Summer 7-2020

Leveraging Smart Contracts for Asynchronous Group Key Agreement in Internet of Things

Victor Youdom Kemmoe
Kennesaw State University

Junggab Son

Leveraging Smart Contracts for Asynchronous Group Key Agreement in Internet of Things

A Thesis Presented to

The Faculty of the Computer Science Department

by

Victor Youdom Kemmoe

In Partial Fulfillment

of Requirements for the Degree

Master of Science, Computer Science

Kennesaw State University

July 2020

Leveraging Smart Contracts for Asynchronous Group Key Agreement in Internet of Things

Approved:

DocuSigned by:
Junggab Son
A197BD0D60714A7...
July 20, 2020

Dr. Junggab Son – Advisor

DocuSigned by:
Coskun Cetinkaya
028A87E4965A4EE...
July 20, 2020

Dr. Coskun Cetinkaya – Department Chair

DocuSigned by:
Jon Preston
FD9E8EC07752427...
July 20, 2020

Dr. John Preston – Dean

In presenting this thesis as a partial fulfillment of the requirements for an advanced degree from Kennesaw State University, I agree that the university library shall make it available for inspection and circulation in accordance with its regulations governing materials of this type. I agree that permission to copy from, or to publish, this thesis may be granted by the professor under whose direction it was written, or, in his absence, by the dean of the appropriate school when such copying or publication is solely for scholarly purposes and does not involve potential financial gain. It is understood that any copying from or publication of, this thesis which involves potential financial gain will not be allowed without written permission.

Victor Youdom Kemmoe

Notice To Borrowers

Unpublished theses deposited in the Library of Kennesaw State University must be used only in accordance with the stipulations prescribed by the author in the preceding statement.

The author of this thesis is:

Victor Youdom Kemmoe

950 Hudson Road SE, Apt 307,

Marietta, GA, 30060

The director of this thesis is:

Dr. Junggab Son

1100 South Marietta Pkwy SE,

Marietta, GA, 30060

Users of this thesis not regularly enrolled as students at Kennesaw State University are required to attest acceptance of the preceding stipulations by signing below. Libraries borrowing this thesis for the use of their patrons are required to see that each user records here the information requested.

Leveraging Smart Contracts for Asynchronous Group Key Agreement in Internet of Things

An Abstract of

A Thesis Presented to

The Faculty of the Computer Science Department

by

Victor Youdom Kemmoe

Bachelor of Science, Kennesaw State University, 2018

In Partial Fulfillment

of Requirements for the Degree

Master of Science, Computer Science

Kennesaw State University

July 2020

Abstract

Group Key Agreement (GKA) mechanism plays a crucial role in the realization of various secure applications in various networks such as, but not limited to, sensor networks, Internet of Things (IoT), vehicular networks, social networks, and so on. To be suitable for IoT, GKA must satisfy several critical requirements. First, a GKA mechanism must be robust against a compromised device attack and satisfy essential secrecy definitions without the existence of a Trusted Third Party (TTP). TTP is often used by IoT devices in the establishment of ad hoc networks and usually, these devices are resource-constrained. Second, the GKA mechanism must be capable of distributing session keys successfully even with offline devices. Third, GKA must reduce the burden of heavy cryptographic computations for IoT devices. Based on these observations, in this paper, we propose a new GKA scheme that satisfies all the aforementioned requirements. The proposed scheme leverages smart contracts to alleviate the computational and storage overheads on the IoT devices induced by cryptographic functions. It also brings the advantage of asynchronism such that offline devices will be able to compute the group key once they are online since the essential information for obtaining the group key is stored inside the blockchain. We implement and test the proposed scheme on an Ethereum test network. The obtained results show that it consumes 5,264,150 gas to create a group, 994,178 gas to add a new member, and 798,431 gas to update a group key when the group has 20 members.

Leveraging Smart Contracts for Asynchronous Group Key Agreement in Internet of Things

A Thesis Presented to

The Faculty of the Computer Science Department

by

Victor Youdom Kemmoe

In Partial Fulfillment

of Requirements for the Degree

Master of Science, Computer Science

Advisor: Dr. Junggab Son

Kennesaw State University

July 2020

Acknowledgment

This thesis will not have been possible without the guidance of my advisor Dr. Junggab Son. Despite having bare-bone knowledge in the area of cryptography, Dr. Son accepted to take me in. His office's door was always open and he never belittled my ideas.

I would like to thank my friends from IIS laboratory. They made the working hours more enjoyable and helped me improve myself overall.

In addition, I would like to thank my friends and my family for their support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

IoT (Internet of Things) devices have become ubiquitous in our daily life (Kangath, 2018), and the data they are dealing with are more sensitive than ever. For instance, we have smartwatches that monitor our hearthbeat, smart vehicles that keep track of our location, and so on. These devices need to communicate and exchange data with each other. To do so, they can engage in group communications over computer networks, such as ad-hoc networks. An instance of such a group communication is a fleet of smart vehicles broadcasting their positions to each other. However, if those communications happen over insecure networks, adversaries can eavesdrop on packets exchanged and obtain copies of those data. To prevent such attacks, group members can use a robust GKA (Group Key Agreement) to devise a session key that will secure their communication channels prior to exchanging messages.

From the definition of NIST (Barker, 2015), a *key agreement* is a key establishment procedure in which two or more participants contribute information to compute a common session key so no participants can obtain that key without the share of others. Generally, a key agreement is considered a GKA if it can support more than two participants. The first known key agreement is Diffie-Hellman key agreement (Diffie & Hellman, 1976). For Alice and Bob to obtain a common session key, they proceed as follows:

1. Both agree on a group G of generator g and of prime order q
2. Alice randomly generates a, computes $u = g^a \bmod q$, and sends u to Bob
3. Bob randomly generates b, computes $v = g^b \bmod q$, and sends v to Alice
4. Both compute the session key $ssk = g^{ab} \bmod q = u^b = v^a$

Nevertheless, this key agreement only supports two parties. Multiple works have proposed group key agreement schemes (Kim, Perrig, & Tsudik, 2004b; Choi, Hwang, & Lee, 2004; Arifi, Gardeshi, & Farash, 2012; Zhang, Wu, Domingo-Ferrer, Qin, & Dong, 2015), but these schemes are based on assumptions that do not hold in IoT environments. For instance, they assume that all group members will be online during the execution of a GKA, whereas IoT systems often employ power management mechanisms that can put some devices in hibernation mode to maximize the lifetime of networks composed of battery-powered devices (Kim et al., 2017; Rahimi & Chrysostomou, 2019), hence, in that case some group members may not be online during the execution of the GKA. Therefore, in such environments, group members should still be able to devise a common key after the execution of a GKA even if some devices are offline. In addition, IoT devices are typically resource-constrained, hence having them execute all cryptographic operations may reduce their lifetimes. Therefore, a GKA scheme could delegate cryptographic operations to reduce their burden. Furthermore, the fact that IoT devices are resource-constrained make them an easy target for adversaries (Meneghello, Calore, Zucchetto, Polese, & Zanella, 2019). Thus, a GKA scheme for IoT should provide post-compromise security, i.e., the security of a communication session between group members should be preserved even if one of the group members was compromised (Cohn-Gordon, Cremers, & Garratt, 2016).

Generally, GKA schemes presume the existence of a TTP (Trusted Third Party), such as a server that will store the public keys of group members and their identities, or that will perform some precomputations on information shared by group members. Meanwhile, a TTP can be regarded as a single point of failure. For example, if a GKA uses a TTP to

perform some precomputations on shared information and that TTP is brought down, then the GKA execution will fail. However, though blockchain was first used in the development of a cryptocurrency system (Nakamoto, 2008), its specifications, such as the immutability of data stored on it, position it as a suitable candidate to replace a TTP. In addition, with smart contracts, users can deploy executable codes on the blockchain (Buterin, 2014). If a user sends a transaction to the blockchain that calls one of these codes, it will trigger the code's execution. *Hence, blockchain can provide an always-online trusted execution environment*.

Our Contributions. In this work, we proposed an asynchronous GKA based on smart contract. Along the way, we define blockchain and smart contract functionalities necessary for its establishment. Also, we define the different security requirements needed and prove that, indeed, our proposed GKA fulfills those security requirements. Our proposed GKA uses a smart contract to host the different key materials and delegate part of the computations. It also supports the addition and removal of members. Simulation results on the Ethereum's test network show that it consumes 5,264,150 gas to create a group, 994,178 gas to add a new member, and 798,431 gas to update a group key, when the group has 20 members.

Outline of this work. Chapter 2 provides an overview of related works, Chapter 3 provides insights about different notions that are necessary for the establishment of our work, Chapter 4 presents our proposed solution, and finally, Chapter 5 provides a conclusion and future works.

Chapter 2

Related Works

The existing works in the literature focus on GKA schemes that reduce the computational burden of group members by delegating a part of the process to a TTP (Veltri, Cirani, Busanelli, & Ferrari, 2013; Islam et al., 2018; Zhang et al., 2018). In these approaches, each group member sends key materials to the TTP. Then, the TTP computes and broadcasts a pre-group key to group members which will be used to derive the final group key. However, all group members need to be online and send their contributions during a given time-frame. In addition, these schemes do not provide post-compromised security and suffer from the drawbacks of using a TTP.

In GKA, the main role of a TTP is to provide PKI (Public Key Infrastructure) related operations. PKIs are used to store and manage public encryption keys used by nodes in a network. To mitigate the issues of traditional PKIs, e.g., high centralization, decentralized PKIs based on blockchain have been proposed (Yao & Wang, 2018; Hu, Xiong, Huang, & Bao, 2018; Al-Bassam, 2017; Singla & Bertino, 2018). Since blockchain is immutable, blockchain-based solutions improve the integrity of public keys. However, using decentralized PKIs do not completely shield GKAs, and thus TTPs are still required to perform some operations such as pre-key computation. To use blockchain in conjunction with GKA, Schindler *et al.* proposed a distributed key generation system that leverages Ethereum's

smart contract for communication (Schindler, Judmayer, Stifter, & Weippl, 2019). In their scheme, TTP is no longer needed to handle keys and some operations are executed on Ethereum through smart contracts for devices' efficiency. However, all group members must be online during the initial operation, and a subset of group members must cooperate to obtain the secret key. This is not practical in an environment where some members have an intermittent connection.

Most importantly, none of the previously cited works offer post-compromise security and asynchronism concurrently. Some of the existing schemes provide those requirements for large groups (Cohn-Gordon, Cremers, Garratt, Millican, & Milner, 2018; Barnes et al., 2020). However, both rely on tree-based Diffie-Hellman construction. Basically, for four nodes Alice, Bob, Carl and David with private-public key $(a, g^a), (b, g^b), (c, g^c), (d, g^d)$, respectively, the group key is computed as follows:

- Alice and Bob combine their keys to compute $g^{(ab)}$
- Carl and David combine their keys to compute $g^{(cd)}$
- the group is computed by combining $g^{(ab)}$ and $g^{(cd)}$: $ssk = g^{g^{(ab)}g^{(cd)}}$

With such a construction, it is difficult to delegate part of the computation unless we rely on a TTP. Hence, they are impractical for resource-constrained IoT devices.

Chapter 3

Preliminaries

In this chapter, we introduce the background knowledge necessary for the establishment of this work. Notably, we cover Elliptic curve, Blockchain, and Smart Contract.

3.1 Notations

We use the following notations:

$\mathbb{Z}_n^*$	Set of integers $\bmod n$ excluding zero
$\{0,1\}^\ell$	Set of all binary strings of fixed length ℓ
$\{0,1\}^*$	Set of all binary strings of arbitrary length
$a \mid\mid b$	Concatenation of the string a and b
$\mid G \mid$	The number of elements in a set G
$a \in_R \mathbb{G}$	The element a is sampled uniformly and at random from the set $\mathbb{G}$
(k,K)	Given a private-public key pair (k,K), k represents the private key, and K represents the public key
$(P.x)$	Given a point P on an elliptic curve, $P.x$ represents the x-coordinate
$Pr(A)$	Probability of event A

3.2 Elliptic curve

The materials covered in this section is based on (Stallings, 2017; Washington, 2008).

An elliptic curve is a function defined by an equation of the form:

$$y^2 = x^3 + ax + b, \tag{3.1}$$

where a and b are constants. That equation is formally known as a **Weierstrass equation**. Figure 3.1 shows a graphical representation of such a function for $\{a = 1, b = 1\}$. For completeness, an elliptic curve includes a point O of coordinate (∞, ∞) known as *point at infinity*, its importance will be implicitly shown later.

Given a field $\mathbb{K}$, we say that an elliptic curve E is defined over $\mathbb{K}$ if $a, b \in \mathbb{K}$, and we denote by $E(\mathbb{K})$ the set of points all points (x, y) lying on the curve E including the point O. Formally speaking,

$$E(\mathbb{K}) = \{O\} \cup \{(x, y) \in \mathbb{K} \times \mathbb{K} \mid y^2 = x^3 + ax + b\}$$

3.2.1 Group on Elliptic curve

From an elliptic curve E defined over a field $\mathbb{K}$ and the set of points $E(\mathbb{K})$, if the equation 3.2 is verified,

$$4a^3 + 27b^2 \neq 0 \tag{3.2}$$

We can define a group equipped with the addition operation $(+)$ denoted by $(E(\mathbb{K}), +)$, and whose properties are the following:

- (Closure) If $A, B \in E(\mathbb{K})$ then $A + B \in E(\mathbb{K})$
- (Associativity) $A + (B + C) = (A + B) + C$ for all $A, B, C \in E(\mathbb{K})$
- (Commutativity) $A + B = B + A$ for all $A, B \in E(\mathbb{K})$

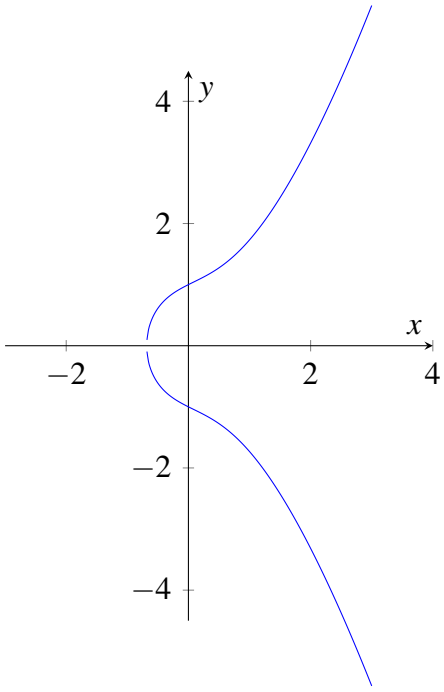


Figure 3.1: Elliptic curve $y^2 = x^3 + x + 1$

- (Identity element) $P + O = P$ for all $P \in E(\mathbb{K})$
- (Inverse element) Given $P \in E(\mathbb{K})$, there exist an element $\overline{P} \in E(\mathbb{K})$ such that $P + \overline{P} = \overline{P} + P = O$

It is worth nothing that in a group generated over an elliptic curve, a multiplication operation $(\times)$ is defined as a binary operation between a scalar and a point, i.e., given a scalar $a \in \mathbb{K}$ and a point $P \in E(\mathbb{K})$,

$$u \times P = \underbrace{P + P + P + \cdots + P}_{u \text{ times}}$$

which is equivalent to the addition of the point P on itself *u times*.

Numerical definition of addition. Given two points $P, Q \in E(\mathbb{K})$ of coordinates (x_P, y_P), (x_Q, y_Q), respectively, it is possible to graphically add them and find the resulting point R of coordinates (x_R, y_R). However, in this work, we are interested in a computable form of the addition operation. Hence, when we compute $R = P + Q$, the addition operation

is performed as follows:

- Case $x_P \neq x_Q$:

$$\begin{cases} x_R = m^2 - x_P - x_Q \\ y_R = m(x_P - x_R) - y_P \end{cases} \tag{3.3}$$

  where $m = \frac{y_Q - y_P}{x_Q - x_P}$

- Case $x_P = x_Q$ and $y_P = y_Q$:

$$\begin{cases} x_R = m^2 - 2x_P \\ y_R = m(x_P - x_R) - y_P \end{cases} \tag{3.4}$$

  where $m = \frac{3x_p^2 + a}{2y_P}$, with a being the value set in equation 3.1.

- Case $x_P = x_Q$ and $y_P = -y_Q$:

$$P + Q = O \tag{3.5}$$

  where O is the point at infinity. In this case, Q is the inverse of the point P and can be denoted as $-P$.

- Case $x_P = x_Q$ and $y_P = y_Q = 0$:

$$P + Q = O \tag{3.6}$$

  it should be noted that in real-world applications, it is difficult to encounter this case.

3.2.2 Elliptic curve over finite field

An elliptic curve E defined over a finite field $\mathbb{F}_q$ is a function of the form

$$y^2 \bmod q = (x^3 + ax + b) \bmod q$$

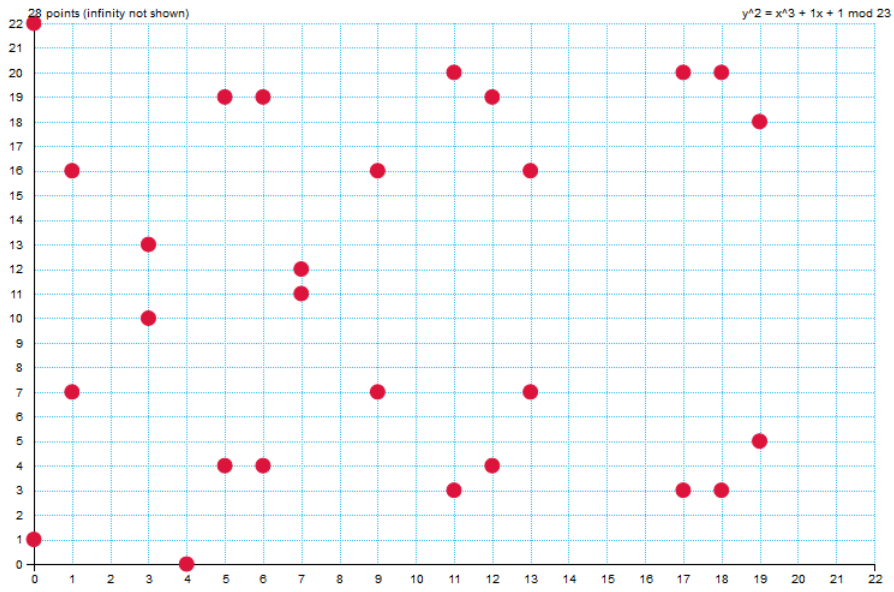


Figure 3.2: Elliptic curve $y^2 = x^3 + x + 1$ over $\mathbb{F}_{23}$

where $a, b \in \mathbb{F}_q$ satisfy the equation 3.2, and q is a prime greater than three. We use $E/\mathbb{F}_q$ to denotes an elliptic curve E defined over $\mathbb{F}_q$.

Figure 3.2 represents a graph of the elliptic curve $y^2 = x^3 + x + 1$ defined over $\mathbb{F}_{23}$. By comparing figure 3.1 and figure 3.2, we can observe that points in figure 3.2 seem to have no correlation, except the fact the graph has a symmetry around the middle.

Points addition over $\mathbb{F}_q$. The addition of points in $E(\mathbb{F}_q)$ follows the same definitions as equations 3.3, 3.4, 3.5, 3.6 with respect to modular arithmetic.

Advantages of Elliptic Curve in Cryptography. Using Elliptic curve over finite field to develop cryptography schemes has the advantage of being less resource-demanding than classical systems, such as RSA. For instance, Blake *et al.* showed that a key of 313-bits in an elliptic curve cryptosystem offered the same security advantages as a key of 4096-bits in RSA (Blake, Seroussi, & Smart, 1999). Furthermore, Daswani and Boneh conducted experiments using a PalmPilot, which was a hand-held device released in 1997 with a CPU of 16MHz, and they showed that generating a 512-bit RSA took 3.4 minutes whereas generating a 163-bits Elliptic curve digital signature took 0.597 seconds (Daswani & Boneh, 1999).

Intractable problems. In computer science, a problem is said to be intractable if there exist no efficient algorithms that can solve it, i.e, there is no algorithm with polynomial time complexity that can solve that problem. Therefore, a **PPT (Probabilistic Polynomial Time) adversary** is any algorithm that has access to a source of randomness and whose number of steps can be express by a function $f()$ such that f is bounded by a polynomial function. *Given $E/\mathbb{F}_q$, for this work, we assume the following two problems to be intractable:*

- **Discrete Logarithm Problem over Elliptic Curve (DLPEC):** Given $P, Q \in E(\mathbb{F}_q)$, it is difficult for any PPT adversary $\mathcal{A}$ to find a value $m \in \mathbb{Z}_n^*$ such that $Q = mP$, i.e., the advantage of $\mathcal{A}$ given by the following expression is negligible:

$$\text{Adv}_{\mathcal{A}}^{ECDLP} = Pr[\{m|Q = mP\} \leftarrow \mathcal{A}(P, Q)]$$

- **Computational Diffie-Hellman problem over Elliptic Curve (CDHEC):** Given tow points $a.P, b.P \in E(\mathbb{F}_q)$ with $a, b \in \mathbb{Z}_n^*$. It is difficult for any PPT adversary $\mathcal{A}$ to compute $ab.P$, i.e., the advantage of $\mathcal{A}$ given by the following expression is negligible:

$$\text{Adv}_{\mathcal{A}}^{CDHEC} = Pr[ab.P \leftarrow \mathcal{A}(aP, bP)]$$

3.2.3 ElGamal Encryption over Elliptic Curve

Given $E/\mathbb{F}_q$ where the DLPEC is instractable in the subgroup $E(\mathbb{F}_q)$, let $P \in E(\mathbb{F}_q)$ be a base point with a large prime order n. The ElGamal encryption over $E(\mathbb{F}_q)$ is a public key encryption scheme made of three algorithms $(\mathsf{GenKey}, \mathsf{Enc}, \mathsf{Dec})$, where GenKey is the key generation algorithm, Enc is the encryption algorithm, and Dec the decryption algorithm. The description of GenKey,Enc, and Dec are as follows:

- GenKey: Alice randomly selects a variable $a \in \mathbb{Z}_n^*$ and computes $A = a.P$. The public key is A and the private key is a.

- Enc: For Bob to send a message m to Alice, it converts m into a point $M \in E(\mathbb{F}_q)$, and uniformly at random selects a secret nonce $r \in \mathbb{Z}_n^*$. Then, it computes $C' = r.P$ and $C = r.A + M$. The ciphertext is (C', C).

- Dec: To decipher the ciphertext (C', C) and obtain the message m from M, Alice computes $K = a.C'$. Then, it computes M from C as follows: $M = C - K = (r.a.P + M) - (r.a.P)$. Once Alice obtains M, it converts the point M into the actual message m.

Homomorphism. An encryption scheme $(\mathsf{GenKey}, \mathsf{Enc}, \mathsf{Dec})$ is homomorphic if it accepts an operator $\circ$, such that $\forall m_1, m_2 \in \mathcal{M}$ the following equation is verified:

$$\mathsf{Enc}(m_1 \circ m_2) = \mathsf{Enc}(m_1) \circ \mathsf{Enc}(m_2)$$

where $\mathcal{M}$ is the set of all possible messages accepted by the scheme.

ElGamal encryption over $E(\mathbb{F}_q)$ is homomorphic for the addition $(+)$ operation. Given the private-public key pair (a, A), two plaintexts m_1, $m_2 \in E(\mathbb{F}_q)$, and some random numbers $r_1, r_2 \in \mathbb{Z}_n^*$, $r = r_1 + r_2$, we demonstrate the homomorphic property:

$$\begin{aligned}
\mathsf{Enc}(m_1) + \mathsf{Enc}(m_2) &= (r_1.P, r_1.a.P + m_1) + (r_2.P, r_2.a.P + m_2) \\
&= ((r_1 + r_2)P, (r_1 + r_2).a.P + (m_1 + m_2)) \\
&= (rP, r.a.P + (m_1 + m_2)) \\
&= \mathsf{Enc}(m_1 + m_2)
\end{aligned}$$

3.3 Blockchain and Smart Contract

This section is based on joint work with William Stone, Jeehyong Kim, Daeyoung Kim, and Junggab Son that appears in IEEE Access (Youdom Kemmoe, Stone, Kim, Kim, & Son, 2020).

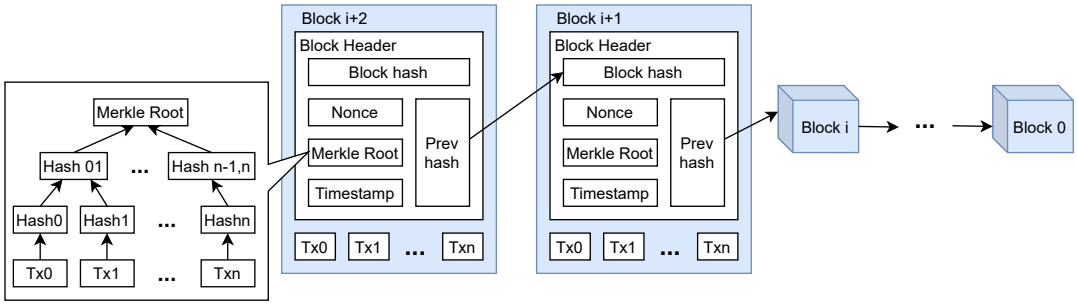


Figure 3.3: Structure of Blockchain

3.3.1 Blockchain

In 2008, Satoshi Nakamoto presented a decentralized payment system named Bitcoin (Nakamoto, 2008). In Bitcoin, the history of transactions is stored in a contiguous set of blocks, in which blocks that are linked and protected through cryptographic primitives. Nakamoto's idea sparked what we know today as *blockchain*. Blockchain is a distributed data structure, more or less similar to a distributed ledger, formed by consecutive blocks linked through cryptographic hashed values. Each block contains a set of transactions, a nonce, a timestamp, and a cryptographic hash value to a preceding block. The nonce, timestamp, hash of the previous block, and an accumulated hash of all transactions are part of the block header. Figure 3.3 shows a general representation of the structure of a blockchain.

A blockchain is maintained by a set of independent nodes on a computer network (each node has a copy of the blockchain). However, It can happen that not all nodes have the same number of blocks. In that case, a general solution is to consider the longest chain as the valid one. Figure 3.4 shows an abstract representation of a blockchain maintained by four independent nodes. Node 2 and Node 4 have four blocks, whereas Node 1 has two blocks and Node 3 has three blocks. In that case, It is the chain held by Node 2 and Node 4 that should be considered as the valid chain.

A blockchain has two basic operations: read (to read the content of blocks) and append (to append new blocks). Although an adversary might try to remove/modify some blocks, it requires extremely expensive computations, which makes them nonviable. For instance,

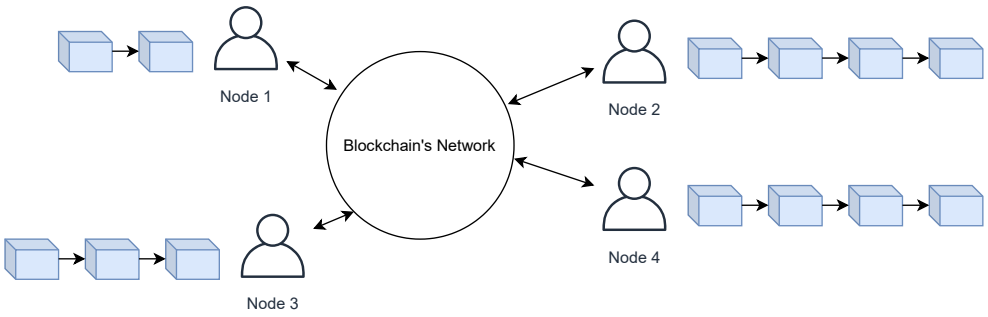


Figure 3.4: Abstract Blockchain maintained by four nodes

given the blockchain

$$b_n \to \cdots \to b_i \to \cdots \to b_2 \to b_1 \to b_0$$

if the block b_i is modify, then its hash value will also change. This will break the link in the blockchain and create two subchains:

$$b_n \to \cdots \to b_{i+2} \to b_{i+1} \qquad b_{i-1} \to \cdots \to b_2 \to b_1 \to b_0$$

Therefore, the original blockchain will become invalid. To ensure that the blockchain stays valid, the attacker will have to modify all the blocks starting from b_{i+1} to b_n (updating the hash value of b_i in b_{i+1} will change the hash value of b_{i+1}) and re-append them. However, for a block to be appended, it must be validated through a consensus protocol. For instance, in Bitcoin, the consensus algorithm requires the hash of a block to have to a specific format (Nakamoto, 2008). Therefore, nodes willing to append new blocks have to compute the hash values of those blocks until the find values that match the specified format. In addition, blockchain has a de facto fault tolerance, because as long as a non-negligible portions of nodes maintaining the blockchain is online, then the blockchain will remain available.

Cryptographic primitives in blockchain. Blockchain is able to provide a trusted execution environment thanks, in part, to crucial cryptographic primitives. Following are two primitives used by blockchain that are important in the establishment of this work:

- **Public Key Cryptosystem**: it is a crytographic system that uses a public-private key

pair such that for a private key k, its related public key K, and a bijective function $f : \mathbb{K} \times \mathbb{I} \to \mathbb{O}$ accepted by the cryptosystem, the equation 3.7 is verified for any pair $(x, y) \in \mathbb{I} \times \mathbb{O}$.

$$\begin{cases} f(K, x) = y \\ f^{-1}(k, y) = x \end{cases} \tag{3.7}$$

where $\mathbb{I}$ is the set of inputs, $\mathbb{O}$ is the set of outputs and $\mathbb{K}$ is the set of keys. Each node U_i interacting with a blockchain has a unique public-private key pair (i, I).

- **Digital Signature**: it is a cryptographic scheme based on public key cryptosystem that allows one party, a *signer*, to digitally sign a message m such that any other parties, *verifiers*, can verify m was issued the signer and was not altered in transit. Formally speaking, a digitial signature scheme consists of three algorithms $(\mathsf{GenKey}, \mathsf{Sign}, \mathsf{Verify})$:
  - $\mathsf{GenKey}(1^\lambda)$: it takes as input a security parameter 1^λ and outputs a private-public key pair (k, K).
  - $\mathsf{Sign}(k, m)$: it takes as input a private key k, a message m. It outputs a signature σ.
  - $\mathsf{Verify}(K, \sigma, m)$: it takes as input a public key K, a signature σ, a message m. It returns "accept" if σ was indeed generated for m using k, or "reject".

  In blockchain, all transactions are digitally signed by their issuers. This prevents a node A from modifying a transaction issued by a node B (Integrity), and it prevents a node C from passing one of its transactions as a transaction issued by another node, or from denying ownership of a transaction it issued (Non-repudiation).

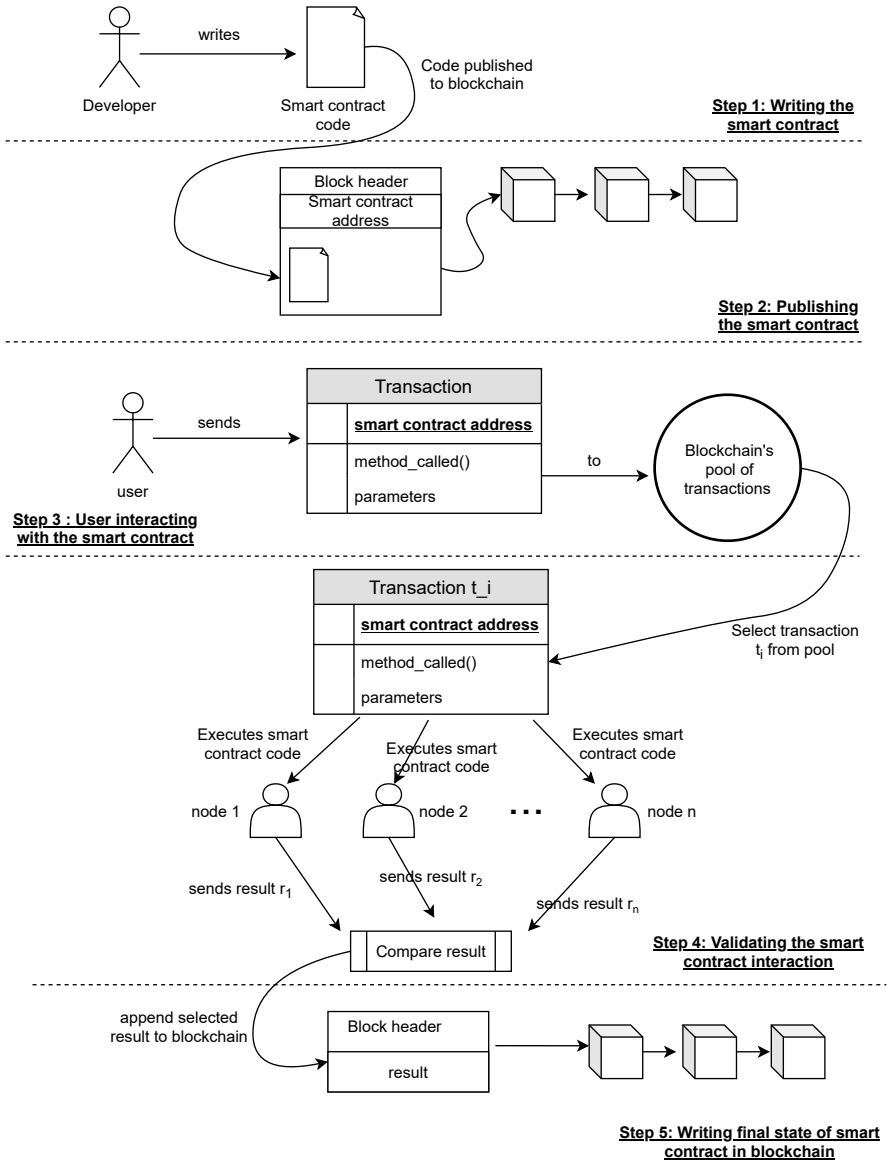


Figure 3.5: Overview of Smart Contracts

3.3.2 Smart Contract

A smart contract is a set of instructions stored on a blockchain that is executed whenever a transaction sent to the blockchain references it. Bitcoin's script language used to execute and verify transactions can be considered as the first application of smart contracts in blockchain. However, it is not turing-complete, i.e., it cannot be used to write any type of program. Later in 2015, Vitalik Buterin (Wood, 2019) created Ethereum, a blockchain platform that features a decentralized payment system and a Turing complete language.

How Smart Contract Work. We provide a high level overview of the functioning of smart contracts, starting from their development until their execution. Figure 3.5 provides a graphical representation of the different steps involve in the functioning of smart contracts. However, before diving in the description of those steps, we give a definition of the terms that will be used:

- Developer: an individual who implements the logic of a smart contract using a specific set of instructions compatible with or provided by a blockchain platform
- User: any entity that uses the services of a smart contract
- Transaction: a query made to a smart contract program
- Blockchain platform: the set of applications and protocols used to maintain and manage a blockchain
- Node: an entity having an account on the blockchain platform that can execute and validate transactions
- Faulty node: a node susceptible to submitting false results after the execution of a smart contract

Following is a step-by-step process of the smart contract:

Step 1: Developers write the logic for the contract in a programming language supported by the blockchain platform they wish to use. Then, using a specific compiler (usually provided by the blockchain platform), they compile the source code representing their smart contract and obtain a byte code.

Step 2: After obtaining the byte code, they will publish it to the blockchain platform where it will be stored on the blockchain. Depending on the platform used, once the smart contract program is published, it will be either read-only or modifiable. For instance, Ethereum

does not allow smart contracts to be modified (Wood, 2019), whereas EOSIO allows overwriting through the uploading of a new byte code (*EOSIO-CLEOS/CLEOS-set*, n.d.). In case it is read-only, to provide an update, the developers will need to publish a new version of the smart contract and redirect users to it. Once uploaded, the smart contract is at its initial state. The initial state represents the initial values of the internal variables of that smart contract.

Step 3: Access to a published smart contract program depends on the blockchain platform. In the case of Ethereum (Wood, 2019) and Neo (*NeoContract White Paper*, n.d.), the blockchain platform returns an address to the developers. The address will then be used to interact with the smart contract. In the case of EOSIO, the smart contract is published to an account (hosted on the blockchain platform) that was previously created by the developers. The identifier of the account will be used to access and interact with the smart contract. Once users obtain the address/identifier, they can begin sending transactions. Each transaction should contain the function of the smart contract that they wish to utilize and the function's arguments. If an amount of platform currency is needed to start the function's execution, that amount will be transferred alongside the transaction. The transaction will be stored in the blockchain platform's pool of transactions that await to be executed and validated. Step 3 in figure 3.5 is based on the functioning of (Wood, 2019; *NeoContract White Paper*, n.d.).

Step 4: From the pool of transactions, the blockchain platform will select a set of transactions to be executed and validated. During the execution phase, the functions of a smart contract that are specified in the transaction will be executed by a set of nodes. During the validation phase, the nodes that executed the transaction will compare their results and select the one to be kept according to a consensus protocol. For instance, in a Byzantine Fault Tolerant (BFT) consensus protocol based on (Pease, Shostak, & Lamport, 1980),

the blockchain platform will select n nodes to execute and validate a set of transactions $T = \{t_1, t_2, ..., t_q\}$, where t_i $(i \in \{1, ..., q\})$ is a transaction and $n \geq 3m + 1$ with m being the maximum number of possible faulty nodes. Each node k will execute the smart contract tied to each transaction and submit a set of results $r_k = \{r_{1_k}, r_{2_k}, ..., r_{q_k}\}$ where r_{i_k} represents the result of each transaction $t_i \in T$, with $k \in (1, ..., n)$. From the set of results $R = \{r_1, r_2, ..., r_n\}$, a result r' that was obtained by n' nodes with $n' > (n + m)/2$ will be considered as the valid result. Also, there is proof-based consensus where instead of a group of nodes agreeing on a final answer, each node has to prove that it has executed a certain operation, or it is in possession of a certain value. The first node to present a valid proof is elected as the leader and is allowed to attach the result of its execution to the blockchain (Nguyen & Kim, 2018). In addition, there are hybrid consensus protocols which are based on both BFT and proof-based protocols (Pass & Shi, 2016; Wu, Song, & Wang, 2020).

Step 5: Once the valid result has been selected, it will be inserted in a block that will be appended to the blockchain. Also, the initial state of each smart contract specified in set T will be updated, i.e., if a validated transaction altered the internal variables of a smart contract, those new values will now be considered as initial values by future transactions.

Chapter 4

Asynchronous Group Key Agreement from Smart Contract

In this chapter, we present a GKA aimed at IoT environments. During the group creation, only the group creator needs to be online, hence the asynchronism. In addition, a smart contract is used to perform some pre-computation and hold necessary information to devise the final group key. This allows to reduce the computational workload and the required amount of storage of group members.

4.1 System and Security Models

4.1.1 System Model

In this section, we give a description of the different components that constitute the system in which our proposed GKA operates.

Ideal Blockchain $\mathsf{F}_{\mathsf{idl}}$. Our system model includes an ideal blockchain denoted by $\mathsf{F}_{\mathsf{idl}}$ and whose properties are the followings:

- $\mathsf{F}_{\mathsf{idl}}$ uses a public key cryptosystem based on an elliptic curve $E/\mathbb{F}_q$ where DLPEC and CDHEC are assumed to be intractable in the subgroup $E(\mathbb{F}_q)$.

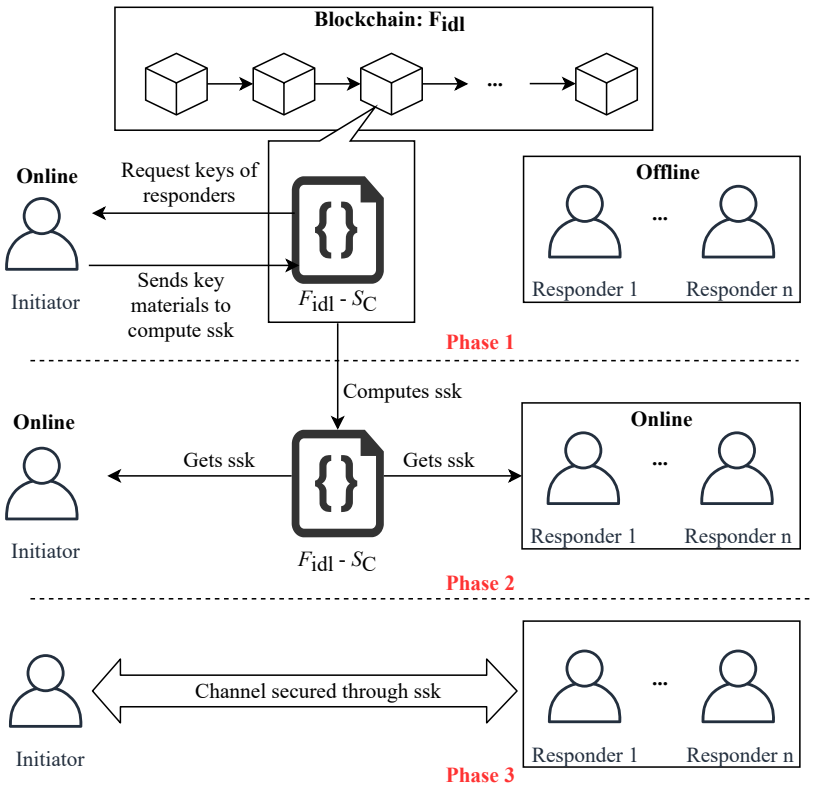


Figure 4.1: Abstract representation of the proposed scheme.

- Each node U_i interacting with $\mathsf{F}_{\mathsf{idl}}$ has an account on the system. The account has a long-term key pair (lk_i, LK_i). The private key lk_i is assumed to be stored offline in a secured memory location of U_i. The public key Lk_i is assumed to be available on $\mathsf{F}_{\mathsf{idl}}$, and it can act has U_i'account identifier.

- Transactions in $\mathsf{F}_{\mathsf{idl}}$ are digitally signed by their issuers. For any transaction T_x, any node can extract the signature and verify it.

- $\mathsf{F}_{\mathsf{idl}}$ supports stateful smart contracts defined using a Turing-complete programming language.

- We assume the access mode of $\mathsf{F}_{\mathsf{idl}}$ to be public, i.e., any node on the blockchain can read information stored in it (Denis, 2018).

Smart Contract $\mathsf{F}_{\mathsf{idl}}\_\mathsf{S}_\mathsf{C}$. Our system is also equipped with a stateful smart contract denoted by $\mathsf{F}_{\mathsf{idl}}\_\mathsf{S}_\mathsf{C}$ that is hosted on $\mathsf{F}_{\mathsf{idl}}$. We provide a formal definition of $\mathsf{F}_{\mathsf{idl}}\_\mathsf{S}_\mathsf{C}$' func-

tionalities in section 4.2.

Participants. Aside from the defined blockchain and smart contract, the system model includes participants that use $\mathsf{F}_{\mathsf{idl}}\_\mathsf{S}_{\mathsf{C}}$ to establish a group key. Let $\mathcal{U}$ be the set of all nodes with an account on $\mathsf{F}_{\mathsf{idl}}$. To participate in the protocol, a node $U_i \in \mathcal{U}$ needs to have a set of ephemeral keys stored in $\mathsf{F}_{\mathsf{idl}}\_\mathsf{S}_{\mathsf{C}}$. From this, we denote $\mathcal{U}_r \subseteq \mathcal{U}$ as the set of all nodes in $\mathsf{F}_{\mathsf{idl}}$ having ephemeral keys stored in $\mathsf{F}_{\mathsf{idl}}\_\mathsf{S}_{\mathsf{C}}$. In $\mathcal{U}_r$, we identify two types of nodes:

- Initiator: it is a node denoted by U_0 that wants to create a group $\mathcal{G}$ with a subset R of nodes in $\mathcal{U}_r$, and establish a group key. It has the role of group administrator that is in charge of adding and removing group members. U_0 is the only node that needs to be online during the execution of the group creation.

- Responders: they are the nodes $R = \{U_1, U_2, \ldots, U_m\}$ that join a group $\mathcal{G}$ created by an initiator U_0, and obtain the exchanged group key, with $R \subseteq \mathcal{U}_r$. Responders are assumed to be less powerful than the initiator and do not need to be online.

During phase one of the protocol's execution, the initiator requests the long-term and ephemeral public keys of responders from $\mathsf{F}_{\mathsf{idl}}\_\mathsf{S}_{\mathsf{C}}$. Then, it uses the keys to compute and transfer key materials to $\mathsf{F}_{\mathsf{idl}}\_\mathsf{S}_{\mathsf{C}}$, which will be used to devise the group key. During phase two, once $\mathsf{F}_{\mathsf{idl}}\_\mathsf{S}_{\mathsf{C}}$ computes the group key (*ssk*), the initiator and responders can request for *ssk* once they are online. During phase three, nodes in $\mathcal{G}$ can use *ssk* to secure their communication channels. Figure 4.1 provides a graphical representation of those phases.

4.1.2 Security Model

In this section, we define the different security properties on which our proposed GKA is based.

- **Weak Backward Secrecy**: new group members cannot have access to previous group keys (Kim, Perrig, & Tsudik, 2004a).

- **Weak Forward Secrecy**: a former group member should not be able to obtain new group keys (Kim et al., 2004a).

- **Key Independence**: an adversary that knows a set of group keys cannot deduce other group keys.

- **Perfect Forward Secrecy (PFS)**: an adversary which is able to obtain the long-term private keys of all group members cannot be able to compute the session keys of previous sessions (Menezes, Oorschot, & Vanstone, 1996). With this property, even if all group members are compromised, the adversary will not be able to obtain a plaintext copy of their past exchanges.

- **Post-Compromise Security (PCS)**: from the definition of Conh-Gordon *et al*. (Cohn-Gordon et al., 2016), PCS is the ability of a protocol to offer security guarantee about a communication between different parties even if one of those parties was compromised. For instance, in case Alice and Bob use a key agreement protocol that provides PCS to secure their communication, if Alice is compromised (the session key and its long-term secret key is exposed), then after a successful execution of a *key refreshing* process, the security of the communication between Alice and Bob should be re-establish. Obviously, this is only possible if during the execution of the key refreshing process, the adversary is passive.

4.2 Group Key Agreement protocol

We now focus on the construction of our GKA. First, we define the different functionalities $\mathsf{F}_{\mathsf{idl}}\_\mathsf{S}_{\mathsf{C}}$, and we introduce the concept of asynchronous biparty key agreement. Second, we define the three functionalities of our proposed GKA: group creation, group key update and member events (join and leave).

$\mathsf{F_{idl}\_S_C}$ functionalities. $\mathsf{F_{idl}\_S_C}$ smart contract is a tuple of PPT algorithms $\mathsf{F_{idl}\_S_C} = (\mathsf{postEphK}, \mathsf{getEphK}, \mathsf{createGroup}, \mathsf{getPreGrpK}, \mathsf{getKeyMaterials}, \mathsf{updateGrpK}, \mathsf{addMember})$ with the following properties:

- $\mathsf{postEphK}(Y_i)$: A node $U_i \in \mathcal{U}$ uploads a set of ephemeral keys Y_i to $\mathsf{F_{idl}\_S_C}$. If there is already a set Y_i stored in $\mathsf{F_{idl}\_S_C}$, executing *postEphK* will overwrite Y_i with the new values.

- $\mathsf{getEphK}(LK_i)$: it returns an unused ephemeral key EK_i from the set Y_i stored in $\mathsf{F_{idl}\_S_C}$ of a node U_i with public key LK_i.

- $\mathsf{createGroup}(\mathcal{G}, Z, K, \Phi, \mathcal{Y})$: this function takes as inputs $\mathcal{G}$ (the set of group members), Z, K, Φ (three set of key materials), $\mathcal{Y}$ (the set of ephemeral keys of members in $\mathcal{G}$). It computes the pre-group key for the group $\mathcal{G}$ and stores K, Φ, and $\mathcal{Y}$ on $\mathsf{F_{idl}\_S_C}$. In addition, it generates and returns $\mathcal{G}$.ID, which is the ID of the group, to all members of $\mathcal{G}$.

- $\mathsf{getPreGrpK}(\mathcal{G}.\mathrm{ID})$: this function is triggered by a node $U_i \in \mathcal{G}$ with $\mathrm{ID} = \mathcal{G}.\mathrm{ID}$. It returns the pre-group key of the group $\mathcal{G}$ to the U_i.

- $\mathsf{getKeyMaterials}(\mathcal{G}.\mathrm{ID})$: this function is triggered by a node $U_i \in \mathcal{G}$ with $\mathrm{ID} = \mathcal{G}.\mathrm{ID}$. It returns the sets $K, \Phi, \mathcal{Y}$ to the U_i.

- $\mathsf{updateGrpK}(\mathcal{G}.\mathrm{ID}, K, \mathcal{Y}, \alpha)$: this function is triggered by a node $U_i \in \mathcal{G}$ with $\mathrm{ID} = \mathcal{G}.\mathrm{ID}$. It updates the pre-group key ssk_{pre} using α and replaces K and $\mathcal{Y}$ stored in $\mathsf{F_{idl}\_S_C}$ with the ones passed as arguments.

- $\mathsf{addMember}(\mathcal{G}.\mathrm{ID}, z, b, \Phi', \mathcal{Y}')$: this function is triggered by U_0, the initiator of the group $\mathcal{G}$ with $\mathrm{ID} = \mathcal{G}.\mathrm{ID}$. It updates Φ with Φ', and $\mathcal{Y}$ with $\mathcal{Y}'$. Also, it appends b to K. Then, it updates the pre-group key using z to include the new member.

Asynchronous biparty key agreement(ABKA). It is a key agreement protocol that allows two parties, initiator and responder, to asynchronously establish a common session

key. In our case, the purpose of such a protocol is to allow the initiator to obtain the different key materials of responders in such a way that no adversary can also obtain them. Therefore, the ABKA must be robust. Thus, for our proposed GKA, we consider **X3DH (extended tripple Diffie-Hellman) protocol** (Marlinspike & Perrin, 2016) as a suitable candidate (also, it is not patented). It uses private-public key pairs defined over $E/\mathbb{F}_q$. Let us consider two parties Alice and Bob. Alice has an identity key pair (ik_a, IK_a), private and public, respectively. Bob has an identity key pair (ik_b, IK_b), a signed pre-key pair (sk_b, SK_b), and a one-time pre-key pair (ok_b, OK_b). The public part of each key pair is available on a TTP. For Alice to exchange a secret key with Bob who is offline, Alice proceeds as follows:

- Alice generates an ephemeral key pair (ek_a, EK_a) and sends EK_a to the TTP.
- Alice request $Ik_b, SK_b,$ and OK_b from the TTP
- Then, Alice executes X3DH $(ik_a, IK_b, ek_a, SK_b, OK_b)$. That operation computes:

$$ssk = KDF[(SK_b)^{ik_a}||(IK_b)^{ek_a}||(SK_b)^{ek_a}||(OK_b)^{ek_a}],$$

where KDF is a Key Derivation Function which can be implemented using a cryptographic hash function (Krawczyk, 2010). It should be noted that OK_b can be omitted. In that case, Alice executes X3DH (ik_a, IK_b, ek_a, SK_b) which computes:

$$ssk = KDF[(SK_b)^{ik_a}||(IK_b)^{ek_a}||(SK_b)^{ek_a}]$$

Bob obtains the exchanged key by executing X3DH(ik_b, IK_a, sk_b, EK_a) which computes:

$$ssk = KDF[(IK_a)^{sk_b}||(EK_a)^{ik_b}||(EK_a)^{sk_b}]$$

For the purpose of our proposed scheme, we have omitted the concept of signed pre-

key and one-time key. Furthermore, we consider long-term key pair to be equivalent with identity key pair. For instance, given a node Alice having long-term key (lk_a, LK_a), ephemeral key (ek_a, EK_a), and a node Bob having long-term key (lk_b, LK_b) and ephemeral key (ek_b, EK_b), we have the following:

$$\text{X3DH}(lk_a, LK_b, ek_a, EK_b) = \text{X3DH}(lk_b, LK_a, ek_b, EK_a)$$

4.2.1 Group creation

The group creation process is the core of our proposed GKA. After a successful execution of this process, a group $\mathcal{G}$ is formed between an initiator and its responders, and a group key is established. It constitutes of four subprocesses which are setup, Initiation, Smart Contract Computation, and Acquisition of the group key. Figure 4.2 shows the overall group creation process with the different transactions involved.

Setup. This process is divided into two phases:

- **Phase 1**: The setup process copies the parameter of the elliptic curve $E/\mathbb{F}_q$ used by $\mathsf{F}_{\mathsf{idl}}$. Specifically, it uses the same large prime q used by $\mathsf{F}_{\mathsf{idl}}$ to define $E'/\mathbb{F}_q$ such that $E'(\mathbb{F}_q) \equiv E(\mathbb{F}_q)$. In addition, it selects the base point $P \in E'(\mathbb{F}_q)$ with large prime order n such that P is the same base point used by $E/\mathbb{F}_q$. It selects two cryptographic hash functions $H_1 : \mathbb{F}_q \to E(\mathbb{F}_q)$ and $H_2 : \{0,1\}^* \to \{0,1\}^\ell$, where ℓ is a fixed length. It is worth nothing that this phase is only executed by all nodes when instantiating the system for the first time.

- **Phase 2**: Each node $U_i \in \mathcal{U}$ generates a set of ephemeral keys $EP_i = (y_i, Y_i)$, where $y_i = \{ek_{i1}, ek_{i2}, \ldots, ek_{iw}\}$, and $Y_i = \{EK_{i1}, EK_{i2}, \ldots, EK_{iw}\}$. For $1 \leq j \leq w$, $ek_{ij} \in_R \mathbb{Z}_n^*$ and $EK_{ij} = ek_{ij} \times P$. Then U_i posts Y_i to $\mathsf{F}_{\mathsf{idl}}\_\mathsf{S_C}$ by sending the transaction $\mathsf{F}_{\mathsf{idl}}\_\mathsf{S_C}$.postEphK($Y_i$) to $\mathsf{F}_{\mathsf{idl}}$. It is worth nothing that EP_i should be frequently replaced.

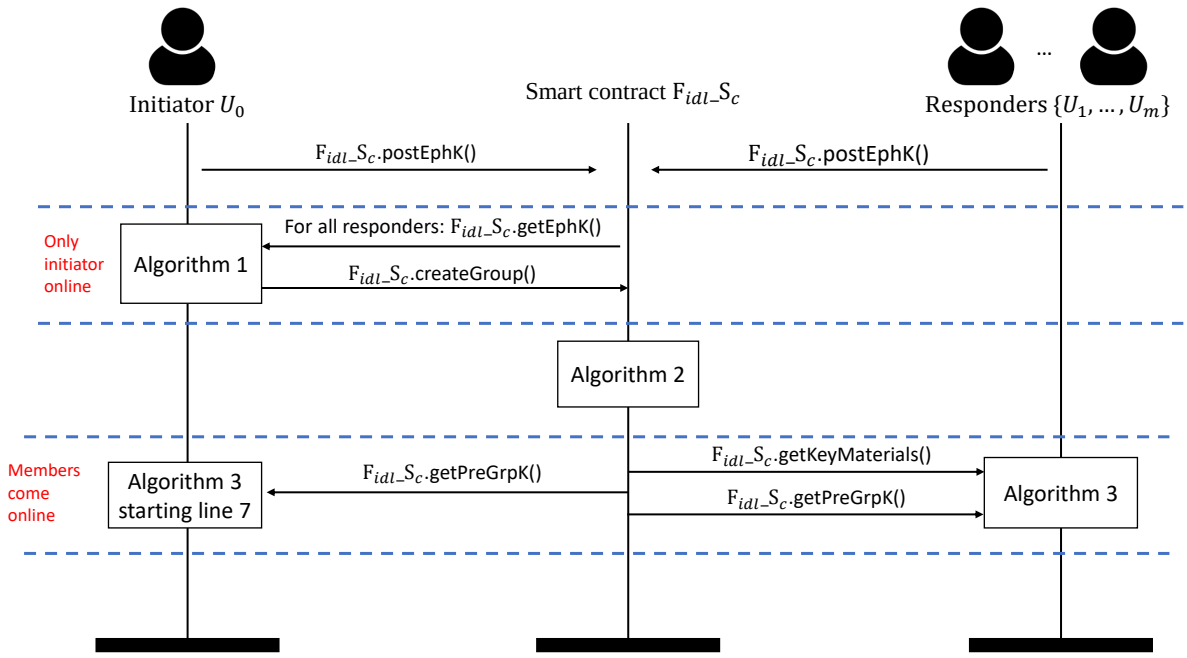


Figure 4.2: Group creation process

Initiation. During the initiation process, the initiator U_0 computes the essential key materials that will be used to generate the group key and sends those to $\mathsf{F_{idl}\_S_C}$. We outline this process in Algorithm 1. Firstly, U_0 selects the variables $\lambda_0, s \in_R \mathbb{F}_q$, and $k \in_R \mathbb{Z}_n^*$ and a non-used ephemeral key pair (ek_0, EK_0). Secondly, it encrypts $H_1(\lambda_0)$ with EK_0 using ElGamal encryption. Finally, for each responder in $R = \{U_1, \ldots, U_m\}$, it gets a non-used ephemeral key from $\mathsf{F_{idl}\_S_C}$ and computes λ using X3DH protocol. Then, it encrypts k and s for each member using $H_2(\lambda)$ and the collected ephemeral key. After this process, it deletes λ. Once everything is done, it sends a transaction $\mathsf{F_{idl}\_S_C}$.createGroup() to $\mathsf{F_{idl}}$.

Smart Contract Computation. Once U_0 terminates the initiation process, the function $\mathsf{F_{idl}\_S_C}$.createGroup() is executed by $\mathsf{F_{idl}}$. We outline this process in Algorithm 2. The pre-group key (ssk_{pre}) is computed as the sum of the encrypted $H_1(\lambda_i)$ variables stored in Z. Following is the value of ssk_{pre} once the computation is done:

$$ssk_{pre} = \sum_{i=0}^{m} [H_1(\lambda_i) + k(EK_i)]$$

Algorithm 1: Initiation of Group

Input: $\mathcal{G} = \{U_0\} \cup R$
Require: $Z[0:m]$, $\mathcal{Y}[0:m]$, $K[1:m]$, $\Phi[0:m]$
begin
  Select $\lambda_0, s \in_R \mathbb{F}_q$ and $k \in_R \mathbb{Z}_n^*$
  Select a non-used ephemeral key pair (ek_0, EK_0)
  $Z[0] \leftarrow H_1(\lambda_0) + k \times EK_0$
  $\mathcal{Y}[0] \leftarrow EK_0$
  for $i \leftarrow 1$ **to** m **do**
    $\mathcal{Y}[i] \leftarrow \mathsf{F_{idl}\_S_C}.\text{getEphK}(LK_i)$
    $\lambda \leftarrow \text{X3DH}(lk_0, LK_i, ek_0, \mathcal{Y}[i])$
    $Z[i] \leftarrow H_1(\lambda) + k \times \mathcal{Y}[i]$
    $K[i] \leftarrow k \oplus H_2(\lambda)$
    $\Phi[i] \leftarrow s \oplus H_2(\lambda || \mathcal{Y}[i].x)$
    Delete λ
  end
  Send transaction $\mathsf{F_{idl}\_S_C}.\text{createGroup}(\mathcal{G}, Z, K, \Phi, \mathcal{Y})$ to $\mathsf{F_{idl}}$
end

The righteousness of this operation is verified by the homomorphic property of ElGamal encryption. During this process, $\mathcal{G}$.ID, the identifier of the group $\mathcal{G}$, is generated and forwarded to all members in $\mathcal{G}$.

Algorithm 2: Computation of Pre-Group Key

Function *createGroup*$(\mathcal{G}, Z, K, \Phi, \mathcal{Y})$ **is**
  $ssk_{pre} \leftarrow 0$
  for $i \leftarrow 0$ **to** m **do**
    $ssk_{pre} \leftarrow ssk_{pre} + Z[i]$
  end
  Generate $\mathcal{G}$.ID
  Store $\mathcal{G}, Z, K, \Phi, \mathcal{Y}$
  return $\mathcal{G}$.ID
end

Acquisition of the group key. Once the Smart Contract Computation step is completed, responders can derive the group key by executing Algorithm 3. Firstly, each $U_i \in R$ needs to get the key materials from $\mathsf{F_{idl}\_S_C}$. Then, using X3DH protocol, they obtain the λ that was computed by the initiator during the Initiation step. Using λ, they decipher the

encrypted value of k and s. Secondly, they get the pre-group key from $\mathsf{F_{idl}\_S_C}$ and compute θ:

$$\theta = H_1(\lambda_0) + H_1(\lambda_1) + \cdots + H_1(\lambda_m).$$

Finally, using KDF, the responders obtain ssk from θ and s. Then, they delete θ and s. The initiator also executes Algorithm 3 to get the group key. However, it starts the execution at line 7 because it already has the key materials stored in local memory.

Algorithm 3: Acquisition of Group Key

Input : idx `//the index of node in` $\mathcal{G}$
Return: ssk `//The group key`

1 **begin**
2 $K, \Phi, \mathcal{Y} \leftarrow \mathsf{F_{idl}\_S_C.getKeyMaterials}(\mathcal{G}.\text{ID})$
3 Get EK_{idx} and EK_0 from $\mathcal{Y}$
4 $\lambda \leftarrow \text{X3DH}(lk_{idx}, LK_0, ek_{idx}, EK_0)$
5 $k \leftarrow K[idx] \oplus H_2(\lambda)$
6 $s \leftarrow \Phi[idx] \oplus H_2(\lambda || EK_{idx}.x)$
7 $ssk_{pre} \leftarrow \mathsf{F_{idl}\_S_C.getPreGrpK}(\mathcal{G}.\text{ID})$
8 $w \leftarrow 0$
9 **for** $i \leftarrow 0$ **to** m **do**
10 $w \leftarrow w + \mathcal{Y}[i]$
11 **end**
12 $w \leftarrow k \times w$
13 $\theta \leftarrow ssk_{pre} - w$
14 $ssk \leftarrow \text{KDF}(s || \theta.x)$
15 Delete θ, s
16 **return** ssk
17 **end**

4.2.2 Group Key Update

To satisfy PCS's requirements, our scheme needs to provide a mechanism to update the group key such that the new group key depends on both the previous group key and freshly generated key materials. This dependency is demonstrated by Cohn-Gordon *et al.* (Cohn-Gordon et al., 2016).

Algorithm 4: Initiation of Key Update

Input : idx `//index of node initiating key update`
Require: $\mathcal{Y}[0:m], K'[1:m]$

begin
  Select $k' \in_R \mathbb{Z}_n^*$, $\lambda'_{idx} \in_R \mathbb{F}_q$, $ek'_{idx} \in y_{idx}$
  $\mathcal{Y}[idx] \leftarrow EK'_{idx}$
  for $i \leftarrow 0$ **to** (m), $i \neq idx$ **do**
    $\mathcal{Y}[i] \leftarrow \mathsf{F_{idl}\_S_C}.\text{getEphK}(LK_i)$
    $\lambda' \leftarrow \text{X3DH}(lk_{idx}, LK_i, ek_{idx}, \mathcal{Y}[i])$
    $K'[i] \leftarrow k' \oplus H_2(\lambda)$
    Delete λ'
  end
  $\alpha \leftarrow 0$
  for $i \leftarrow 0$ **to** (m) **do**
    $\alpha \leftarrow \alpha + \mathcal{Y}[i]$
  end
  $\alpha \leftarrow \alpha \times k' - w - H_1(\lambda_{idx}) + H_1(\lambda'_{idx})$
  Send transaction $\mathsf{F_{idl}\_S_C}.\text{updateGrpK}(\mathcal{G}.\text{ID}, K', \mathcal{Y}, \alpha)$ to $\mathsf{F_{idl}}$
end

The key update process is two-fold. Let us $U_{idx} \in \mathcal{G}$ be a node that wants to update the group key. Firstly, U_{idx} executes the Initiate Key Update process outlined in Algorithm 4. It selects $k' \in_R \mathbb{Z}_n^*, \lambda'_{idx} \in_R \mathbb{F}_q$ and a non-used ephemeral key pair (ek'_{idx}, EK'_{idx}). Next, it gets a set of non-used ephemeral key of other members in $\mathcal{G}$ from $\mathsf{F_{idl}\_S_C}$. Then, using X3DH protocol, it computes λ' for others. After that, it encrypts k' using λ'. Finally it computes α and sends the transaction $\mathsf{F_{idl}\_S_C}.\text{updateGrpK}()$ to $\mathsf{F_{idl}}$. Following the homomorphic property of ElGamal encrytion, we have

$$\alpha = \sum_{i=0}^{m} [k'(EK'_i)] - \sum_{i=0}^{m} [k(EKi)] - H_1(\lambda_{idx}) + H_1(\lambda'_{idx}),$$

where EK'_i represents the newly selected ephemeral keys during Algorithm 4 and EK_i represents the old ephemeral keys.

Secondly, once $\mathsf{F_{idl}}$ receives $\mathsf{F_{idl}\_S_C}$.updateGrpK(), it updates ssk_{pre} to ssk'_{pre} by computing $ssk_{pre} + \alpha$:

$$ssk'_{pre} = H_1(\lambda_0) + \cdots + H_1(\lambda_{idx}) - H_1(\lambda_{idx}) + H_1(\lambda'_{idx}) + \cdots + H_1(\lambda_m) + \sum_{i=0}^{m}[k'(EK'_i)]$$

After this, other nodes in $\mathcal{G}$ can obtain the new group key according to Algorithm 5. Each of which gets the new key materials from $\mathsf{F_{idl}\_S_C}$, and using X3DH protocol, it computes λ'. Then, each node deciphers the encrypted value of k' by using $H_1(\lambda')$. Next, each node gets ssk'_{pre} from $\mathsf{F_{idl}\_S_C}$, and from ssk'_{pre}, it computes θ'. Finally, using KDF, each node derives the new group key ssk' from the previous key ssk and θ'. The initiator of the group key update, U_{idx}, also obtains the new group key by executing Algorithm 5, but starting at line 5 since it already has the new key materials stored in local memory.

Algorithm 5: Acquisition of New Key

Input : j //the index of node in $\mathcal{G}$ getting in the new key
Return: ssk'//the new group key

1 **begin**
2 $K', \Phi, \mathcal{Y} \leftarrow \mathsf{F_{idl}\_S_C}$.getKeyMaterials($\mathcal{G}$.ID)
3 $\lambda' \leftarrow$ X3DH$(lk_j, LK_{idx}, ek_j, EK_{idx})$
4 $k' \leftarrow K'[j] \oplus H_2(\lambda)$
5 $ssk'_{pre} \leftarrow \mathsf{F_{idl}\_S_C}$.getPreGrpK($\mathcal{G}$.ID)
6 $w' \leftarrow 0$
7 **for** $i \leftarrow 0$ **to** $(|\mathcal{G}|)$ **do**
8 $w' \leftarrow w' + \mathcal{Y}[i]$
9 **end**
10 $w' \leftarrow k' \times w'$
11 $\theta' \leftarrow ssk'_{pre} - w'$
12 $ssk' \leftarrow$ KDF$(ssk||\theta'.x)$
13 Delete θ'
Return: ssk'
14 **end**

4.2.3 Member Events

Given a group $\mathcal{G}$, it is possible for the initiator, U_0, to add or remove a member. Having access to these functionalities is particularly important because IoT environments are dynamic, i.e., devices are added and removed depending on users' needs.

Add Member. The addition of member is a process divided into two parts. First, for U_0 to add a new member U_{m+1} to a group $\mathcal{G}$, U_0 initiates the addition of a new member process as outlined by Algorithm 6. More in details, U_0 selects a new secrete variable $s' \in_R \mathbb{F}_q$ and a non-used ephemeral key pair (ek_0, EK_0). Then, for each member in $\mathcal{G}$ including the new member U_{m+1}, U_0 gets a non used ephemeral key from $\mathsf{F_{idl}\_S_C}$ and computes λ. Using $H_2(\lambda)$, U_0 encrypts the secret variables s' for each member. For the new member U_{m+1}, U_0 computes $b = k \oplus H_2(\lambda)$ (since U_{m+1} will need k to get the group key, but others already possess k) and $z = H_1(\lambda) + k \times EK'_{m+1}$ (this variable will be used to update the pre-group key). Once everything has been computed, U_0 erases λ from memory and sends the transaction $\mathsf{F_{idl}\_S_C}$.addMember() to $\mathsf{F_{idl}}$.

Second, once $\mathsf{F_{idl}}$ receives the transaction $\mathsf{F_{idl}\_S_C}$.addMember(), it triggers the execution of the function addMember() with the given parameters. That function updates the values of ssk_{pre} as follows:

$$\begin{aligned} ssk_{pre} &= ssk_{pre} + z \\ &= \sum_{i=0}^{m+1} [H_1(\lambda_i) + k(EK'_i)] \end{aligned}$$

After the smart contract completes its execution, the members of $\mathcal{G}$ derive the new group key by following Algorithm 7. More precisely, the members $\{U_1, U_2 \ldots, U_m, U_{m+1}\}$ get the update key materials from $\mathsf{F_{idl}\_S_C}$ through the transaction $\mathsf{F_{idl}\_S_C}$.getKeyMaterials(). Then, from $\mathcal{Y}$, each extracts U_0's ephemeral key and its ephemeral key used by U_0. Using those keys, each computes λ by using the X3DH protocol and deciphers s'. However, U_{m+1}, the new member, gets the new pre-group key ssk'_{pre} from $\mathsf{F_{idl}\_S_C}$, obtains the secret value k

and computes the sum of ephemeral keys used by U_0. Using that sum, k and ssk'_{pre}, U_{m+1} computes θ'. Then, using θ' and s', it computes the new group key as $ssk' = KDF(s'||\theta'.x)$. For the old group members $\{U_0, U_1, \ldots, U_m\}$, they execute the *else-case* in Algorithm 7. They request the pre-group key from $\mathsf{F}_{\mathsf{idl}}\_\mathsf{S}_\mathsf{C}$ and compute θ'. Then, they compute the new group key.

Algorithm 6: Initiate the addition of a new member

Input: $\mathcal{G} = \{U_0, U_1, \ldots, U_m\}$,
    U_{m+1}//The new member to be added
Require: $\mathcal{Y}'[0:m+1]$, $\Phi'[1:m+1]$
begin
    Select $s' \in_R \mathbb{F}_q$
    Select a non-used ephemeral key pair (ek_0, EK_0)
    $\mathcal{Y}'[0] \leftarrow EK_0$
    for $i \leftarrow 1$ **to** $(m+1)$ **do**
        $\mathcal{Y}'[i] \leftarrow \mathsf{F}_{\mathsf{idl}}\_\mathsf{S}_\mathsf{C}$.getEphK$(LK_i)$
        $\lambda \leftarrow$ X3DH$(lk_0, LK_i, ek_0, \mathcal{Y}[i])$
        $\Phi'[i] \leftarrow s' \oplus H_2(\lambda||\mathcal{Y}[i].x)$
        if *i=m+1* **then**
            $z = H_1(\lambda) + k \times \mathcal{Y}'[i]$
            $b = k \oplus H_2(\lambda)$
        Delete λ
    end
    Send transaction $\mathsf{F}_{\mathsf{idl}}\_\mathsf{S}_\mathsf{C}$.addMember$(\mathcal{G}$.ID$, z, b, \Phi', \mathcal{Y}')$
end

Remove Member. To remove a member $U_j \in \mathcal{G}$, U_0 simply re-executes the **group creation process** starting at the Initiation for the new group $\mathcal{G}/U_j$.

4.3 Security Analysis

In this section, we thoroughly analyze the security of the proposed group key exchange scheme.

Man-in-the-Middle attack: Given the current state of our GKA, it is possible for an adversary $\mathcal{A}$ to impersonate the initiator U_0 from the point of view of a member U_i and to

Algorithm 7: Get new key after member addition

Input : idx//the index of node in $\mathcal{G}$
Return: ssk' //The new group key

begin
  $K, \Phi, \mathcal{Y} \leftarrow \mathsf{F_{idl}\_S_C}.\mathsf{getKeyMaterials}(\mathcal{G}.\mathrm{ID})$
  Get EK_{idx} and EK_0 from $\mathcal{Y}$
  $\lambda \leftarrow \mathrm{X3DH}(lk_{idx}, LK_0, ek_{idx}, EK_0)$
  $s' \leftarrow \Phi[idx] \oplus H_2(\lambda || EK_{idx}.x)$
  if $idx = m+1$ **then**
    //This case is performed by the new member U_{m+1}
    $ssk'_{pre} \leftarrow \mathsf{F_{idl}\_S_C}.\mathrm{getPreGrpK}(\mathcal{G}.\mathrm{ID})$
    $k = K[idx] \oplus H_2(\lambda)$
    $w' \leftarrow 0$
    for $i \leftarrow 0$ **to** *m+1* **do**
      $w' \leftarrow w' + \mathcal{Y}[i]$
    end
    $w' \leftarrow k \times w'$
    $\theta' = ssk'_{pre} - w'$
  else
    //This case is performed by the old members including the initiator
    $w' \leftarrow w + k \times \mathcal{Y}[m+1]$
    $ssk'_{pre} \leftarrow \mathsf{F_{idl}\_S_C}.\mathrm{getPreGrpK}(\mathcal{G}.\mathrm{ID})$
    $\theta' = ssk'_{pre} - w'$
  end
  $ssk' = KDF(s' || \theta'.x)$
  Delete s', θ'
  Return: ssk'
end

impersonate U_i from the point of view of U_0. To prevent this attack, once can use a smart contract to maintain a list of authorized nodes in case the amount of connected devices is small. However, for large infrastructures, one can use a Decentralized Public Key Infrastructure (DPKI) on top of our scheme to authenticate public keys. (Sivakumar & Singh, 2017; Al-Bassam, 2017; Patsonakis, Samari, Kiayiasy, & Roussopoulos, 2019) propose DPKI systems based on smart contracts, which make them suitable candidates to supplement our scheme.

Definition 1 (**Session).** *We consider a* session *to be a communication channel between members in a group $\mathcal{G}$ protected by a secret group key ssk.*

Theorem 1 (**Perfect Forward Secrecy).** *Given $E/\mathbb{F}_q$ with base point P of order n, a group $\mathcal{G} = \{U_0, \ldots, U_m\}$, their long-term private keys $\delta = \{lk_0, \ldots, lk_m\}$ and their long-term public keys $\Delta = \{LK_0, \ldots, LK_m\}$, there is no PPT adversary $\mathcal{A}$ who can reveal ssk, the secret group key of $\mathcal{G}$.*

Proof. From the Algorithm 3, $ssk = \text{KDF}(s||\theta.x)$. Given that $\mathcal{F}_{idl}$ access is public, $\mathcal{A}$ can obtain ssk_{pre}, $\mathcal{Y} = \{EK_0, \ldots, EK_m\}$, $\Phi = \{s \oplus H_2(\lambda_1||EK_0.x), \ldots, s \oplus H_2(\lambda_m||EK_m.x)\}$ and $K = \{k \oplus H_2(\lambda_1), \ldots, k \oplus H_2(\lambda_m)\}$ from $\mathcal{F}_{idl}$-$\mathcal{S}_C$.

From Algorithm 1, $\lambda_i = \text{X3DH}(lk_0, LK_i, ek_0, EK_i)$. However, under CDHEC, it is impossible for $\mathcal{A}$ to compute λ_i from given $\delta, \Delta,$ and $\mathcal{Y}$. Without λ_i, $\mathcal{A}$ cannot obtain k and s from K and Φ. Also, under DLPEC, $\mathcal{A}$ cannot find k and compute $w = \sum_{i=0}^{m}[k(EK_i)]$ from $\mathcal{Y}$. $\mathcal{A}$ cannot compute $\theta = ssk_{pre} - w$. Thus, given $\delta, \Delta, ssk_{pre}, \mathcal{Y}, \Phi$ and K, $\mathcal{A}$ cannot compute ssk. Therefore, Perfect Forward Secrecy is verified. □

Theorem 2 (**Known session key attack - Security).** *Given a group $\mathcal{G} = \{U_0, \ldots, U_m\}$ and a set of their previous session keys $\mathcal{D} = \{ssk_1, ssk_2, \ldots, ssk_n\}$, it is impossible for any PPT adversary $\mathcal{A}$ to reveal ssk_{n+1}, the secret key of a future session, or ssk_0, the secret key of a past session.*

Proof. First, let's suppose that every key in $\mathcal{D}$ were derived after a group creation process or member addition process. In this case, any session key in $\mathcal{D}$ has the form $ssk = \text{KDF}(s||\theta.x)$, with $\theta = \sum_{i=0}^{m} H_1(\lambda_i)$ and $s \in_R \mathbb{F}_q$. Since θ is computed from randomly generated variables and s is randomly generated, ssk is random. Furthermore, since a KDF is collision-resistant (Krawczyk, 2010), ssk is unique, so all keys in $\mathcal{D}$ are uncorrelated. Therefore, $\mathcal{A}$ cannot use key in D to reveal ssk_{n+1} or ssk_0.

Second, let's suppose some keys in $\mathcal{D}$ were derived after a key update, i.e., $ssk_{i+1} = \text{KDF}(ssk_i||\theta'.x)$. In this case, knowing ssk_i is not sufficient to reveal ssk_{i+1}.

Hence, our proposed scheme is resistant against Known session key attack. □

Theorem 3 (Post-compromise Security (PCS))**.** *Against a passive adversary $\mathcal{A}$, given a session between members in a group $\mathcal{G} = \{U_0, \ldots, U_m\}$, it is sufficient for a compromised group member $U_i \in \mathcal{G}$ to perform the update group key process to re-establish the security of the session.*

Proof. Once U_i is compromised, it is possible for $\mathcal{A}$ to obtains $k, \lambda_i,$ and ssk. However, if after the compromised U_i performs the Update Key process,

$$ssk_{pre} = H_1(\lambda_0) + \cdots + H_1(\lambda_i) + \cdots + H_1(\lambda_m) + \sum_{i=0}^{m} [k(EK_i)]$$

is updated to

$$ssk'_{pre} = H_1(\lambda_0) + \cdots + H_1(\lambda'_i) + \cdots + H_1(\lambda_m) + \sum_{i=0}^{m} [k'(EK'_i)]$$

and ssk is updated to $ssk' = KDF(ssk||\theta'.x)$. Without knowledge of λ'_i, $\mathcal{A}$ cannot compute $\theta'.x$ and obtains ssk'. Therefore members in $\mathcal{G}$ can use ssk' to re-secure their session. This is only possible if $\mathcal{A}$ is passive after the compromise of U_i. Hence, PCS is verified under the presence of a passive adversary. □

Weak Forward secrecy. As stated in section 4.2.3, The removal of a group member

is equivalent to executing the group creation process minus that member. From theorem 2, we showed that each session key obtained after running a group creation process is unique. Therefore, it is not possible for a former member to obtain group keys generated after its removal unless it is able to break the DLPEC and CDHEC. Hence, Weak Forward Secrecy is verified.

Weak Backward secrecy. Once a new member, U_{m+1}, is added to a group, a new secret value s' is generated and shared to all members. Using s' and $\theta' = \sum_{i=0}^{m+1} H_1(\lambda_i)$, the new group key is computed as follows: $ssk' = KDF(s'||\theta'.x)$. It is possible for U_{m+1} to obtain the value $\theta = \sum_{i=0}^{m} H_1(\lambda_i) = \theta' - H_1(\lambda_{m+1})$ that was used to generate the group key $ssk = (s||\theta.x)$ before its addition. However, since s is unknown from U_{m+1}, it cannot compute ssk. Unless one of the old member was dishonest and didn't erase the value s, then U_{m+1} can collide with that member to obtain the old group key. We expect the probability of such an event to be negligible. Furthermore, from theorem 2, we know that each new key is independent from the others. Therefore, unless U_{m+1} colludes with a dishonest old member, it cannot obtain previous group keys. Hence, Weak Backward secrecy is verified.

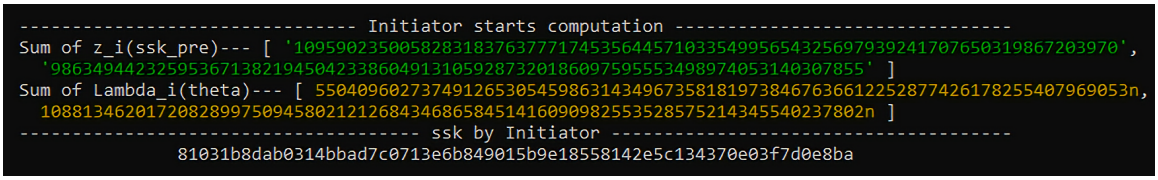
4.4 Implementation

This section demonstrates the feasibility of the proposed GKA. For this simulation, we used Rinkeby Testnet[1], (one of Ethereum's test networks).

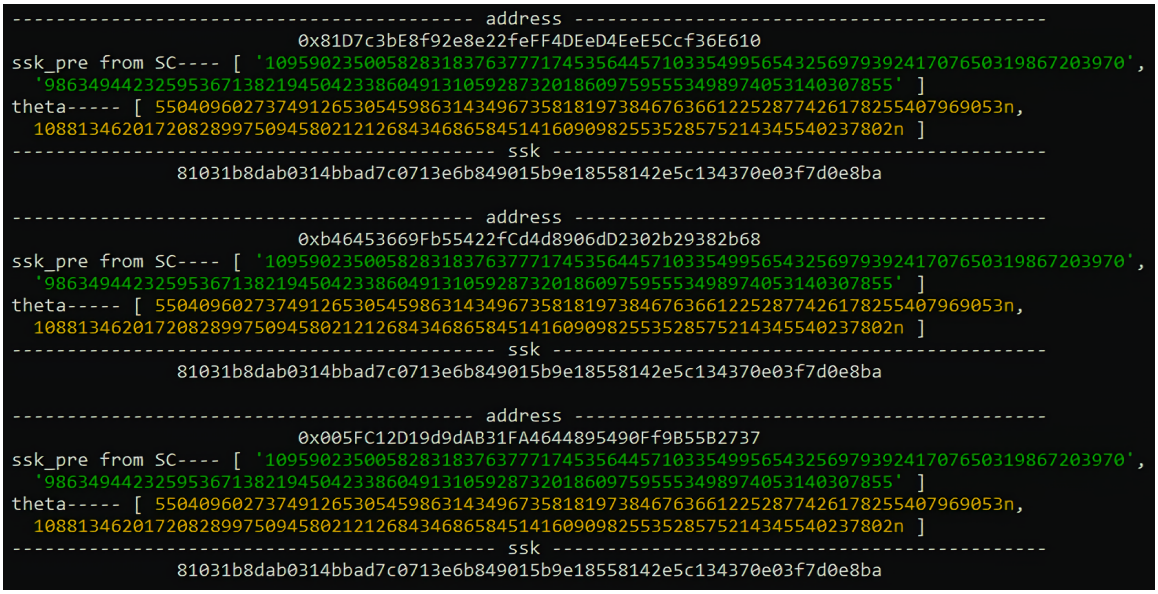
For local computation, nodes use Node.js v10.16.3 and its Crypto library[2], and Web3.js is used to interact with the blockchain. $\mathsf{F}_{\mathsf{idl}}\_\mathsf{S}_\mathsf{C}$ was written and complied using Solidity version 0.5.16. Then, it was deployed to Rinkeby Testnet[3] with Truffle[4].

The simulation environment includes a group $\mathcal{G}$ with one initiator and three responders. Each node in $\mathcal{G}$ has an account on Rinkeby Testnet with long-term keys stored in Meta-

[1] https://www.rinkeby.io/
[2] https://nodejs.org and https://nodejs.org/api/crypto.html
[3] Address: "0xC48f9bb74ebaD1EEf59967b6e2Ba245f4D37F89C"
[4] https://www.trufflesuite.com/



(a) An *ssk* initialized by an initiator



(b) The *ssk* obtained by responders

Figure 4.3: Experiment results

Mask[5]. Figure 4.3 depicts the execution results of the proposed scheme. Figure 4.3a shows *ssk* obtained by the initiator, while Figure 4.3b shows *ssk* obtained by the three responders, after executing the proposed scheme. The key values, ssk_{pre}, θ, and *ssk*, are denoted by green, yellow, and white texts, respectively. As shown in the figures, all members in $\mathcal{G}$ obtained the same ssk_{pre}, θ, and *ssk*. These results shows the validity of the proposed scheme.

To show the efficiency of the proposed scheme, we measured the gas consumption of each $\mathsf{F_{idl}\_S_C}$ functions. Deploying $\mathsf{F_{idl}\_S_C}$ on Rinkeby Testnet consumed 2,239,679 gas. However, this is a one-time cost since once $\mathsf{F_{idl}\_S_C}$ is available on the blockchain, there is no need to redeploy it. The execution of the function $\mathsf{postEphK}()$ consumes approximately 157,365 gas to post five ephemeral keys. This amount must be paid by each

[5]https://metamask.io/

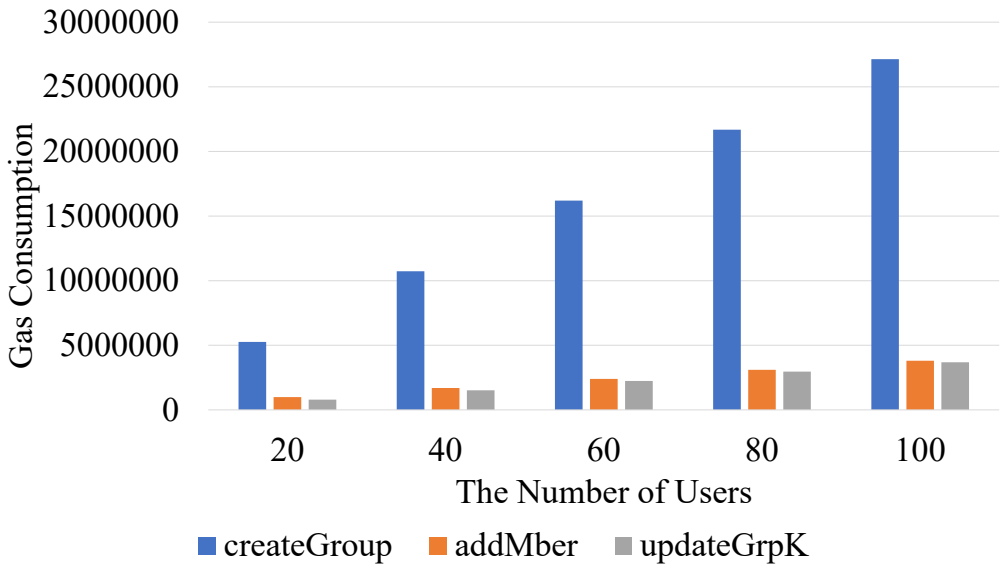


Figure 4.4: Amount of Gas Consumption in response to change in the number of users.

node on the blockchain that desires to register/update their ephemeral keys in $\mathsf{F}_{\mathsf{idl}}\_\mathsf{S}_\mathsf{C}$. The functions $\mathsf{getEphK}(), \mathsf{getPreGrpK}(), \mathsf{getKeyMaterials}()$ do not require gas to be executed since they only access data already stored in $\mathsf{F}_{\mathsf{idl}}\_\mathsf{S}_\mathsf{C}$. For the functions $\mathsf{createGroup}()$ and $\mathsf{updateGrpK}()$, their gas consumption are proportional to the size of a group. Figure 4.4 shows the gas consumption of those functions for different group size. Also, in figure 4.4, we have the gas consumption of the function $\mathsf{addMber}()$ that represents the function that we have implemented to add a new member in a group. Like $\mathsf{createGroup}()$ and $\mathsf{updateGrpK}()$, the gas consumption of $\mathsf{addMber}()$ is proportional to the size of a group. The function $\mathsf{createGroup}()$ consumes approximately 250,000 gas per user, $\mathsf{updateGrpK}()$ consumes approximately 40,000 gas per user, and $\mathsf{addMber}()$ consumes approximately 50,000 gas per user. As we can see from 4.4, in the case of a group of 20 members, $\mathsf{createGroup}()$ consumed 5,264,150 gas, $\mathsf{updateGrpK}()$ consumed 794,431 gas and $\mathsf{addMber}()$ consumed 994,178 gas. It is worth noting that for $\mathsf{createGroup}()$ and $\mathsf{addMber}()$, the cost is covered by the initiator since it is the administrator of the group. The cost for $\mathsf{updateGrpK}()$ is covered by the node that initiated the update key process. Furthermore, the member leave operation is implemented as a replay of the group creation process minus the group member that was removed.

Chapter 5

Conclusion

IoT devices present many challenges for current cryptographic systems since they are resource constrained, battery powered, prone to connection failure, and so one.

In this work, we focused on group key agreement. we presented a smart contract-based group key agreement protocol aimed at IoT environments. The proposed protocol relies on blockchain to store key materials and on smart contracts to delegate part of the computation. Therefore it does not need a trusted third party. It allows the addition and removal of members to and from a group. Also, it supports post-compromised security under a passive adversary. The security analysis showed that our proposed group key agreement is secured under DLPEC and CDHEC problems. Metrics from the implementation phase showed that our proposed protocol can be applied in real-world settings.

Future works. As potential points of improvement, we aim at looking for ways to reduce the gas consumption of smart contract's functions and design a better member removal process. In addition, we hope to port this construction in a more extensive cryptographic model such as the Universal Composability model to further analyze the security of our proposed protocol.

Bibliography

Al-Bassam, M. (2017). Scpki: A smart contract-based pki and identity system. In *Proceedings of the acm workshop on blockchain, cryptocurrencies and contracts* (pp. 35–40).

Arifi, M., Gardeshi, M., & Farash, M. S. (2012). *A new efficient authenticated id-based group key agreement protocol.* Cryptology ePrint Archive, Report 2012/395. (https://eprint.iacr.org/2012/395)

Barker, E. (2015). *Recommendation for key management –part 1: General(revision 3).* Retrieved from http://dx.doi.org/10.6028/NIST.SP.800-57pt1r4

Barnes, R., Beurdouche, B., Millican, J., Omara, E., Cohn-Gordon, K., & Robert, R. (2020, March 6). *The Messaging Layer Security (MLS) Protocol* (Internet-Draft No. draft-ietf-mls-protocol-09). Internet Engineering Task Force. Retrieved from https://datatracker.ietf.org/doc/html/draft-ietf-mls-protocol-09 (Work in Progress)

Blake, I., Seroussi, G., & Smart, N. (1999). *Elliptic curves in cryptography*. Cambridge University Press. doi: 10.1017/CBO9781107360211

Buterin, V. (2014). A next generation smart contract & decentralized application platform. *Ethereum White Paper*, 1-36. Retrieved from https://cryptorating.eu/whitepapers/Ethereum/Ethereu_white_paper.pdf

Choi, K. Y., Hwang, J. Y., & Lee, D. H. (2004). Efficient id-based group key agreement with bilinear maps. In F. Bao, R. Deng, & J. Zhou (Eds.), *Public key cryptography –*

pkc 2004 (pp. 130–144). Berlin, Heidelberg: Springer Berlin Heidelberg.

Cohn-Gordon, K., Cremers, C., & Garratt, L. (2016). On post-compromise security. In *2016 ieee 29th computer security foundations symposium (csf)* (p. 164-178).

Cohn-Gordon, K., Cremers, C., Garratt, L., Millican, J., & Milner, K. (2018). On ends-to-ends encryption: Asynchronous group messaging with strong security guarantees. In *Proceedings of acm sigsac conference on computer and communications security (ccs)* (p. 1802–1819).

Daswani, N., & Boneh, D. (1999). Experimenting with electronic commerce on the palmpilot. In *Proceedings of the third international conference on financial cryptography* (p. 1–16). Berlin, Heidelberg: Springer-Verlag.

Denis, V. (2018). *The different types of blockchains.* Retrieved from `https://medium.com/the-capital/the-different-types-of-blockchains-456968398559` (Accessed: 2020-02-8)

Diffie, W., & Hellman, M. (1976). New directions in cryptography. *IEEE Transactions on Information Theory*, *22*(6), 644-654.

Eosio-cleos/cleos-set. (n.d.). `https://developers.eos.io/manuals/eos/latest/cleos/command-reference/set/set-contract`. (Accessed: 2019-07-10)

Hu, Y., Xiong, Y., Huang, W., & Bao, X. (2018). Keychain: Blockchain-based key distribution. In *Proceedings of the 4th international conference on big data computing and communications (bigcom)* (pp. 126–131).

Islam, S. H., Obaidat, M. S., Vijayakumar, P., Abdulhay, E., Li, F., & Reddy, M. (2018). A robust and efficient password-based conditional privacy preserving authentication and group-key agreement protocol for vanets. *Future Generation Computer Systems*, *84*, 216 - 227.

Kangath, V. (2018). *Iot is everywhere — how iot is changing our daily lives.* Retrieved from `https://www.ness.com/iot-is-everywhere-how-iot-is-changing-our-daily-lives/`

Kim, D., Wang, W., Son, J., Wu, W., Lee, W., & Tokuta, A. O. (2017). Maximum lifetime combined barrier-coverage of weak static sensors and strong mobile sensors. *IEEE Transactions on Mobile Computing*, *16*(7), 1956-1966.

Kim, Y., Perrig, A., & Tsudik, G. (2004a, July). Group key agreement efficient in communication. *IEEE Transactions on Computers*, *53*(7), 905-921.

Kim, Y., Perrig, A., & Tsudik, G. (2004b, February). Tree-based group key agreement. *ACM Trans. Inf. Syst. Secur.*, *7*(1), 60–96. Retrieved from https://doi.org/10.1145/984334.984337 doi: 10.1145/984334.984337

Krawczyk, H. (2010, August). Cryptographic extraction and key derivation: The hkdf scheme. In *Proceedings of advances in cryptology-crypto 2010* (p. 631-648).

Marlinspike, M., & Perrin, T. (2016). The x3dh key agreement protocol. , 1–11. Retrieved from https://https://www.signal.org/docs/specifications/x3dh/x3dh.pdf

Meneghello, F., Calore, M., Zucchetto, D., Polese, M., & Zanella, A. (2019). Iot: Internet of threats? a survey of practical security vulnerabilities in real iot devices. *IEEE Internet of Things Journal*, *6*(5), 8182-8201.

Menezes, A. J., Oorschot, P. C. v., & Vanstone, S. A. (1996). *Handbook of applied cryptography* (1th ed.). CRC Press.

Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system. , 1–9. Retrieved from https://bitcoin.org/bitcoin.pdf

Neocontract white paper. (n.d.). https://docs.neo.org/docs/en-us/basic/technology/neocontract.html. (Accessed: 2019-10-24)

Nguyen, G.-T., & Kim, K. (2018). A survey about consensus algorithms used in blockchain. *JIPS*, *14*, 101-128.

Pass, R., & Shi, E. (2016). *Hybrid consensus: Efficient consensus in the permissionless model.* Cryptology ePrint Archive, Report 2016/917. (https://eprint.iacr.org/2016/917)

Patsonakis, C., Samari, K., Kiayiasy, A., & Roussopoulos, M. (2019, April). On the practicality of a smart contract pki. In *Proceedings of ieee international conference on decentralized applications and infrastructures (dappcon)* (p. 109-118). doi: 10 .1109/DAPPCON.2019.00022

Pease, M., Shostak, R., & Lamport, L. (1980). Reaching Agreement in the Presence of Faults. *Journal of the ACM*, *27*(2), 228–234. doi: 10.1145/322186.322188

Rahimi, P., & Chrysostomou, C. (2019). Improving the network lifetime and performance of wireless sensor networks for iot applications based on fuzzy logic. In *2019 15th international conference on distributed computing in sensor systems (dcoss)* (p. 667-674).

Schindler, P., Judmayer, A., Stifter, N., & Weippl, E. (2019). *Ethdkg: Distributed key generation with ethereum smart contracts.* Cryptology ePrint Archive, Report 2019/985. (`https://eprint.iacr.org/2019/985`)

Singla, A., & Bertino, E. (2018). Blockchain-based pki solutions for iot. In *Proceedings of ieee 4th international conference on collaboration and internet computing (cic)* (pp. 9–15).

Sivakumar, P., & Singh, K. (2017). Privacy based decentralized public key infrastructure (pki) implementation using smart contract in blockchain. In *Proceedings of the 2nd advanced workshop on blockchain: Technology, applications, challenges* (p. 1-6).

Stallings, W. (2017). *Cryptography and network security: Principles and practice* (7th ed.). Pearson Education.

Veltri, L., Cirani, S., Busanelli, S., & Ferrari, G. (2013). A novel batch-based group key management protocol applied to the internet of things. *Ad Hoc Networks*, *11*(8), 2724 - 2737.

Washington, L. C. (2008). *Elliptic curves: Number theory and cryptography* (2th ed.). CRC Press.

Wood, G. (2019). Ethereum: A secure decentralised generalised transaction ledgerbyzan-

tium version 7e819ec - 2019-10-20. , 1–39. Retrieved from https://ethereum.github.io/yellowpaper/paper.pdf

Wu, Y., Song, P., & Wang, F. (2020). Hybrid consensus algorithm optimization: A mathematical method based on pos and pbft and its application in blockchain. *Mathematical Problems in Engineering*, 1–13. Retrieved from https://doi.org/10.1155/2020/7270624 doi: 10.1155/2020/7270624

Yao, H., & Wang, C. (2018). A novel blockchain-based authenticated key exchange protocol and its applications. In *Proceedings of ieee third international conference on data science in cyberspace (dsc)* (pp. 609–614).

Youdom Kemmoe, V., Stone, W., Kim, J., Kim, D., & Son, J. (2020). Recent advances in smart contracts: A technical overview and state of the art. *IEEE Access*, 1-20.

Zhang, L., Wu, Q., Domingo-Ferrer, J., Qin, B., & Dong, Z. (2015). Round-efficient and sender-unrestricted dynamic group key agreement protocol for secure group communications. *IEEE Transactions on Information Forensics and Security*, *10*(11), 2352-2364.

Zhang, Q., Gan, Y., Liu, L., Wang, X., Luo, X., & Li, Y. (2018). An authenticated asymmetric group key agreement based on attribute encryption. *Journal of Network and Computer Applications*, *123*, 1 - 10. Retrieved from http://www.sciencedirect.com/science/article/pii/S1084804518302704 doi: https://doi.org/10.1016/j.jnca.2018.08.013

Appendices

Appendix A

Glossary

We provide a brief definition to some of the terms used in this work.

Active adversary: is an adversary that can tamper with a communication channel. For instance, it can inject messages or interfere with the transmission of messages issued by rightful parties.

Ephemeral key: is a cryptographic key that is short-lived. It is generated to be used in a specific key establishment procedure, and after usage, it should be discarded.

Gas (*Ethereum*): is a metric used by the Ethereum platform to evaluate the cost of executing a transaction.

Long-term key: is a cryptographic key that is supposed to be used over a long period of time. For instance, in the case of a device, such a key can be used throughout the lifetime of that device.

Passive adversary: is an adversary that can only listen to messages exchanged in a communication channel.