable source of bootstrapping data

can be one of those discussed in section 4.2.1. For each source supported by the device, the device attempts to obtain bootstrapping data from it. The Bootstrap sources approached in order of their close proximity to the device. A device only process one bootstrapping server at a time.

b) *Authentication:* Secondly, when contacting a bootstrap server, it is mandatory for the device to authenticate itself to the server. The device authenticates itself using its TLS client certificate which is the same as its IDevID certificate. The server validates the client TLS certificate through the manufacturer trust anchor obtain earlier. Moreover, the device may possess more trust anchor certificates other than its TLS client certificate. If the device supports connecting to remote well-known servers, then it must authenticate the server through a pre-configured list of trust anchors for well-known servers. In addition, the device requires trust anchor certificates from the manufacturer for validating the ownership voucher. Devices should have the certificates chain of trust anchor certificates up to and including the self-signed root certificate.

c) *Bootstrap data transmission & processing:* After establishing a connection to the bootstrap server, whether trusted or not, the device starts receiving artifacts to bootstrap itself. The type data acceptable from bootstrapping sources depends on their trust state from the devices point of view. Untrusted sources can provide signed or unsigned redirect information, bearing in mind that the server which the device was redirected to must provide the device with further redirect information. Contradictory to redirect information, untrusted sources can only provide signed onboarding information. On the other hand, trusted sources can provide both redirect and onboarding information regardless signed or not. Since trusted sources already establish an authentic and secure channel with devices, it is redundant to have the transmitted information signed, therefore unnecessary. Naturally, the ownership voucher and the owner certificate are signed in any case and relevant revocation information may be additionally included in case it cannot be obtained dynamically. If the device received bootstrapping information that does not obey the restrictions respective to its source's trust status, the device should discard the information and quit the bootstrapping process with the said source.

   Whenever a device receives signed data, it must validate the data signature before processing it regardless of the data type. Signed data must always be accompanied by an owner voucher and an owner certificate. To validate the signed data the device must first validate the owner voucher and certificate. The device begins by validating the voucher's signature using its pre-configured trust anchors. It also verifies that the voucher's creation timestamp was in the past if the device is equipped with an accurate clock, and that the voucher "assertion" value is acceptable by the device. In addition, the serial number of the device has to match that of the voucher as well as the "idevid-issuer", if present. If valid, the device extracts the now trusted "pinned-domain-cert" certificate from the voucher. Next, the device authenticates the owner certificate by verifying its certificate path to the extracted "pinned-domain-cert", and if specified in the voucher, the revocation status of the certificate chain used to sign the owner certificate has to be checked. Finally, if the owner certificate is validated and the conveyed information was truly signed by that certificate, only then can the signed redirect or onboarding information be processed.

   Redirect information provides a device with a list of bootstrapping servers. The device

proceeds through the list till it reaches a bootstrap server it can bootstrap from. If a server provides the device with bootstrapping data, the device must attempt to process it before proceeding to the next bootstrap server. A bootstrap server may further redirect the device to other bootstrap servers. In this case, implementations must limit the number of recursive redirection to a maximum of ten redirects to avoid indefinite redirections. Trusted redirect information may be accompanied by a trust anchor certificate. Hence, the device must authenticate the redirected to server certificate against the certificate obtained from the preceding bootstrap server. If no trust anchor is provided or the redirect information is untrusted, the device must process the information provisionally.

Onboarding information is parsed by the device to extract the required instructions to be executed to evidently bootstrap the device. Onboarding information defines the boot image a device must run, pre-configuration scripts, initial configuration to be committed, and post-configuration scripts. The device must process the received information in that order. The boot image is verified if it complies with the required boot image. If the requirement is not satisfied, the device must download the boot image from the URIs defined in the onboarding information. After verifying the installed image, the device must reboot which will lead it to restart the bootstrapping process but with the new boot image, but then the device will not halt at this step and will proceed to further steps. If pre-configuration scripts are specified, the device must execute them and log their outputs. If an initial configuration is specified, the device must apply it according to the approach stated in the "configuration-handling" node. Similar to pre-configuration scripts, post-configuration scripts must be executed and their output must be logged.

Whether a device should return progress reports of the bootstrapping process is decided according to the trust state of the bootstrapping server. A device must not send progress reports to untrusted sources. However if the source is trusted, the source devices the minimum verbosity level it requires from a device in the "reporting-level" node of the "get-bootstrapping-data" RPC-reply. Moreover, the device may send more reports than originally specified by the trusted source. If an error occurs during any of the stages, devices must roll back the current step and previous steps and exit the bootstrapping process. However, results of some steps of the bootstrapping process may be retained, like an updated boot image. When eligible, devices must send error reports whenever they occur at any of the onboarding information processing stages. The bootstrapping process with current source should be quitted and the device should start the process over with the next bootstrapping source, if available.

4.3 Protocols Comparison

BRSKI and SZTP are two different protocols which tackle the same problem. Their main goal is to automatically and securely provision trust anchors to devices without any manual interference at the device. Hence, allowing such unconfigured devices to authenticate the identity of their new owners and establish secure channels for further processes like certificate enrollment. This section compares and contrasts both protocols. In addition, Appendix A provides a comparison between the two protocols and their terminology in a tabular form.

As mentioned, BRSKI and SZTP share a similar aim. BRSKI aims at storing a root certificate on the pledge that is sufficient for verifying the owner's domain registrar identity. While SZTP is more comprehensive as it involves updating the pledge boot image, commit an initial configuration, and execute arbitrary scripts. The fundamental architecture for both protocols is the same as both function on three elemental entities: the pledge, an owner representative known as registrar, and the manufacturer's MASA. However, unlike BRSKI, the communication between the owner's registrar and the MASA is not intrinsic to the protocol. BRSKI is integrated with EST as an enrollment protocol to be utilized after bootstrapping, but it is not limited to it. On the other hand, SZTP is not natively integrated with an enrollment protocol. Both protocols use HTTP on top of TLS as their transport protocol, but BRSKI specifically restricts the use of TLS to version 1.2 or higher. In addition, BRSKI proposes the usability of CoAP as an alternative for HTTP. Both protocols do not require any specific cryptographic algorithm.

The bootstrapping data differ between both protocols. For SZTP, the bootstrapping data are composed of the types of conveyed information explained earlier, the ownership voucher, and owner certificate. However, for BRSKI, it is only the ownership voucher artifact. Therefore, both protocols are reliant on the voucher artifact to provide the trust anchor necessary to verify the owner certificate. The owner requests the voucher similarly in both protocols. Nonceless vouchers are provided to owners out-of-band, while nonced vouchers are requested dynamically during a protocol run. In general, bootstrapping data are protected. In BRSKI and SZTP's untrusted channels the data must be signed, but for SZTP's trusted TLS channels the data may be optionally signed.

Owners supporting either protocol provide means to discover pledges and for pledges to obtain bootstrapping data from. For BRSKI, it depends on a domain proxy to discover pledges mDNS or GRASP broadcast messages. It introduces only one bootstrap source which the domain registrar. The registrar acts in principal as the owner's CA and is responsible for bootstrapping pledges as well as communicating with the manufacturer services. Meanwhile, SZTP provides more options for pledges for discovery. An owner can administer a DNS server, a DHCP server, or a redirect server to respond to pledges discovery broadcast message and redirect the pledge to a source it can bootstrap from. However, similar to BRSKI, pledges can only obtain bootstrapping information from one source, namely, a bootstrap server. That is a server which is capable of providing redirect information as well as onboarding information. Nevertheless, bootstrapping sources of both protocols can be hosted locally at the owner side or remotely by a trusted third party.

Both protocols allow owners to authorize the pledges which are allowed to initiate the protocol according to their vendor-specific serial numbers which are known in advance to a protocol run. Moreover, BRSKI can optionally couple the serial number with a specific pledge type and/or a specific vendor.

An owner needs to know the URI for the MASA responsible for issuing vouchers for the pledge running the bootstrapping protocol. BRSKI recommends that the IDevID of the pledge contains URI for MASA to contact. On the other side, SZTP does not propose an inherent method to obtain the MASA URI. It rather relies on the OOB owner-manufacturer enrollment process to acquire the URI. After contacting the MASA, owners decide on whether to accept to bootstrap the pledge. In both protocols, the decision is made according to the verification of the attributes of the issued voucher. Also, BRSKI checks the MASA audit-log

for further verification. As SZTP is more concerned with the pledge-owner communication, checking of the MASA audit-log is not discussed. However, SZTP does not limit it.

BRSKI allows for pledge ownership transfer through issuance of new vouchers. Even though it is also feasible in SZTP, it is out of the protocol's scope. On the contrary, SZTP allows owners to provision domain specific configuration to pledges during the bootstrapping process, but BRSKI is not capable of providing such information.

A pledge is the protocol initiator for both protocols. Pledges compatible with either protocol must have an initial pre-configured state with a set of minimum required parameters. The initial state must contain the pledge IDevID and the trust anchors needed to verify the MASA. Optionally for both protocols, the device can be configured with a list of well-known remote bootstrap servers and their respective trust anchors. Moreover, SZTP pledges may be configured with a TLS client certificate and its related intermediate certificates. The initial state trust anchors are not updatable via either protocol methods.

The bootstrap source must be authenticated even if it is not yet trusted. For either protocol, pledges establish a TLS channel with the bootstrap source. But if the pledge is not able to validate the source against one of its trust anchors, the TLS channel is provisionally trusted.

For both protocols, pledges must check the validity of received vouchers with respect to their expiry time. This is dependent on whether the device has an accurate clock and on whether the voucher is nonced or nonceless, as expiry time might be omitted in nonced voucher. Pledges must also check the revocation status of received certificates, if revocation information is provided or if revocation checks are enforced.

On the Manufacturer side, BRSKI optionally supports manufacturer tracking, while SZTP does not. Each protocol relies on the MASA for voucher issuance. However, BRSKI relies on the MASA for further functionalities it supports such as voucher renewal and maintaining and providing the voucher audit-log.

5 Post-Compromise Security

With the increasing number of security and data breaches, individuals and organizations are far more concerned about their digital security and their data privacy. This is realized by the broadening discussion of data usage by companies and governments and their implications on privacy. Furthermore, this has reflected a growing demand for secure communication techniques in the various use cases of digital communication. Consequently, researchers and developers have been continuously seeking to improve the security features of algorithms and protocols and propose new ones rather than focusing solely on applications' features. Nevertheless, many deployed systems are built with alleged security guarantees, which are often violated. The reasons are the internal state compromises that occur through participants' exposures or the exploiting of newly discovered vulnerabilities in the presumed secure algorithms or protocols.

A key compromise in message exchange algorithms that use the duplicate static keys for all messages allows an adversary to decrypt all messages. The long-term private key defines a party's identity; thus, any compromise of the identity key can lead to an impersonation attack against its owner. A critical security feature of communication protocols is forward security which ensures the security of previous sessions if the current key is compromised. Nevertheless, the following question remains: Are there security guarantees for future sessions after a compromise? The security feature in question is referred to as backward secrecy, future secrecy, or post-compromise security. The "future secrecy" term was composed by Marlinspike [43] in their discussion of an advanced ratcheting algorithm, while "post-compromise security" was first coined by Cohn-Gordon et al. [44] in their definition and formal realization of the concept. In this thesis, we use the terms future secrecy and post-compromise security interchangeably since they are less confused with the forward secrecy property, from our point of view. According to [44], post-compromise security is informally defined as "A protocol between Alice and Bob provides Post-Compromise Security (PCS) if Alice has a security guarantee about communication with Bob, even if Bob's secrets have already been compromised.". In the narrower context of long-term keys, it translates to that an algorithm is post-compromise secure if the compromise of a long-term key does not allow subsequent ciphered messages to be decrypted by passive attackers. Figure 5.1 depicts the difference between forward secrecy and future secrecy. Following the model of [44], a client compromise can be broken down into weak compromise and total compromise. A weak compromise is when the client key is still private, however, the adversary has limited access to its usage. For example, an adversary can manipulate a compromised client to obtain a digital signature over data of his choice using the desired private key. On the other hand, a total compromise is more severe as the adversary learns the private key of the client and impersonate him without restrictions. Their work showed that future secrecy is achievable in both forms of compensation. Future secrecy in a weak compromise model is achievable if both parties can construct a secret in the "test" session given that the adversary's access to the compromised key is

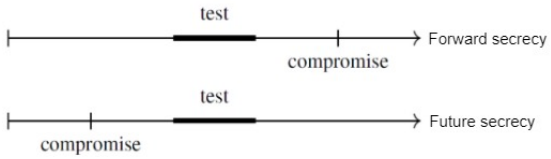

Figure 5.1: Forward and future secrecy. Figure reproduced from [44]. The "test" session indicates the session the adversary aims to attack. For either case, the aim of the security property is to guarantee security of the test session after compromise.

revoked within that session. As for a total compromise model, future secrecy is achievable if both parties are capable of running a secure intermediate session where they construct a new intermediate secret before the "test" session, where both secrets, the compromised and intermediate, are used in the "test" session to derive a new secret. Likewise, the adversary's access to the compromised key is revoked in the intermediate and "test" sessions.

The Signal protocol is considerably one of the renowned instant messaging protocols with advanced security features [21]. It provides the critically desired forward and future secrecy, in addition to immediate decryption. These properties have never been accomplished in tandem by communications systems prior to the advent of Signal [45]. Moreover, the protocol can be utilized in synchronous and asynchronous settings. Signal is composed of two main components: The X3DH protocol [46] that is responsible for the initial authenticated key exchange (AKE), and the double ratchet algorithm [47] that is responsible for generating secure message key in Zero Round Trip Time (0-RTT). This chapter reflects on the details of the signal protocol components. Precisely, our discussion is only concerned with two-party communication, while group communication is out-of-scope. We present a formal verification of X3DH using OFMC. Furthermore, we survey work done on formal verification of the Signal protocol and the post-quantum security of Signal.

5.1 Extended Triple Diffie-Hellmann (X3DH)

Key establishment protocols are utilized by communicating parties to establish shared secrets, in some cases, with the aid of a trusted third party [48]. As per [48], key agreement is a key establishment technique in which a shared secret is derived by parties as a function of information related to them, such that the secret in not predeterminable nor deducable by outsiders.

The Security of protocols is required to be proven to avoid the devastating effects of malicious attacks. Therefore, verification of security protocols is a vital process. Verification is proving that a protocol model is secure by achieving a set of security goals. There are various model checking tools that provide verification.

X3DH is a key agreement protocol intended for asynchronous settings [46]. The protocol can establish a shared secret between two parties in an environment where it is recurring that one of the parties is offline and the other wishes to send it an encrypted message. Also, X3DH provides the desired feature of forward secrecy.

This chapter discusses the X3DH key agreement protocol [46] and verifies the security of the

Owning Role	Notation	Description
Alice	IK_A	Long-term Identity Key
	EK_A	Ephemeral Key
Bob	IK_B	Identity Key
	SPK_B	Signed Prekey
	OPK_B	one-time Prekey

Table 5.1: X3DH keys.

protocol using OFMC [12].

5.1.1 Protocol Overview

5.1.1.1 Roles

A protocol run involves three roles: Alice, Bob, and a server. In this description, Alice wants to send an encrypted message to Bob. Bob is the party that may be offline at that time and wishes to enable other parties to derive a shared secret with it, through a set of public information Bob publishes. The server is responsible for 1. storing the public information published by Bob, and 2. storing messages for offline parties till they are fetched. The server is not trusted, however, it is assumed to be resilient against DoS.

5.1.1.2 Keys

X3DH utilizes Elliptic Curve asymmetric key pairs. All keys used in a protocol run must all be derived from the same curve, either $X25519$ or $X448$. Each role has to have a set of keys to run the protocol. Table 5.1 lists the public keys required for each role. Note that for each public key, there exists a corresponding private key at its owner.

- *Identity keys*: They are long-term public keys known by their corresponding parties before the protocol run.
- *Ephemeral Keys*: This type of key pair is freshly generated within the protocol run.
- *Signed Prekey*: This key pair is generated and signed by its owner. The prekey is signed using the private identity key. The life time of this key pair is shorter than that of the identity key pairs as it is updated periodically by its owner. The corresponding role owns only one signed prekey at a time.
- *One-time Prekey*: The corresponding role generates multiple one-time prekey. Each is can be used for only one protocol run. The responsible party is supposed to supply the these keys as they should not run out. In case there are not any keys left, the protocol can run, however without one of the DH operations as explained in section 2.3.

5.1.1.3 A Protocol Run

At first, the protocol starts with a registration phase. Initially, The party acting in the role of Bob publishes to server the public information required to run the protocol with it by any party acting as Alice. Bob publishes his $IdentityKey(IK)_B$, SPK_B, Bob's prekey signature, and a set of OPK_B.

Next, the inline party sends the initiates the protocol by sending the first message. First of all, Alice fetches a *prekey bundle* from the server to contact Bob. This bundle contains:

- Bob's Identity key IK_B.
- Bob's signed prekey SPK_B.
- Bob's prekey signature.
- If exists, a one-time prekey OPK_B. The server deletes the OPK_B sent to Alice.

Before proceeding, Alice verifies the prekey signature and quits the protocol if the verification fails. At this point, Alice has enough information from Bob to deduce a shared secret. Alice generates her Ephemeral key pair EK_A. With the set of available keys, Alice performs three DH operations which are extended to four if a OPK_B is available. Figure 5.2 presents the relation between the keys.

- $DH1 = DH(IK_{A_p}{}^6, SPK_B)$
- $DH2 = DH(EK_{A_p}, IK_B)$
- $DH3 = DH(EK_{A_p}, SPK_B)$
- $DH4 = DH(EK_{A_p}, OPK_B)$

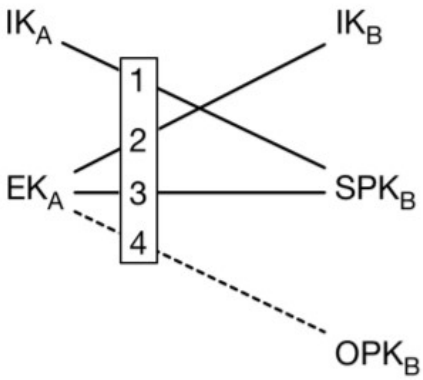

Figure 5.2: X3DH operations [46].

The DH outputs are fed into a KDF to generate the shared secret SK. Next, Alice deletes her EK_{A_p} and all DH outputs for forward secrecy. At this moment, Alice is ready to send the initial message with the content encrypted using SK. The initial message from Alice to

[6] p indicates the private key of the key pair.

Bob has to have enough information for Bob to generate the same SK. The initial message contains:

- IK_A
- EK_A
- Identifiers of Bob's prekeys used by Alice
- An initial ciphertext. The ciphertext can be used as an initial message for a post-X3DH communication protocol, e.g. Double Ratchet algorithm.

Upon receiving the initial message, Bob attempts to derive the SK. Using the key identifiers sent by Alice, Bob loads the private keys corresponding to the public keys Alice used. In combination with the keys IK_A and EK_A Alice sent, Bob can compute the same SK by doing the three (or four) DH operations.

Next, Bob attempts to decrypt the ciphertext. If the message is successfully decrypted to the expected format, e.g. the format of the first message of the post-X3DH protocol, then the protocol run was successful. Otherwise, Bob aborts the protocol and discards the SK.

5.1.1.4 Security Considerations

Authentication is essential for both parties to guarantee the identity of who they are communicating with. Thus, Alice and Bob must authenticate the keys IK_A and IK_B. However, the protocol specification does not discuss authentication methods.

The one-time prekey used in the fourth and optional DH calculation is for protection against replay attacks as they ensure freshness of the protocol run. Absence of a one-time prekey could lead a replayed message to be accepted by Bob believing that Alice had sent it in the current protocol run.

A server can be a cause of attacks if malicious. It can carry out a Denial of Service attack if it refuses to forward the messages. It can deliberately not distribute one-time prekeys, exposing the protocol to replay attacks. Also, one party can drain all the one-time prekeys, if the server is not attentive to such action, leading to replay attacks.

5.1.2 OFMC Verification

This section aims at formally verifying the X3DH protocol using the automated verification tool OFMC. Briefly, OFMC uses the clear and declarative modeling language AnB. The goal of the tool is verify the security of the modeled protocol, under an intruder model, in a fashion similar to deciding whether a mathematical statement is true or false. The intruder model is based on the famous Dolev-Yao model [15]. The intruder is assumed to control the network. Therefore, the network should not be relied on for any protection guarantees. Furthermore, the intruder is assumed to be knowledgeable to all cryptographic primitive. Lastly, OFMC does not rule out the possibility that the protocol participants can be dishonest.

Presented in listing 5.1 the code for modeling our variant of a secure X3DH protocol under the Dolev-Yao intruder model in OFMC using the AnB language.

```
Protocol : X3DH #Protocol name

Types :
       Agent A, B, s;
       Function kdf, pk, ik;
       Number g, NA, EKA, OTP, PREKEY, MSG1, MSG2;

Knowledge :
          A : A, B, s, g, pk(s), pk(B), pk(A), inv(pk(A)), ik(A), {A, pk(A)}
              ↪ inv(pk(s)), kdf;

          B : A, B, s, g, pk(s), pk(A), pk(B), inv(pk(B)), ik(B), kdf;

          s : A, B, s, g, pk, inv(pk(s)), kdf, ik;

where A!=B

Actions:

       B -> s: {B, exp(g, ik(B)), exp(g, OTP), {exp(g, PREKEY)}inv(pk(B)) }
           ↪ pk(s) # B registers at server

       s -> A: {B, pk(B)}inv(pk(s)) # s announces B is ready

       A -> s : A, B, NA # later in time, A asks to communicate with B

       s -> A : { A, B, exp(g, ik(B)), exp(g, OTP), {exp(g, PREKEY)}inv(pk(B
           ↪ )), NA }inv(pk(s))

       A -> B : {A, pk(A)}inv(pk(s)),

                {exp(g, OTP), exp(g,EKA),exp(g, ik(A)), exp(g,PREKEY)}inv(pk(
                    ↪ A)),

                {| A, B, MSG1 |}kdf(
                      exp(exp(g,ik(A)), PREKEY), #DH1
                      exp(exp(g,EKA), ik(B)), #DH2
                      exp(exp(g,EKA), PREKEY), #DH3
                      exp(exp(g,OTP), EKA) #DH4
                      )

       B -> A : {| B, A, MSG2 |}kdf(
                      exp(exp(g,ik(A)), PREKEY), #DH1
                      exp(exp(g,EKA), ik(B)), #DH2
                      exp(exp(g,EKA), PREKEY), #DH3
```

```
42                      exp(exp(g,OTP), EKA) #DH4
43                      )



46 Goals:
47        B authenticates A on exp(g, EKA), exp(g, ik(A))
48        A authenticates s on exp(g, OTP), exp(g, PREKEY), exp(g, ik(B))
49        B authenticates s on exp(g, OTP), exp(g, PREKEY), exp(g, ik(B))
50        MSG1 secret between A,B
51        MSG2 secret between A,B
```

Listing 5.1: X3DH OFMC Model

5.1.2.1 Types section

Here, the parameters of the protocol are defined, e.g. variables, constants, roles, etc. Line 4 defines the participants of the protocol of type **Agent**.

- A and B: Variables indicating Alice and Bob. A variable may be dishonest in an OFMC protocol run.
- s: Constant indicating the server. Constants act as trusted third parties.

Line 5 defines the functions of the protocol.

- kdf: Key derivation function.
- pk: A function to model asymmetric key pairs. A public key for Alice is modeled as $pk(A)$ and the corresponding private key is computed using the internal function $inv()$ as follows $inv(pk(A))$.
- ik: models the private component of the identity key of a party for the DH calculations.

Alice and Bob have long-term signing keys modeled by the function $pk()$ and long-term identity keys modeled by the function $ik()$.

Line 6 defines the numbers used in the protocol. Lower-case numbers are constants, while upper-case numbers are random and freshly generated during a protocol run. Some private components of keys are modeled as numbers as they are required to be freshly generated during a protocol run which can not be done using functions.

- g: Public prime base for DH calculations.
- NA: Alice's Nonce.
- EKA: The private component of Alice's Ephemeral key.
- OTP: The One-time prekey's private component.
- $PREKEY$: the prekey's private component.
- $MSG1$ and $MSG2$: Random numbers used as placeholders for random and fresh messages.

5.1.2.2 Knowledge section

This section of the model defines what knowledge is initially known to each party before the protocol starts. Line 9 defines Alice's knowledge which contains, in addition to the previously defined parameters, Alice's certificate modeled as $\{A, pk(A)\}inv(pk(s))$. This translates to Alice's identity A and Alice's public key $pk(A)$ are signed using the servers private key $inv(pk(s))$.

Line 15 holds a *where* clause that strictly defines A and B as different parties to avoid the scenario where A or B talk to itself.

5.1.2.3 Actions section

The Actions section states the protocol's message sequence between the participating parties for a successful protocol run. Throughout this section messages are referred to according to their line number, e.g message 19 is the message in line number 19 of the model code.

Message 19 represents the registration phase where Bob publishes his public information to the server. The whole message is encrypted for authenticity by the server's public key $pk(s)$. The message contains the following:

- $exp(g, Z)$: The public DH key of the corresponding private key Z. As known in DH, the public key of a party is of the form $g^Z \ mod \ p$. For simplicity, OFMC omits the prime modulus operation $mod \ p$. Hence, it is needed to only model the public key g^Z by using the OFMC modulus exponentiation function $exp(g, Z)$.
- $\{exp(g, PREKEY)\}inv(pk(B))$: The prekey of Bob signed by Bob's signing key.

Message 21 is not an actual part of the protocol, however, it is included for the sake of modeling the protocol with the chosen tool. This is because of the asynchronous communication model of OFMC and the strands concept [**"gls –ofmc"Tut**].

In message 23, Alice requests to fetch from the server a key bundle to communicate with Bob. A nonce NA is attached for freshness.

Message 25 is the server's response to Alice's key bundle request. The message contains the requested key bundle required to communicate with Bob, as well as Alice's nonce. The whole message is integrity protected by the server's signature.

Message 27 represents the initial message from Alice to Bob. It is composed of the following:

- (Line 27) Alice's certificate for authenticity.
- (Line 29) Bob's keys that Alice received from the server which will used to compute SK. All these keys are signed by Alice for integrity protection.
- (Line 31) The initial ciphertext encrypted by the symmetric key SK. Each output of the four DH operations is input into the KDF resulting in SK.

Message 38 is the last message of the protocol run. Here, Bob shows he is able to compute the same SK.

Notably, the modeling of the DH operations in messages 27 and 38 is unnatural in contrast to how DH is normally performed. Ordinarily, a DH secret is the computation of $exp(g^M, N)$

where g^M is a public value and N is a value private to the party performing the operation. According to exponential power of power rule $exp(g^M, N)$ is equivalent to $exp(g^N, M)$. Therefore, the other party should follow that intuition to result in the same secret output as its counterpart. It is observable that the symmetric key computation in message 27 is identical to the one in message 38. The behavior for modeling $DH1$, $DH2$, and $DH3$ of message 27 is not expected, since Alice is explicitly using Bob's secrets ($PREKEY$ and $ik(B)$) in the DH operations although they ought not be in Alice's knowledge. This behavior is present in message 38, however, it is as it should be. To the reader, it could mean that Alice is knowledgeable of Bob's private keys, but, this is not the case. OFMC aims to abstract the key creation process and makes it implicit for the tool users. The OFMC compiler comprehends the exponential power of power property which enables it to automatically do the computation. Despite the previous clarification, $DH4$ computation contradicts what was mentioned. The M and N values for $DH4$ are swapped back to what would be naturally expected in comparison to the previous DH operations of message 27. Although, this behavior is considered unnatural for message 38 as now Bob is considered to be using Alice's secret keys. However, this exceptional behavior for $DH4$ is a result of a limitation of OFMC's AnB-translator message composition heuristics, as the translator is not capable of recognizing four inversions in a row. Therefore, the exceptional behavior is a workaround to make the AnB-translator realize how to compose the message. This workaround was thankfully guided by Prof. Mödersheim, the composer of OFMC.

5.1.2.4 Goals section

This section of the model lists the security goals which the protocol must achieve at the end of a run. There are two types of goals

- Authenticity: When a party wants to make sure a certain message has been generated and sent by the other party. In this model, Alice and Bob authenticate each others' public keys. Bob authenticates the server on the keys he received in message 27 from Alice, to be sure that the keys are forwarded by Alice from the server without tampering.
- Secrecy: The requirement for a message to be secret only between some parties.

5.1.2.5 Deviations from specification

- **Registration Phase:** The specification was not specific about how to securely publish Bob's keys to the server and the security of the connection between the server and the communicating parties. Therefore, the registration message from Bob to the server is encrypted by the server's public key.
- **Use of Signatures:** The specification discouraged the use of signatures as they reduce deniability which is a desired feature in the messaging application setting which the protocol is intended for. However, this model used signatures.
  a) Signature are used in responses from the server to assure integrity of messages to receivers.
  b) Alice and Bob have an additional key pair for signing other than the identity key pairs. They are used to sign parts of messages which were vulnerable to attacks by

intruders and would break the protocol's security. For example, Bob's keys which Alice used to compute SK in message 27.

- **Use of Certificates:** Alice used a certificate signed by the server to authenticate itself in-band as opposed to the specification which ruled the authentication between the parties as a necessary but out of scope operation.

5.2 Double Ratchet Algorithm

The adoption of a ratchet mechanism is a popular mechanical approach that enables forward movement but inhibits backward movement. A ratchet-like action is performed in the context of secure communication by utilizing randomization in every state update, such that a compromised state is insufficient for the decryption of any subsequent transmission. The Double Ratchet is a cryptographic asynchronous message exchange algorithm that provides high security to the communicating parties. Asynchronicity in this context refers to that even if the counterpart is not online, the messages should be conveyed (or the key exchange should be done) for a two-party conversation. Furthermore, a significant design goal dubbed 0-RTT is the ability to transfer payload data without necessitating online exchanges.

The algorithm enables the exchange of encrypted messages based on a shared secret key between the two parties where each message is encrypted with its specific ephemeral key. The double ratchet algorithm is a combination of two ratchet constructions which provide enhanced security properties. The first outer ratchet is inherited from OTR's asymmetric ratchet to benefit from its future secrecy property which is obtained through the use of ephemeral key exchanges. Coupled by an inner symmetric ratchet for forward secrecy, the algorithm was formed and was formerly named Axolotl Ratchet. Additionally, KDF chains are a core concept of the algorithm. This section goes through the methodology of the algorithm, its features, and its security properties.

5.2.1 KDF Chain

As mentioned in section 2.1.7, a KDF is a function that takes as input a key and an input and produces a cryptographically secure hash output that is random-like. A KDF chain is a series of connected KDFs where one output key of a KDF is a KDF key input of a succeeding KDF. Figure 5.3 shows an illustration for a KDF chain that takes three external inputs and produces three random output keys.

The algorithm session has three KDF chains: root chain, sending chain, and receiving chain. Each chain is advanced whenever its relevant ratchet performs a step. More details about ratchet steps and their relation to the KDF chains are discussed in later sections.

KDF chains have a set of characteristics [47]:

- *Resilience:* An adversary who does not know the KDF keys perceives the output keys as random. Even if the opponent has complete control over the KDF inputs, this is still valid.

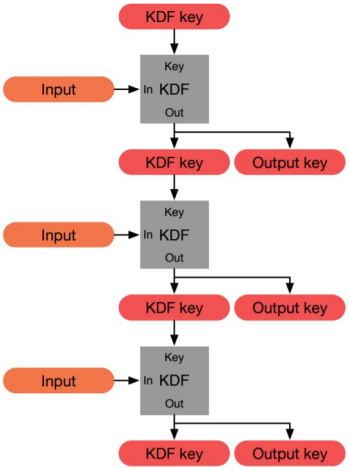

Figure 5.3: A 3 input KDF chain [47].

- *Forward secrecy:* An adversary who discovers the KDF key at some point in the future cannot distinguish between past output keys and random.
- *Future secrecy:* If future inputs have contributed adequate randomness, future output keys seem random to an adversary who learns the KDF key at some point in the future.

5.2.2 Symmetric-key Ratchet

The sending and receiving chains are constructed of symmetric-key ratchets where Alice's sending chain is equivalent to Bob's receiving chain and vise versa. Essentially, a symmetric-key ratchet is a KDF chain with a constant input through out the chain steps. However, unlike with random input, a constant input does not provide future secrecy.

In a symmetric-key ratchet, the KDF key illustrated in figure 5.3 is the chain key, while a KDF's output key is a unique message key. For a sending chain, a message key is used to encrypt the outgoing message. On the other side, a message key in the receiving chain is used to decrypt the incoming message. The process of calculating the next chain and message keys is an advance in the sending/receiving chain. This is referred to as a *symmetric ratchet step*.

Message keys are not re-introduced into the KDF chain, therefore, they can be discarded without proposing any security risk to earlier or later ratchet outputs. Also, it is possible to store message keys to handle out-of-order messages on the receiving end. Storing message keys introduce a risk to only their respective messages. However, a compromise of a chain key can lead to further compromise of future chain keys as future chain keys rely on previous ones.

5.2.3 Diffie-Hellman Ratchet

The DH ratchet provides the future secrecy property. When paired with the symmetric ratchet, the combination addresses the symmetric ratchet lack of future secrecy. Each party generates a DH key pair known as their *ratchet key pair*. The parties exchange their public

keys within message headers with which each can compute an equivalent DH secret output using their own private key that is equivalent to the public key they sent. The process of generating a new DH output when receiving a new public key is referred to as a *DH ratchet step*. Accordingly, a passive adversary that compromises one party's private key at a point in time is incapable of deducing the upcoming DH outputs due to the use of newly generated key pair for each ratchet step.

Next, we discuss an algorithm run where a DH ratchet is advanced to create two different DH secret outputs between Alice and Bob. Bob is assumed to be the algorithm initiator. Figure 5.4 serves as visual aid for our discussion where we proceed in steps as labeled in the diagram.

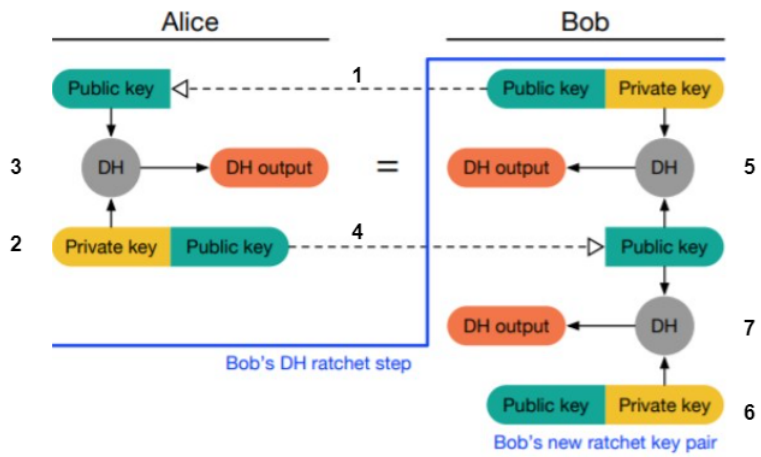

Figure 5.4: Bob DH Ratchet step. Figure reproduced from [47].

- *Step 1:* Bob generates his first ratchet key pair to send the initial message to Alice. The initial message contains Bob's public key.
- *Step 2:* Upon receiving Bob's public key, Alice generates her ratchet key pair.
- *Step 3:* Alice performs a DH calculation between her private key and Bob's public key generating her side of the DH output.
- *Step 4:* Alice advertises to Bob her public portion of her ratchet key pair that was used to generate the DH output.
- *Step 5:* After obtaining Alice's ratchet public key, Bob performs a DH calculation between it and his current ratchet private key resulting in Bob's copy of the shared DH output.
- *Step 6:* Furthermore, Bob generates a new ratchet key pair to generate the next DH output.
- *Step 7:* Bob performs a DH calculation between hit new ratchet private key and Alice's public key generating a new DH output.

The steps described above are Bob's DH ratchet step as Bob was the initiator of the ratchet step by advertising his public key. Similarly, Alice's ratchet step starts by sending her public

key to Bob and concludes after performing the same steps mentioned above but with the roles reversed.

Linking the algorithm to the messaging context, the DH outputs represent the sending and receiving chains' root keys. Each party need to have two chains, sending and receiving chains. Alice's sending chain is equivalent to Bob's receiving chain and vice versa. As depicted in figure 5.5, when a party receives a public key and computes its chain key using a private key that was generated prior to receiving the public, the resulting chain key is the receiving chain key. However, if they private key is generated after receiving the public key, the resulting key is the sending chain key. Simply put, if the public key is used in a DH operation upwards it produces a receiving chain key, while using it in a downwards DH operation generates a sending chain key.

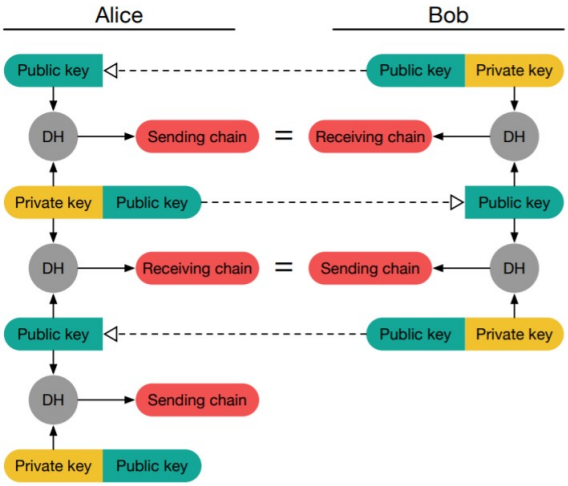

Figure 5.5: DH chain keys generation [47].

However, the description so far is simplified. The algorithm is augmented by a KDF chain to improve resilience and future secrecy. The KDF chain key is a shared secret between both parties while the KDF inputs are the DH outputs from the DH ratchet. Every step through the KDF chain results in a new KDF root key and a sending/receiving chain root key. So a full DH ratchet step consists of updating the root KDF chain twice, generating both a sending and a receiving chain key. Figure 5.6 illustrates a full DH ratchet step.

5.2.4 Double Ratchet

The double ratchet combines both the symmetric-key ratchet and the DH ratchet to merge the security advantages provided by each. The outer ratchet is the DH ratchet that provides ephemeral DH outputs which improve future secrecy. The DH outputs are fed as inputs to a KDF root chain lying between the outer and inner ratchets. It contributes to augmenting the security features by providing resilience and forward secrecy to the generated keys. The initial Root Key (RK) for the KDF chain is a shared secret between the parties, e.g. an

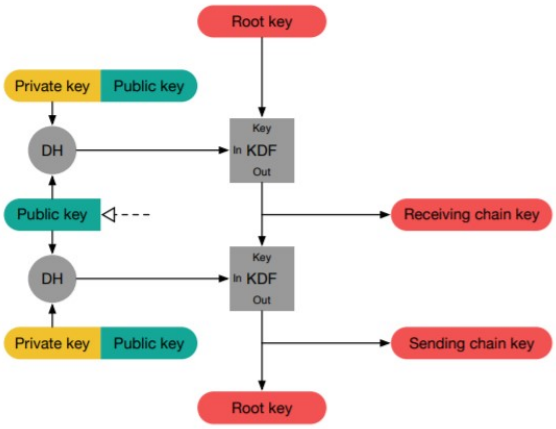

Figure 5.6: A full DH Ratchet step [47].

output of a key agreement protocol. The KDF chain outputs a new RK for the next KDF and a new root Chain Key (CK) to create a new sending/receiving chain. Lastly the inner ratchet is the symmetric-key ratchet. Their root chain keys are the CKs generated from the previous layer. They form the sending and receiving chains. When this ratchet is advanced it generates a new CK and a message key. The CK is used in the next KDF and the message key is used to encrypt or decrypt the message, depending on its respective chain.

Figure 5.7 serves as an example scenario for an algorithm run as it illustrates Alice's perspective of her double ratchet algorithm after a series of steps. The figure is split into four columns

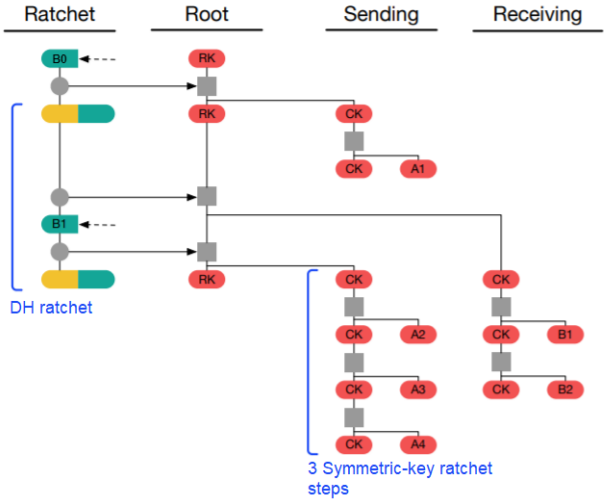

Figure 5.7: Double ratchet from Alice's point of view. Figure reproduced from [47].

describing the aspects of the algorithm and how they are connected to each other. The first column represents the DH ratchet, the second shows the root KDF chain, and the last two depict the sending and receiving symmetric-key ratchets. Messages are denoted by the initial

of the sender and the number of the message, e.g. $A1$ denotes Alice's first message. Keys follow the same abbreviation convention, e.g. $B1$ denotes Bob's first message key. Whether the key is a public key or a message key is represented by their highlighted color. Message keys are highlighted in red while public keys are highlighted in green.

Alice is initialized by receiving the initial public key $B1$ which she uses to compute the input for the root KDF chain through a DH operation. Along with the pre-shared RK, Alice computes a root CK for her sending chain and a new RK. Using the initial CK, Alice creates a sending chain by performing a symmetric-key ratchet and producing a new CK and a symmetric key $A1$ that she uses to encrypt her first message to Bob.

Till this point, Alice is able to send encrypted messages to Bob as it had created her sending chains. However, she cannot decrypt Bob's messages as she has not yet created a receiving chain to produce the keys required to decrypt Bob's messages. As soon as Bob sends his next message with a new public key, Alice uses the public key $B1$ attached in the message header to step her DH ratchet. $B1$ is used in a DH operation with the previously created key pair to ultimately produce a root CK to create a receiving chain. Alice performs a symmetric-key ratchet step of her receiving chain to produce the symmetric key $B1$ that decrypts the received message. Alice advances the same receiving chain to produce message keys to decrypt further message from Bob, e.g. $B2$, as long as the received messages do not contain new public keys in the message headers. However, since Alice received $B1$ and had to perform a full DH ratchet step, she has to generate a new ratchet key pair ultimately create a new sending chain. This implies if Alice wishes to send further encrypted message, she must use the new sending chain to generate the new message key. Thus in figure 5.7, $A2$ is produced from the newly generated sending chain not the old one. As long as Alice has not received a new public key, she keeps advancing the sending chain to generate encryption keys for her outgoing messages, e.g. $A2$ and $A3$ in figure 5.7.

Parties should delete old keys as they no longer have a use. Keys are considered old if they have already been used and they are not going to be used in the future. For example, RKs and CKs which have already been input into KDFs or message keys that have already been used to encrypt/decrypt messages are considered old keys. Deleting old and useless keys is a good security practice as it prevents intruders from the possibility of recomputing keys that are dependent on those keys. Nevertheless, some keys can be saved to support additional functionalities such as handling out-of-order messages as explained in section 5.2.4.1.

5.2.4.1 Out-of-order Messages

The Double ratchet algorithm can manage messages arriving out-of-order by simple additions to the message headers from the sender side. By including the message's index N ($N = 0$ for message 1) in the sending chain and the length PN of the previous sending chain.

If the sent message does not trigger a DH ratchet step on the receiver side, then the out-of-order message belongs to the current receiving chain of the receiver. The difference between N and the actual receiving chain length is the number steps the receiver has to advance his receiving chain to obtain the message key required to decrypt the received message. The intermediate message keys for the skipped messages are stored in case they arrive later.

On the other hand, if the message triggers a DH ratchet step, then the receiver uses the PN to determine how many messages are skipped of his current receiving chain. The difference between the current length of the chain and PN is the number of steps needed to advanced in the current chains and their produced message keys have to be saved. In this case, N determines the number of skipped messages in the new receiving chain after advancing the DH ratchet. Likewise, the message keys for the skipped messages have to be saved supposing they are delayed for any reason.

Implementation wise, the algorithm defines a *MAX_SKIP* constant that specifies the maximum number of message keys that should be saved for skipped messages. It should tolerate message loss or delay. However, it should not be too high that it allows for a malicious sender to trigger excessive recipient computation.

5.2.4.2 Header Encryption

As headers contain ratchet public keys as well as PN and N values, a passive attacker can infer the ordering of messages inside a session, or which messages belong to particular sessions. Therefore, encrypting message headers can reinforce messages security. Header encryption is achievable through each party having symmetric keys for both sending and receiving chains to encrypt or decrypt messages' headers. One chain header key is responsible for handling the header encryption of all messages within its respective chain. The header keys are integrated into the double ratchet algorithm as follows.

Originally, Alice and Bob had only one pre-shared secret, the RK. To integrate header encryption, the initial shared secrets need to include two unique HKs, one for the sending chain and the other for the receiving chain (referred to as Next Header Key (NHK) in [47]). The initial HKs are used for their respective first generated chains of the algorithm. To continuously obtain new HKs for the newly generated chains which are a result of advancing the DH ratchet, the root KDF chain is amended. Instead of generating only a new RK and a new respective CK with each step, the root chain is modified to additionally produce a NHK of the relevant chain. A NHK is a HK that is to be used for the next chain of the same type that is to be generated after the next ratchet step. In this manner, at any point in time, before generating any chain of the algorithm, there exists already a header key to be used for that chain. When the chain is created, the NHK becomes the current HK for the chain and a new NHK is generated simultaneously for the upcoming chain. Figure 5.8 shows an illustration of when HKs are generated and how NHK are used for the next chains.

5.3 Formal Verification of Signal Protocol

For completeness, we mention below former related work aimed at formally analyzing the Signal protocol without diving into details.

Cohn-Gordon et al. [49] were the first to address Signal's security in a formal manner. The verification methodology is highly comprehensive and sophisticated, and it is designed particularly for the Signal protocol [45]. A completely adversarially controlled network is used to evaluate the protocol. Through the definition of a security model, the research covers

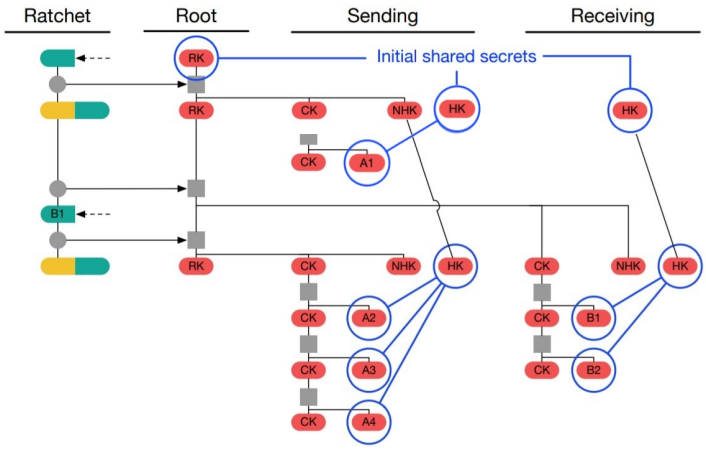

Figure 5.8: Usage of HKs and NHKs in the double ratchet algorithm. Figure reproduced from [47].

Signal's X3DH and Double Ratchet protocols as a multi-stage authenticated key exchange protocol. The model depicts the ratcheting key update structure as a multi-stage model, where instead of just a sequence, it is a tree-like structure of stages that reflect the chains in Signal. The model enables different parties to run numerous, simultaneous sessions, each with its own set of stages. Secrecy and authentication of message keys in the computational model, under a rich compromise scenario, are the high-level features targeted to be verified by hand. Nevertheless, forward and future secrecy are implied goals, as derived session keys should be kept secret in a range of compromise circumstances. For instance, if a long-term secret is compromised but a medium or ephemeral secret is not, or if a state is compromised and a secure asymmetric stage happens afterwards. Since Signal does not cleanly separate key exchange from subsequent data messages, the model had to reorder some procedures to achieve this separation. In addition and contrary to Signal, the model does not re-use DH keys for signatures. Finally, the research shows that Signal's cryptographic core delivers the desirable security attributes specified in the security model, based on normal cryptographic assumptions. Reassuringly, its design is free of any severe defects. The model, on the other hand, does not address Signal's instantaneous decryption feature, which is a distinctive feature that privileges it to other protocols that lack it [45]. Furthermore, because the model is exclusive to the Signal protocol, it cannot be used as a reference notion for ratcheted key exchange (RKE) because it provides a lower degree of security than would be expected for RKE [50].

According to Alwen et al. [45], one of the critical shortcomings of formal Signal-related investigations prior to their work [50, 51, 52, 53] is that they all accomplish FS and PCS by expressly forgoing immediate decryption and, therefore, message-loss resilience. Abandoning this property facilitates the algorithmic design for these provably secure Signal alternatives, but it also makes them inadequate in situations when message loss is a possibility. For instance, the protocol may employ an unstable transport technology like UDP, or a centralized server could be dropping messages due to a wide variety of unforeseen events. The authors contend that none of the studies available at the time were entirely adequate. To address this issue, they provided a formal and generic definition of secure communications that includes the necessary security properties: forward secrecy, future secrecy, and, in particular, immediate

decryption. Their work is the first to address the analysis of the Signal protocol, primarily the double-ratchet aspect, without waiving the immediate decryption property and in a well-defined generalized model that is not only specific for the Signal protocol. The methodology of the paper is generalizing and abstracting out the reliance on the specific Diffie-Hellman key exchange by not relying on random oracles, in addition to clarifying the role of various cryptographic hash functions used inside the current Signal approach. The outcome was a generalized modular model of the Signal protocol which was developed using the following modules: continuous key agreement (CKA), forward-secure authenticated encryption with associated data (FS-AEAD), and a two-input hash function. The design of their model is inclusive of other well-studied forms of state compromise proposed by [50] and [52] without sacrificing the immediate decryption property.

5.4 Post-Quantum Security of Signal Protocol

The majority of cryptographic primitives are based on mathematical concepts that can be computed theoretically. However on the practical side, these calculations, are computationally challenging. Current cryptographic primitives are robust enough that they cannot be broken by an adversary with typically limited processing capacity. Previously, Shor [54] and Grover [55] proposed quantum algorithms that, in theory, can infringe the cryptographic principles in a wide range of cryptography primitives.

Shor's algorithm is a quantum computer algorithm for determining an integer's prime factors. The algorithm executes in polynomial time, implying that the integer factorization problem can be performed effectively on a quantum computer. As a result, it could be used to break public-key cryptography schemes like RSA, Finite Field Diffie-Hellman key exchange, and Elliptic Curve Diffie-Hellman key exchange.

Grover's algorithm, commonly known as the quantum search algorithm, is an unstructured search strategy that enhances search performance in unsorted data. When compared to standard counterpart techniques, it results in a quadratic speedup. Grover's algorithm, in the context of cryptography, basically tackles the problem of function inversion. The approach might be used in a variety of symmetric-key cryptography brute-force attacks, including collision and pre-image attacks. For example, it could brute-force a 128-bit symmetric cryptographic key in roughly 2^{64} iterations, or a 256-bit key in roughly 2^{128} iterations.

The performance of a quantum computer is substantially superior to that of a regular computer. The security of various cryptographic primitives is jeopardized by the emergence of quantum computers. Hence, if a quantum computer with enough qubits is utilized, most asymmetric cryptography methods and protocols will be broken. On the other hand, if sufficiently high key sizes are used, most symmetric encryption techniques are now deemed quantum-safe. The same may be stated for the majority of hash functions as most of them stay quantum secure [56], given that it is required to create hashes of double the size [57].

Some existing cryptography is quantum-safe, although the majority of post-quantum algorithms are still in development. The NIST began the first step of standardizing numerous post-quantum algorithms in 2017. This standardization procedure examines 69 post-quantum algorithms [58]. A subset of 26 algorithms were picked from the original candidate algorithms

to proceed to the second phase of the procedure, where another group of algorithms was chosen to advance to the third stage of standardization [59]. Because the standardization process is not yet complete, the algorithms may be vulnerable to unforeseen flaws. During the process, the vulnerabilities are detected and the algorithms are refined, resulting in fewer uncovered issues. Nevertheless, the adoption of post-quantum algorithms at the moment is accompanied by some hardships. For instance, the majority of post-quantum algorithms employ keys of a greater size than those currently in use. Increased key size may make storing private keys on constrained devices, such as IoT, more complex, as well as impose a message overhead when conveying keys. Furthermore, the greater part of post-quantum algorithms require more complex computations. This computational complexity leads to longer computation times and increased energy usage.

As discussed earlier, the Signal Protocol consists of two fundamental components: the initial X3DH key exchange and the Double Ratchet for message exchange and the key update. The cryptographic primitives used in the protocol are: key exchanges, KDFs, signature schemes, and symmetric encryption. The protocol specification employs the specific cryptographic algorithms listed in table 5.2.

Table 5.2: Pre-Quantum Signal Protocol Algorithms

Cryptographic Primitives	Algorithm used
Key exchange	ECDH, with Curve25519
KDF	HKDF, with SHA-512
X3DH Signature Scheme	XEdDSA, with Curve25519
Double Ratchet Symmetric Encryption	AES-256 in CBC mode

Some of the protocol's components utilize algorithms which are considered quantum secure, or can be with a slight modification. First, the KDF component which is usually based on cryptographic hash functions. It is crucial to use quantum secure hash functions, which is not related to the factorization problem. The Double Ratchet specification recommends implementing HKDF on an HMAC with SHA-256 or SHA-512. A HKDF based upon a SHA-2 as its underlying hash function has a significant impact on Gorver's algorithm search cost [60]. So far, such HKDF is considered post-quantum secure.

Next, the AES symmetric encryption algorithm used for message encryption. A quantum computer, with its improved calculating capabilities, can degrade this algorithm. Although not to the level that Shor's algorithm undermines asymmetric systems. The algorithm's quantum cryptographic security can be boosted by increasing the key size.

Furthermore, Elliptic Curve Cryptography (ECC) algorithms are based on discrete logarithm problems for elliptic curves which are solved within a reasonable amount of time using Shor's algorithm. Therefore, Elliptic Curve Diffie-Hellmann (ECDH) is directly threatened by quantum computers and its operations are not post-quantum secure within the Signal protocol. Thus, each key exchange algorithm based on ECDH as well as the signature scheme based on Curve25519 or Curve448 Elliptic Curves (ECs), which were recommended by the protocol specification, must be substituted by alternative post-quantum algorithms. Currently, many of the post-quantum key exchange algorithms require interaction from communicating parties during the key exchange, i.e. synchronous algorithms. While such algorithms are not a

suitable substitute for use in X3DH, they can be used in a workaround for the double ratchet algorithm. However, they are not applicable as a substitute for X3DH as a non-interactive key exchange algorithm is essential.

Although no real-world post-quantum secure variant is yet realized, Duits [61] was the first to introduce a Key Encapsulation Mechanism (KEM) based variant of Signal. She has proposed approaches to create hybrid post-quantum and complete post-quantum variants of the Signal protocol. A hybrid scheme is one that combines post-quantum algorithms and non post-quantum algorithms. She defines the current period where classical computers are dominant, while quantum computers are scarce and still under development as the transitional period. For the transitional period, she believes that a hybrid protocol would be useful and sufficient against a passive quantum attack. The hypothesis being, if the post-quantum scheme is broken, the security of the non post-quantum schemes could be relied upon. The work presents four partially hybrid variants of the protocol, meaning that at least one ECDH key exchange is substituted. All the presented variants are not a perfect solution for a quantum secure Signal protocol as each lacks a security property in contrast to the pre-quantum variant. The partial hybrid variants could be used as building blocks and combined to create further variants. For the transition period, it is argued that the most straightforward method would be to include at least one post-quantum initial key exchange, next to the ECDH X3DH protocol. Namely, an extra post quantum One Time Pre-key (OTP) or IK key exchange. However, neither variant offer future secrecy in the post-quantum world, as well as other security properties. For a complete quantum secure Signal protocol, her analysis concluded that the following algorithms are the most optimal substitutes among the algorithms in the experiment: the Supersingular isogeny Diffie-Hellman (SIDH) algorithm, specifically the SIDH503 version, for X3DH; and a lattice-based KEM, specifically Kyber-512, for the double ratchet algorithm. The SIDH503 algorithm is considered a perfect substitute for ECDH, while kyper512 imposes less delay per message by 0.03 seconds. Although, Kyber-512 requires a small change to integrate it into Signal Protocol unlike SIDH503 that can be used out of the box.

However, Brendel et al. [62] discuss limitations of SIDH-based key exchanges and KEM-based algorithms. Their claim was that key exchanges based on SIDH are not an adequate replacement for X3DH as they are not secure due to their vulnerability to attacks against the protocol's key reuse. Although other SIDH proposals are more secure [63, 64, 65], their usability may be constrained due to their high cost, lack of scalability, or being inconclusive. On the other side, even though some of the KEM-based algorithms in the NIST process rely on the Fujisaki-Okamoto transform which provides safe key reuse for one party, they are inadequate to achieve asynchrony for an AKE as the encapsulating party cannot contribute non-ephemeral input and has to fully disclose the secret key behind their encapsulation. The authors present the concept of a split KEM to transfer the intended key-reusability of a DH-based protocol to a KEM-based flow. Their work intends to employ split KEM to transform the X3DH handshake into a quantum-secure KEM setting without additional message flows. However, due to the unsolved challenge of proving that known KEMs are secure under their notion of split KEM with key reuse on both sides of communicating parties. Their work concludes that further research is required to develop securely robust post-quantum solutions with the same level of versatility as Diffie–Hellman-based primitives.

Building upon the outcome of two previously mentioned works, Stadler et al. [66] describe a hybrid solution for a quantum secure Signal protocol by combining the original protocol with

a post-quantum variant.Their proposal employs signatures in its replacement for the X3DH ECDH operations to avoid the vulnerability of KEM algorithms against key reuse. However among other deviations from the original protocol, the most obvious impact was the loss of deniability. Nevertheless, their approach towards double ratchet was to alter the algorithm using a KEM. Their implementation of the approach did not change any of the symmetric cryptography algorithm used in the pre-quantum protocol. However, the ECDH algorithm was substituted by Kyber-786 and the signature algorithm was substituted by Dilithium 1280x1024.

6 Demo Implementation

This chapter presents our implementation of a post-compromise secure system to be utilized in an IoT setting. Python 3.8 is the language used for implementing our demo. The implementation follows the description of algorithms discussed in chapter 5. Notably, the X3DH implementation follows our formal model rather than the protocol specification.

The implementation is split between two main classes: a server and a client. A client implements both the X3DH and the double ratcheting algorithm as a post-key establishment message exchange algorithm. On the other hand, in our demo, the server has two roles. First, it acts as a CA for clients. Second, it stores and distributes prekey bundles during the X3DH phase, afterwards the server acts as a centralized relay node between clients.

6.1 Used Cryptography

The implementation depends solely on the *cryptography* package [7], specifically version 3.4.7, to provide the necessary cryptographic functions. Having all the required cryptographic functions and primitives in one library saved us the pain of incompatibility issues. Table 6.1 lists the cryptographic primitives used throughout the implementation.

Table 6.1: Used cryptographic standards and algorithms in demo implementation.

Cryptographic Primitive	Standard/Algorithm
Certificates	X.509 v3
Long-term signing/encryption keys	RSA
Key exchange	Curve25519
Symmetric encryption	Fernet (AES 128 in CBC mode)
KDF	HKDF
Hash function	SHA256

6.2 Server side

The server is the first entity to be started. Since the server doubles as a CA, it starts by generating its own self-signed certificate. The server's RSA key pair are used for both encryption and signing. Although a bad practice, clients use the public key to encrypt data to the server as well as verify signatures from the server. The self-signed certificate is saved

[7] https://cryptography.io/en/3.4.7/

to disk, mimicking a some sort of a key bootstrapping mechanism for clients. Next, the server opens a websocket and is ready to accept connections from clients. Clients will tend to obtain certificates signed from the CA, thus, for the sake of simulation, the CA accepts and signs all received CSR. The server keeps track of all connected clients, their registered keys, certificates, online status, and message count. Additionally, the server buffers messages for offline clients and forwards them back when they are available. Furthermore, the server handles requests from clients and responds with the appropriate messages. Table 6.2 describes the message types sent by the server.

Table 6.2: Server message types.

Message Type	Description
CP	Certification response. Contains a certificate for the clients long-term key that is signed by the server which acts as the CA.
GET_PEERS_resp	Response to the client's GET_PEERS request. Returns a dictionary that contains an array of names of clients connected to the server.
KEYBUNDLE	Response to the client's FETCH_KEYBUNDLE request. Returns to the requester an X3DH prekey bundle of the specifically requested peer.
ERROR	An error response for an invalid request.

6.3 Client side

A client implementation is composed of 2 sub-clients: a *X3DH_Client* and a *DoubleRatchet_Client*. At the beginning, the X3DH_Client is in control. It starts by loading the server certificate from file to utilize its key in further functionalities. Next, it generates the keys necessary for the client to carry out further: the long-term RSA signing key certified from the CA, its Curve25519 identity key for key exchange, as well as the remaining keys for the X3DH protocol. Each client generates all the necessary keys that enable them to play either role in the X3DH protocol, Alice or Bob. Reflecting on the X3DH protocol, Bob performs his registration phase where he uploads his X3DH keys. On the other hand, Alice queries the server for the list of available peers. After determining which party it wants to communicate with, assuming it is Bob, Alice fetches a prekey bundle for Bob from the server and computes the shared secret. Alice proceeds by sending the initial message to Bob. The initial message contains Bob's keys which were used to generate the shared secret, in addition to the initial message of the double ratchet algorithm encapsulated within the message's body and encrypted using the X3DH shared secret. At this moment, the DoubleRatchet_Client comes into action to generate the initial double ratchet message, and it takes over from this point over. The DoubleRatchet_Client is responsible for maintaining and advancing the double ratchet chains, as well as encrypting and decrypting double ratchet messages.

The double ratchet algorithm description states that a party, Bob, can send several messages from the same sending chain, and for Alice to decrypt those messages, she has to create the equivalent receiving chain. For the receiving chain to be created, Alice needs to create a new key pair to perform the necessary operations, and upon the creation of a new key pair, the new

public key is shared with Bob to create a new receiving chain. The same happens on Bob's side and so on. Assuming a new public key is shared with every message, no party would have the chance to send more than one message per chain. Therefore, our implementation only sends the newly generated on command, rather than automatically, to allow the client more freedom to decide when to update their sending chain. Table 6.3 lists all message types supported by the client.

Table 6.3: Client message types.

Message Type	Description
CR	Certification request. A client sends a CSR request to the server which acts as a CA to obtain a digital certificate for its long-term identity key.
REGISTER_TO_SERVER	Corresponds to X3DH registration phase. The clients uploads to the server its X3DH keys.
GET_PEERS	Query of other clients connected to the server.
FETCH_KEYBUNDLE	Request a X3DH prekey bundle of one of the available peers.
InitialMsg	Sends the initial X3DH message from one client to the other. The body of this message encapsulates a DR_INITIAL_MSG encrypted with the X3DH shared secret.
DR_INITIAL_MSG	Transports the initiating party's public DH key for the first double ratchet mesasge.
UPDATE_SPK	Updates a party's SPK stored at the server.
UPDATE_OPK	Updates a party's OPKs stored at the server.
ENC_CLIENT_MSG	An encrypted double ratchet message from one party to the other
OFFLINE_NOTIFICATION	Sent to the server to simulate the client going offline.
ONLINE_NOTIFICATION	Sent to the server to simulate the client going back online.

The client offers a command-line interface that allows for user input to control the program through a set of implemented commands. Below we list the implemented commands[8].

- *csr:* sends a certificate signing request to the server.
- *reg*2*server:* registers client to server.
- *updateSPK:* updates client's SPK at server.
- *updateOPKs:* updates clientś OTPs at server.
- *goOnline:* notifies the server a client is online.
- *goOffline:* notifies the server a client is offline.

[8]Commas (',') in commands are part of command syntax and must be included. ¡X¿ must be replaced by the appropriate parameter without inclusion of the angle brackets.

- *peers:* gets names of peers registered at the server.
- $fetchKeyBundle, < peerName >:$ fetches key bundle of the specified peer from server.
- $init, < peerName >:$ sends the initial message to the specified peer.
- $send, < peerName >, < textToBeEncrypted >:$ sends an encrypted message to the specified peer without updating a new sending chain root.
- $sendW/NewSendChain, < peerName >, < textToBeEncrypted >:$ sends an encrypted message to the specified peer and update sending chain root.

6.4 Simulation Execution

First, the server must be running and ready to accept clients to run the demo. Then, multiple clients can connect to the server and communicate with one another simultaneously. A client requires three arguments to start: a unique name, a boolean value to run in verbose mode or nor, and a boolean to allow user input from the command line or not. The specified arguments should be input in the mentioned order and separated by a blank. The last argument should only be set to *True* in case of executing an automated script. A client connects automatically to the server without any required user input. Nevertheless, depending on the role, a client must execute a set of commands in order.

Either role has to start by running *csr*. Bob has to execute *reg2server* to be able to receive a message from Alice. Bob can wait for communication or can act as Alice for other registered users. Alice has to execute *fetchKeyBundle*, then *init*, before executing *send* to send messages to Bob. Afterwards, Alice and Bob can exchange messages using

6.5 Room for Improvement

In this section we mention limitations of the implementation and improvement suggestions.

- *Use of RSA.* RSA is the asymmetric algorithm used for the client signing key and the server's identity key. ECC uses substantially shorter keys than RSA keys while maintaining the same security level. For example, a 256-bit symmetric key requires more than 15,000-bit RSA to secure it; on the other hand, ECC employs an asymmetric key size of just 512 bits to provide the same level of security. Therefore, the replacement of RSA keys with ECC keys is more suitable for a constrained environment as it will lead to a reduced performance complexity cost and, accordingly, less power consumption and a more compact implementation design.
- *Multiple usage for the same key.* As mentioned, the server uses the same long-term key pair for signing and encryption. Although feasible, it is a dreadful cryptography practice. Therefore, The server should possess two certificates for two different keys, encryption and signing keys.
- *Use of websockets.* Websockets are considered more lightweight than HTTP, and they provide low-latency communication as they maintain a single TCP channel open between parties. However, the imposed communication overhead is considered high, and

websockets are regarded as ill-fitting for the IoT setting. Therefore, CoAP and MQTT are well-suited alternatives for the communication protocol.

- *Lack of feedback.* Some of the client messages implemented lack response feedback from the server, e.g., the UPDATE_SPK message. Implementing a server response for messages would give clients a more comprehensive idea of whether their requests were successful or failed for a specific reason.
- *Handling out-of-order messages.* Since it is highly anticipated in the field of IoT for messages to be delayed or dropped due to the reduced reliability of the utilized protocol, e.g., CoAP. Thus, implementing the handling of out-of-order messages, as presented by the double ratchet algorithm, will be of added value.

7 Discussion

Bootstrapping protocols may not be directly adequate for all IoT constrained classes due to underlying PKI protocols. However, with employing a lightweight PKI like [67], the protocols may become more suitable.

8 Conclusion and Future Work

Summarize the thesis and provide a outlook on future work.

Bibliography

[1] Stefan Marksteiner et al. "An overview of wireless IoT protocol security in the smart home domain". In: *2017 Internet of Things Business Models, Users, and Networks*. IEEE. 2017, pp. 1–8.

[2] International Telecommunication Union. *X.509 : Information technology - Open Systems Interconnection - The Directory: Public-key and attribute certificate frameworks*. 2019. URL: https://www.itu.int/rec/T-REC-X.509-201910-I/en.

[3] Petra Wohlmacher. "Digital certificates: a survey of revocation methods". In: *Proceedings of the 2000 ACM workshops on Multimedia*. 2000, pp. 111–114.

[4] *Bootstrapping*. URL: https://en.wikipedia.org/wiki/Bootstrapping (visited on 08/08/2021).

[5] Kent Watsen et al. *A Voucher Artifact for Bootstrapping Protocols*. RFC 8366. May 2018. DOI: 10.17487/RFC8366. URL: https://www.rfc-editor.org/info/rfc8366.

[6] Russ Housley. *Cryptographic Message Syntax (CMS)*. RFC 5652. Sept. 2009. DOI: 10.17487/RFC5652. URL: https://www.rfc-editor.org/info/rfc5652.

[7] Martin Björklund. *The YANG 1.1 Data Modeling Language*. RFC 7950. Aug. 2016. DOI: 10.17487/RFC7950. URL: https://www.rfc-editor.org/info/rfc7950.

[8] Max Pritikin, Peter E. Yee, and Dan Harkins. *Enrollment over Secure Transport*. RFC 7030. Oct. 2013. DOI: 10.17487/RFC7030. URL: https://rfc-editor.org/rfc/rfc7030.txt.

[9] Tero Mononen et al. *Internet X.509 Public Key Infrastructure Certificate Management Protocol (CMP)*. RFC 4210. Sept. 2005. DOI: 10.17487/RFC4210. URL: https://www.rfc-editor.org/info/rfc4210.

[10] Peter Gutmann. *Simple Certificate Enrolment Protocol*. RFC 8894. Sept. 2020. DOI: 10.17487/RFC8894. URL: https://www.rfc-editor.org/info/rfc8894.

[11] Michael Myers and Jim Schaad. *Certificate Management over CMS (CMC)*. RFC 5272. June 2008. DOI: 10.17487/RFC5272. URL: https://www.rfc-editor.org/info/rfc5272.

[12] Sebastian Mödersheim and Luca Vigano. "The open-source fixed-point model checker for symbolic analysis of security protocols". In: *Foundations of Security Analysis and Design V*. Springer, 2009, pp. 166–194.

[13] Luca Viganò. "Automated Security Protocol Analysis With the AVISPA Tool". In: *Electronic Notes in Theoretical Computer Science* 155 (2006). Proceedings of the 21st Annual Conference on Mathematical Foundations of Programming Semantics (MFPS XXI), pp. 61–86. ISSN: 1571-0661. DOI: https://doi.org/10.1016/j.entcs.2005.11.052. URL: https://www.sciencedirect.com/science/article/pii/S1571066106001897.

[14] Sebastian Mödersheim. "Algebraic properties in Alice and Bob notation". In: *2009 International Conference on Availability, Reliability and Security*. IEEE. 2009, pp. 433–440.

[15] Danny Dolev and Andrew Yao. "On the security of public key protocols". In: *IEEE Transactions on information theory* 29.2 (1983), pp. 198–208.

[16] Lily Chen et al. "Recommendation for key derivation using pseudorandom functions". In: *NIST special publication* 800 (2008), p. 108.

[17] Peter Gutmann. *Key Management through Key Continuity (KCM)*. Internet-Draft draft-gutmann-keycont-01. Work in Progress. Internet Engineering Task Force, Sept. 2008. 20 pp. URL: https://datatracker.ietf.org/doc/html/draft-gutmann-keycont-01.

[18] Nikita Borisov, Ian Goldberg, and Eric Brewer. "Off-the-record communication, or, why not to use PGP". In: *Proceedings of the 2004 ACM workshop on Privacy in the electronic society*. 2004, pp. 77–84.

[19] 2022. URL: https://otr.cypherpunks.ca/Protocol-v3-4.0.0.html.

[20] Fabrice Boudot, Berry Schoenmakers, and Jacques Traoré. "A fair and efficient solution to the socialist millionaires' problem". In: *Discrete Applied Mathematics* 111.1 (2001). Coding and Cryptology, pp. 23–36. ISSN: 0166-218X. DOI: https://doi.org/10.1016/S0166-218X(00)00342-5. URL: https://www.sciencedirect.com/science/article/pii/S0166218X00003425.

[21] Nik Unger et al. "SoK: secure messaging". In: *2015 IEEE Symposium on Security and Privacy*. IEEE. 2015, pp. 232–249.

[22] Adrian Perrig and JD Tygar. "TESLA broadcast authentication". In: *Secure Broadcast Communication*. Springer, 2003, pp. 29–53.

[23] Manisha Malik, Maitreyee Dutta, and Jorge Granjal. "A Survey of Key Bootstrapping Protocols Based on Public Key Cryptography in the Internet of Things". In: *IEEE Access* 7 (2019), pp. 27443–27464. DOI: 10.1109/ACCESS.2019.2900957.

[24] *2017 Passenger Summary - Annual Traffic Data*. Jan. 2019. URL: https://aci.aero/data-centre/annual-traffic-data/passengers/2017-passenger-summary-annual-traffic-data/.

[25] Boeing. *Commercial Market Outlet 2020 – 2039*. Tech. rep. 2020. URL: https://www.boeing.com/commercial/market/commercial-market-outlook/.

[26] ReadID. *Which countries have ePassports?* 2020. URL: https://www.readid.com/blog/countries-epassports (visited on 08/07/2021).

[27] Lisa Angiolelli-meyer, Jeremy Stokes, and Peter Smallridge. *Automated Border Control Implementation Guide*. Tech. rep. December. IATA, 2015. URL: https://aci.aero/Media/183a7715-da8f-4664-a82c-36f8b961ef28/FiQQnQ/About%20ACI/Priorities/Facilitation/ABC_Implementation_Guide_2nd_Edition.pdf.

[28] Ruggero Donida Labati et al. "Biometric recognition in automated border control: a survey". In: *ACM Computing Surveys (CSUR)* 49.2 (2016), pp. 1–39.

[29] "IEEE Standard for Local and metropolitan area networks - Secure Device Identity". In: *IEEE Std 802.1AR-2009* (2009), pp. 1–77. DOI: 10.1109/IEEESTD.2009.5367679.

[30] Mohit Sethi, Behcet Sarikaya, and Dan Garcia-Carrillo. *Secure IoT Bootstrapping: A Survey*. Internet-Draft draft-irtf-t2trg-secure-bootstrapping-00. Work in Progress. Internet Engineering Task Force, Apr. 2021. 24 pp. URL: https://datatracker.ietf.org/doc/html/draft-irtf-t2trg-secure-bootstrapping-00.

[31] Paul Wouters et al. *Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)*. RFC 7250. June 2014. DOI: 10.17487/RFC7250. URL: https://rfc-editor.org/rfc/rfc7250.txt.

[32] 3GPP. *Generic Authentication Architecture (GAA); Generic Bootstrapping Architecture (GBA)*. Technical Specification (TS) 33.220. Release 14. 3rd Generation Partnership Project (3GPP), Dec. 2016. URL: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=2280.

[33] Olaf Bergmann, Stefanie Gerdes, and Carsten Bormann. "Simple keys for simple smart objects". In: *Workshop on Smart Object Security*. Vol. 258. Citeseer. 2012.

[34] Max Pritikin et al. *Bootstrapping Remote Secure Key Infrastructure (BRSKI)*. RFC 8995. May 2021. DOI: 10.17487/RFC8995. URL: https://rfc-editor.org/rfc/rfc8995.txt.

[35] Kent Watsen, Mikael Abrahamsson, and Ian Farrer. *Secure Zero Touch Provisioning (SZTP)*. RFC 8572. Apr. 2019. DOI: 10.17487/RFC8572. URL: https://rfc-editor.org/rfc/rfc8572.txt.

[36] Carsten Bormann, Brian E. Carpenter, and Bing Liu. *GeneRic Autonomic Signaling Protocol (GRASP)*. RFC 8990. May 2021. DOI: 10.17487/RFC8990. URL: https://rfc-editor.org/rfc/rfc8990.txt.

[37] Jeffrey O Kephart and David M Chess. "The vision of autonomic computing". In: *Computer* 36.1 (2003), pp. 41–50.

[38] Stuart Cheshire and Marc Krochmal. *DNS-Based Service Discovery*. RFC 6763. Feb. 2013. DOI: 10.17487/RFC6763. URL: https://rfc-editor.org/rfc/rfc6763.txt.

[39] Stuart Cheshire and Marc Krochmal. *Multicast DNS*. RFC 6762. Feb. 2013. DOI: 10.17487/RFC6762. URL: https://rfc-editor.org/rfc/rfc6762.txt.

[40] Ralph Droms. *Dynamic Host Configuration Protocol*. RFC 2131. Mar. 1997. DOI: 10.17487/RFC2131. URL: https://rfc-editor.org/rfc/rfc2131.txt.

[41] Chris Newman and Graham Klyne. *Date and Time on the Internet: Timestamps*. RFC 3339. July 2002. DOI: 10.17487/RFC3339. URL: https://rfc-editor.org/rfc/rfc3339.txt.

[42] Paul E. Hoffman and Jakob Schlyter. *The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA*. RFC 6698. Aug. 2012. DOI: 10.17487/RFC6698. URL: https://rfc-editor.org/rfc/rfc6698.txt.

[43] 2013. URL: https://signal.org/blog/advanced-ratcheting/.

[44] Katriel Cohn-Gordon, Cas Cremers, and Luke Garratt. "On post-compromise security". In: *2016 IEEE 29th Computer Security Foundations Symposium (CSF)*. IEEE. 2016, pp. 164–178.

[45] Joël Alwen, Sandro Coretti, and Yevgeniy Dodis. *The Double Ratchet: Security Notions, Proofs, and Modularization for the Signal Protocol*. 2020. URL: https://eprint.iacr.org/2018/1037.pdf.

[46] Moxie Marlinspike and Trevor Perrin. "The x3dh key agreement protocol". In: *Open Whisper Systems* (2016).

[47] Trevor Perrin and Moxie Marlinspike. *The Double Ratchet Algorithm*. Nov. 2016. URL: https://www.signal.org/docs/specifications/doubleratchet/doubleratchet.pdf.

[48] Alfred J Menezes, Paul C Van Oorschot, and Scott A Vanstone. *Handbook of applied cryptography*. CRC press, 2018.

[49] Katriel Cohn-Gordon et al. "A formal security analysis of the signal messaging protocol". In: *Journal of Cryptology* 33.4 (2020), pp. 1914–1983.

[50] Bertram Poettering and Paul Rösler. "Asynchronous ratcheted key exchange". In: *Cryptology ePrint Archive* (2018).

[51] Mihir Bellare et al. "Ratcheted encryption and key exchange: The security of messaging". In: *Annual International Cryptology Conference*. Springer. 2017, pp. 619–650.

[52] Joseph Jaeger and Igors Stepanovs. "Optimal channel security against fine-grained state compromise: The safety of messaging". In: *Annual International Cryptology Conference*. Springer. 2018, pp. 33–62.

[53] F Betül Durak and Serge Vaudenay. "Bidirectional asynchronous ratcheted key agreement with linear complexity". In: *International Workshop on Security*. Springer. 2019, pp. 343–362.

[54] Peter W Shor. "Algorithms for quantum computation: discrete logarithms and factoring". In: *Proceedings 35th annual symposium on foundations of computer science*. Ieee. 1994, pp. 124–134.

[55] Lov K. Grover. "A Fast Quantum Mechanical Algorithm for Database Search". In: *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*. STOC '96. Philadelphia, Pennsylvania, USA: Association for Computing Machinery, 1996, pp. 212–219. ISBN: 0897917855. DOI: 10.1145/237814.237866. URL: https://doi.org/10.1145/237814.237866.

[56] Daniel J Bernstein. "Cost analysis of hash collisions: Will quantum computers make SHARCS obsolete?" In: *SHARCS* 9 (2009), p. 105.

[57] Gilles Brassard, Peter Høyer, and Alain Tapp. "Quantum cryptanalysis of hash and claw-free functions". In: *Latin American Symposium on Theoretical Informatics*. Springer. 1998, pp. 163–169.

[58] Lily Chen et al. *Report on post-quantum cryptography*. Vol. 12. US Department of Commerce, National Institute of Standards and Technology ..., 2016.

[59] Gorjan Alagic et al. "Status report on the second round of the NIST post-quantum cryptography standardization process". In: *US Department of Commerce, NIST* (2020).

[60] Xenia Bogomolec, John Gregory Underhill, and Stiepan Aurélien Kovac. "Towards Post-Quantum Secure Symmetric Cryptography: a Mathematical Perspective". In: *Cryptology ePrint Archive* (2019).

[61] Ines Duits. "The post-quantum signal protocol: secure chat in a quantum world". MA thesis. University of Twente, 2019.

[62] Jacqueline Brendel et al. "Towards post-quantum security for signal's X3DH handshake". In: *International Conference on Selected Areas in Cryptography*. Springer. 2020, pp. 404–430.

[63] Reza Azarderakhsh, David Jao, and Christopher Leonardi. "Post-quantum static-static key agreement using multiple protocol instances". In: *International Conference on Selected Areas in Cryptography*. Springer. 2017, pp. 45–63.

[64] Andrea Basso et al. "On adaptive attacks against Jao-Urbanik's isogeny-based protocol". In: *International Conference on Cryptology in Africa*. Springer. 2020, pp. 195–213.

[65] Dan Boneh et al. "Multiparty non-interactive key exchange and more from isogenies on elliptic curves". In: *Journal of Mathematical Cryptology* 14.1 (2020), pp. 5–14.

[66] Sara Stadler et al. "Hybrid Signal protocol for post-quantum email encryption". In: *Cryptology ePrint Archive* (2021).

[67] Joel Höglund et al. "PKI4IoT: Towards public key infrastructure for the Internet of Things". In: *Computers & Security* 89 (2020), p. 101658.

[68] Kent Watsen, Russ Housley, and Sean Turner. *Conveying a Certificate Signing Request (CSR) in a Secure Zero Touch Provisioning (SZTP) Bootstrapping Request*. Internet-Draft draft-ietf-netconf-sztp-csr-12. Work in Progress. Internet Engineering Task Force, Dec. 2021. 36 pp. URL: `https://datatracker.ietf.org/doc/html/draft-ietf-netconf-sztp-csr-12`.

A BRSKI vs. SZTP

A.1 Terminology Comparison

BRSKI		SZTP	
Domain	The set of entities, belonging to the owner of the device, that share a common local trust anchor. This includes the proxy, registrar, Domain Certificate Authority, etc..	Owner	• The person or organization that purchased or otherwise owns a device. • The term is used through out the RFC to represent the the sub-entities the owner might be using within their domain, like registrar, proxy, or CA, without explicitly referring to them.
Registrar	A representative of the domain that is configured, to decide whether a new device is allowed to join the domain.		
Proxy	• A domain entity that helps the pledge join the domain. A join proxy facilitates communication for devices that find themselves in an environment where they are not provided connectivity until after they are validated as members of the domain. • The pledge is unaware that they are communicating with a proxy rather than directly with a registrar.		

BRSKI		SZTP	
Pledge	The unconfigured device to be bootstrapped	Device	The unconfigured device to be bootstrapped
Manufacturer	The entity that created the device	Manufacturer	• The entity that created the device • Through out the RFC, the term generally covers the services provided by the manufacturer without explicitly referring to them, like, MASA.
Voucher	RFC 8366	Ownership voucher	RFC 8366

A.2 Protocol Comparison

	BRSKI	**SZTP**
Standardization	RFC 8995	RFC 8572
Related RFCs	Voucher artifact [5]	
Aim	Results in the pledge storing a root certificate sufficient for verifying the registrar identity. The installed trust anchor can be used for later certificate enrollment protocols (typically, EST)	Without any manual interference beyond physical placement, securely update the boot image, commit an initial configuration, and execute arbitrary scripts to address auxiliary needs.
Communication channels covered by protocol	Pledge ↔ Registrar ↔ MASA	Device ↔ Owner (↔ MASA: not protocol inherent)
Remote/Local bootstrapping sources support	remote (/.wellknown/...) and local	
Device bootstrap sources	Domain Registrar	Removable storage, DNS server, DHCP server, or Bootstrap server
Protocol initiator	Pledge (Device)	
Functionality support (M): Mandatory (O): Optional	• (M) Pledge-Registrar Discovery • (M) Pledge: polling • (M) if EST following BRSKI: CSR attributes retrieval request • (M) MASA: voucher issuance • (M) MASA: voucher renewal • (M) MASA voucher audit log • (O) Manufacturer: Ownership tracking	• (M) Device: polling • (M) if Bootstrap server is used: provide redirect information and/or onboarding information • (M) DHCP/DNS server: can provide redirect information only due to technical limitations • (M) MASA: voucher issuance

	BRSKI	SZTP
Device initial state	• IDevID • Manufacturer installed trust anchor(s) associated with the manufacturer's MASA	• IDevID • Optional: TLS client cert & related intermediate certs • Trust anchors to validate ownership voucher (signed by manufacturer) • List of well-known bootstrap servers • Trust anchors to authenticate configured well-known bootstrap servers
Discovery of bootstrap sources	Yes, mDNS/ GRASP	Only through redirections from device supported bootstrap sources
Device authentication	IDevID	IDevID
Device authorization	• A specific device (serial number) from a specific vendor • A specific device type or a specific vendor	based on device's serial number
Bootstrap source authentication	Intially, Provisional TLS	Initially, Provisional TLS if no TA available
Enrollment protocol integration	(R) EST	Conveying CSR in SZTP draft [68]
Bootstrap data	• Voucher • LDevID	• Redirect information (auxilary) • Onboarding information: boot image, configuration, post-config scripts, voucher, owner certificate

	BRSKI	SZTP
Bootstrapping data protection	• Voucher signed by manufacturer • LDevID signed by domain CA • No additional encryption (only TLS encryption in Transit)	• Trusted channel: may be signed and/or encrypted • Untrusted channel: signed and may be encryped
Owner voucher-request time	• Nonced: in-band • Nonceless: Out-of-band	• nonced: in-band • Nonceless: owner-manufacturer enrollment phase
Acceptance of device by Domain	Checking voucher and its presence in the MASA audit-log	Checking the voucher
Determining MASA to contact	URI in IDevID or manual confiuration of registrar	Out of scope
Progress reports	Yes, voucher status telemetry	Yes, only to trusted servers
Timeliness	• Nonceless vouchers: expiry time • Nonced vouchers: nonces (and expiry time)	
Revocation checks	• No revocation for vouchers • certificate revocation checks only, depending on pledge capabilities	
Ownership transfer	Yes, by voucher issuance	• Out of scope • (Owner $\leftrightarrow$ MASA communication is not inherent to the protocol, but ownership transfer is possible through new vouchers by the MASA)

	BRSKI	**SZTP**
Updatable Manufacturer Trust Anchors (before bootstrap process)	out-of-scope	Out of scope (through a verifiable process, such as a software upgrade using signed software images)
Transport protocol	HTTP (or CoAP) / TLS1.2+	HTTP/TLS
Required Cryptographic algorithms	None	None
Domain specific configuration provisioning to device	out of scope	yes

B Certificate Enrollment Protocols Comparison