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

Victor Youdom Kemmoe*, Yongseok Kwon*, Seunghyeon Shin*,
Rasheed Hussain†, Sunghyun Cho‡, and Junggab Son*
* Dept. of Computer Science, Kennesaw State University, USA. Email: vyoudomk@students.kennesaw.edu
† Networks and Blockchain Lab, Innopolis University, Innopolis, Russia. Email: r.hussain@innopolis.ru
‡ Dept. of Computer Science and Engineering, Hanyang University, South Korea, Email: chopro@hanyang.ac.kr

***Abstract*—Group Key Agreement (GKA) mechanisms play a crucial role in realizing various applications in different networks, such as sensor networks and the Internet of Things (IoT). To be suitable for IoT, a GKA must satisfy several critical requirements. First, a GKA 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 to establish ad hoc networks securely, and usually, these devices are resource-constrained. Second, the GKA must be able to distribute session keys successfully, even with offline devices. Third, a GKA must reduce the burden of heavy cryptographic computations for IoT devices. Based on these observations, we propose a new GKA scheme that satisfies all the requirements above. The proposed scheme leverages smart contracts to alleviate the computational and storage overheads on 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.**

***Index Terms*—Blockchain, Smart Contracts, Cryptography, Secure communication, Group Key Agreement, Internet of Things**

I. Introduction

Secure group communication is one of the highest priority tasks in critical environments such as sensor networks, Internet of Things (IoT), and so on [1]. By using a proper Group Key Agreement (GKA) scheme, group members can establish a secure channel to exchange valuable information with each other. As the complexity and heterogeneity of networking environments increase, GKA schemes encounter more critical challenges than traditional networks. For instance, IoT devices are generally resource-constrained and battery-powered. Therefore, it is easier for an adversary to compromise them [2]. In this regard, GKA schemes for IoT require post-compromise security, such that a compromised device should still be able to establish a secure communication channel against the adversary [3]. Also, IoT applications often adapt a power management system that can put some devices in hibernation mode to maximize the lifetime of networks consisting of battery-powered devices. In this regard, the underlying GKA mechanism should be functional even with offline devices. In addition, executing heavy cryptographic computations on an IoT device can shorten its lifetime. In this regard, a GKA scheme could support delegated operations of those functions to reduce the computational burden of the devices.

This research was supported by the MSIT (Ministry of Science, ICT), Korea, under the High-Potential Individuals Global Training Program (2019-0-01601) supervised by the IITP (Institute for Information & Communications Technology Planning & Evaluation). Corresponding Author: Junggab Son (Email: json@kennesaw.edu)

Many existing GKA schemes assume the existence of a Trusted Third Party (TTP) and do not support post-compromise security, a few instances are [4]–[6]. Hence, such schemes are impractical for IoT environments. Although blockchain was initially developed for cryptocurrency systems by actualizing the concept of distributed and immutable ledger of transactions [7], it can be used as an alternative solution to TTP due to its traits such as transactions in a block cannot be easily tampered. Moreover, smart contracts in a blockchain allow us to deploy executable codes on the blockchain [8]. Thus, the blockchain's attributes can help in establishing a trusted execution environment (which is currently missing).

To fill the gaps, this paper aims to apply a smart contract-based mechanism to enable secure and asynchronous GKA for IoT environment. The major contributions are summarized below:

- System and security requirements of GKA schemes based on smart contracts are defined,
- A new asynchronous GKA scheme based on smart contract is proposed
- Feasibility of the proposed scheme is demonstrated by implementing it on the Ethereum platform.
- The security analysis shows that the proposed scheme satisfies the defined security model.

II. Background and Preliminaries

A. Blockchains and Smart Contracts

A blockchain is a distributed data structure consisting of consecutive blocks of transactions. Each block of transactions is linked to the previous block through a hashed value. A blockchain is maintained by a set of independent nodes on a computer network and has an access mode that dictates which node can interact with it. A new block is appended after it is validated by nodes through a consensus mechanism. The appended blocks cannot be deleted or modified. A node may deny an issued transaction or pass it as the transaction of another node. To prevent this, blockchains use public-key cryptography. Each node is required to sign each transaction it issues with its private key.

A *smart contract* is a set of instructions stored in a blockchain that is executed whenever a transaction, that referenced it, is sent to the blockchain. A smart contract can be stateless (it only receives data, processes the data, and returns the results) or stateful (additionally, it has access to an internal memory that can store variables). In the case of stateful smart contracts, the *state* is stored on the blockchain, and it can be updated after executing the smart contract. Furthermore, since elements stored on a blockchain cannot be deleted, one can access previous states of a smart contract by navigating the blockchain. To this end, the combination of a blockchain and a smart contract can allow the development of decentralized and tamper-proof applications.

B. Preliminaries

Let $E/\mathbb{F}_q$ denotes an elliptic curve E defined over a finite field $\mathbb{F}_q$, where q is a large prime. E is defined by the following equation: $E : y^2 = x^3 + ax + b$ where $a, b \in \mathbb{F}_q$. $E(\mathbb{F}_q)$ is a subgroup of points (x, y) on E such that $(x, y) \in \mathbb{F}_q \times \mathbb{F}_q$. We assume the following two problems to be intractable:

Elliptic Curve Discrete Logarithm Problem (ECDLP). Given $P, Q \in E(\mathbb{F}_q)$, it is difficult for any Probabilistic Polynomial Time (PPT) adversary $\mathcal{A}$ to find a value $m \in \mathbb{Z}_n^*$ such that $Q = mP$.

Computational Diffie-Hellman over Elliptic Curve (CDHEC). Given $(a.P, b.P) \in E(\mathbb{F}_q)$ with $a, b \in \mathbb{Z}_n^*$. It is difficult for any PPT adversary $\mathcal{A}$ to compute $ab.P$.

C. ElGamal encryption over Elliptic curve

Let $P \in E(\mathbb{F}_q)$ be a base point of $E/\mathbb{F}_q$ with order n. Following the description of ElGamal [9], the ElGamal encryption over $E(\mathbb{F}_q)$ is an encryption scheme made of three polynomial-time algorithms $(\mathcal{K}, \mathcal{E}, \mathcal{D})$:

- **Key generation** $\mathcal{K}$: 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.
- **Encryption** $\mathcal{E}$: To send a message $M \in E(\mathbb{F}_q)$ to Alice, Bob randomly selects a secret $r \in \mathbb{Z}_n^*$, then it computes $C' = r.P$ and $C = r.A + M$. The cipher text is (C', C).
- **Decryption** $\mathcal{D}$: To obtain the message M from (C', C), Alice computes $K = a.C'$, and after, subtracts K from C: $M = C - K = (r.a.P + M) - (r.a.P)$

Homomorphism. An encryption scheme $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ is homomorphic if it accepts an operator $\circ$, such that $\forall m_1, m_2 \in \mathcal{M}$ the following equation is verified: $\mathcal{E}(m_1 \circ m_2) = \mathcal{E}(m_1) \circ \mathcal{E}(m_2)$, 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 key pair (a, A), two plaintexts m_1, m_2, and some random numbers $r_1, r_2 \in \mathbb{Z}_n^*$, $r = r_1 + r_2$, the homomorphic property is then:

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

D. X3DH protocol

Marlinspike *et al.* proposed extended triple Diffie-Hellman (X3DH) protocol which enables an asynchronous key exchange between two parties [10]. It uses asymmetric key defined over $E/\mathbb{F}_q$. Let us consider two parties Alice and Bob. Alice has an identity key pair (ik_a, IK_a). 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 $(ik_a, IK_b, ek_a, SK_b, OK_b)$:

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

where KDF is a Key Derivation Function which can be implemented using a cryptographic hash function [11]. 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:

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

In this work, we replaced the concept of signed pre-key with ephemeral key, and we consider long-term keys to be equivalent to identity keys.

III. System and Security Models

A. System Model

Ideal Blockchain $\mathcal{F}_{idl}$. We assume that our system model includes an Ideal Blockchain $\mathcal{F}_{idl}$ whose traits are defined as follows:

- $\mathcal{F}_{idl}$ uses an elliptic curve $E/\mathbb{F}_q$ where the ECDLP and the CDHEC are intractable in the subgroup $E(\mathbb{F}_q)$.
- A node U_i that interact with $\mathcal{F}_{idl}$ possesses an account that holds a long-term key pair (lk_i, LK_i).
- $\mathcal{F}_{idl}$ supports stateful smart contracts defined using a Turing-complete programming language.
- For enhanced security, we assume the access mode of $\mathcal{F}_{idl}$ to be public [12].

Smart Contract $\mathcal{F}_{idl}\_\mathcal{S}_C$. We assume the existence of a stateful smart contract hosted on $\mathcal{F}_{idl}$, denoted by $\mathcal{F}_{idl}\_\mathcal{S}_C$.

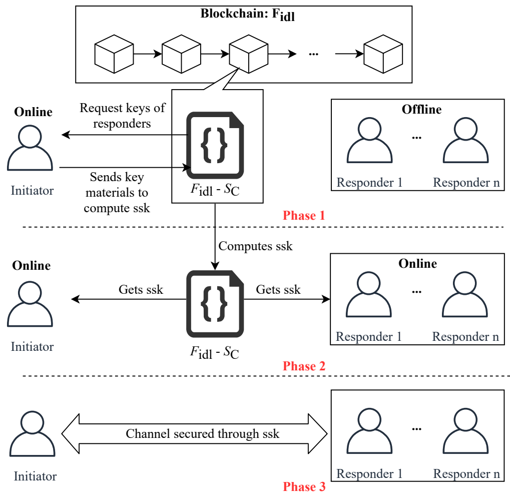

Blockchain: $\mathrm{F_{idl}}$
...
Online
Request keys of responders
Sends key materials to compute ssk
Initiator
$\mathrm{F_{idl}}$ - $\mathrm{S_C}$
Offline
Responder 1
Responder n
Phase 1
Computes ssk
Online
Gets ssk
Gets ssk
Online
Initiator
$\mathrm{F_{idl}}$ - $\mathrm{S_C}$
Responder 1
Responder n
Phase 2
Channel secured through ssk
Initiator
Responder 1
Responder n
Phase 3

Fig. 1: Abstract representation of the proposed scheme.

Participants. We have participants who use $\mathcal{F}_{idl}\_\mathcal{S}_C$ to exchange a group key. Let $\mathcal{U}$ be the set of all nodes with a registered account in $\mathcal{F}_{idl}$. To participate in the protocol, a node $U_i \in \mathcal{U}$ also needs to have a set of ephemeral keys stored in $\mathcal{F}_{idl}\_\mathcal{S}_C$. From this, we denote $\mathcal{U}_r \subseteq \mathcal{U}$ as the set of all nodes in $\mathcal{F}_{idl}$ having ephemeral keys stored in $\mathcal{F}_{idl}\_\mathcal{S}_C$. In $\mathcal{U}_r$, we identify two types of nodes:

- Initiator: it is a node denoted by $U_0 \in \mathcal{U}_r$ that wants to create a group $\mathcal{G}$ with a subset R, and exchange a group key. It has the role of group administrator and is in charge of adding and removing group members. U_0 needs to be online during the execution of the proposed scheme.
- 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 gets the long-term and ephemeral public keys of responders from $\mathcal{F}_{idl}\_\mathcal{S}_C$. Then, it uses those keys to compute and transfer key materials to $\mathcal{F}_{idl}\_\mathcal{S}_C$, which will be used to devise the group key. During phase two, once $\mathcal{F}_{idl}\_\mathcal{S}_C$ 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 1 provides a graphical representation of those phases.

B. Security Model

Our proposed GKA solution is based on the following security notions.

- **Key Independence**: an attacker that knows a set of group keys cannot deduce other group keys.
- **Perfect Forward Secrecy (PFS)**: previously computed group keys should remain out of reach for a PPT adversary that is able to retrieve the long-term private keys of some or all members of a group.
- **Post-Compromise Security (PCS)**: following the definition of Conh-Gordon *et al.* [3], members in a group $\mathcal{G}$ should still be able to establish a secure communication channel against a compromiser even if one of their devices was compromised.

IV. Proposed Group Key Agreement Mechanism

In this section, we discuss the proposed smart contract-based GKA in detail. First, we describe some important notations that we are going to use:

- $P.x$: Given $E/\mathbb{F}_q$, we denote by $P.x$ the x-coordinate of a point $P \in E(\mathbb{F}_q)$
- $h \in_R G$: the variable h is randomly and uniformly selected from the set G
- (k, K): For a private key $k \in \mathbb{Z}_n^*$, we denote its corresponding public key as $K = k.P$, with $K, P \in E(\mathbb{F}_q)$
- $|G|$: denotes the number of elements in a set G
- $a||b$: denotes a concatenation of a and b

A. Function Definition

The following functions need to be implemented as essential components of the proposed scheme and are a part of $\mathcal{F}_{idl}\_\mathcal{S}_C$.

- **postEphK**(Y_i): A node $U_i \in \mathcal{U}$ uploads a set of ephemeral keys Y_i to $\mathcal{F}_{idl}\_\mathcal{S}_C$. If there is already a set Y_i stored in $\mathcal{F}_{idl}\_\mathcal{S}_C$, executing *postEphK* will overwrite Y_i with the new values.
- **getEphK**(LK_i): it returns an unused ephemeral key EK_i from the set Y_i stored in $\mathcal{F}_{idl}\_\mathcal{S}_C$ of a node U_i with public key LK_i.
- **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 $\mathcal{F}_{idl}\_\mathcal{S}_C$. In addition, it generates and returns $\mathcal{G}$.ID, which is the ID of the group, to all members of $\mathcal{G}$.
- **getPreGrpKey**($\mathcal{G}$.ID): this function is triggered by a node $U_i \in \mathcal{G}$ with ID $= \mathcal{G}$.ID. It returns the pre-group key of the group $\mathcal{G}$ to the U_i.
- **getKeyMaterials**($\mathcal{G}$.ID): this function is triggered by a node $U_i \in \mathcal{G}$ with ID $= \mathcal{G}$.ID. It returns the set $K, \Phi, \mathcal{Y}$ to the U_i.
- **updateGrpK**($\mathcal{G}$.ID$, K, \mathcal{Y}, \alpha$): this function is triggered by a node $U_i \in \mathcal{G}$ with ID $= \mathcal{G}$.ID. It updates the pre-group key ssk_{pre} using α and replaces K and $\mathcal{Y}$ stored in $\mathcal{F}_{idl}\_\mathcal{S}_C$ with the ones passed as arguments.

B. Setup

It instantiate a local elliptic curve group based on the parameters of $E/\mathbb{F}_q$ used by $\mathcal{F}_{idl}$.Then, 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.

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 $1 \le j \le w$, $ek_{ij} \in_R \mathbb{Z}_n^*$ and

Algorithm 1: Initiation of Group

```
1  Input: $\mathcal{G} = \{U_0\} \cup R$
2  Require: $Z[0:m]$, $\mathcal{Y}[0:m]$, $K[1:m]$, $\Phi[0:m]$
3  begin
4      Select $\lambda_0, s \in_R \mathbb{F}_q$ and $k \in_R \mathbb{Z}_n^*$
5      Select a non-used ephemeral key pair $(ek_0, EK_0)$
         $Z[0] \leftarrow H_1(\lambda_0) + k \times EK_0$
6      $\mathcal{Y}[0] \leftarrow EK_0$
7      for $i \leftarrow 1$ to $m$ do
8          $\mathcal{Y}[i] \leftarrow \mathcal{F}_{idl}\_\mathcal{S}_C$.getEphK($LK_i$)
9          $\lambda \leftarrow$ X3DH($lk_0, LK_i, ek_0, \mathcal{Y}[i]$)
10         $Z[i] \leftarrow H_1(\lambda) + k \times \mathcal{Y}[i]$
11         $K[i] \leftarrow k \oplus H_2(\lambda)$,
12         $\Phi[i] \leftarrow s \oplus H_2(\lambda || \mathcal{Y}[i].x)$
13         Delete $\lambda$
14     end
15     Send transaction
         $\mathcal{F}_{idl}\_\mathcal{S}_C$.createGroup($\mathcal{G}, Z, K, \Phi, \mathcal{Y}$) to $\mathcal{F}_{idl}$
16 end
```

Algorithm 2: Acquisition of Group Key

```
   Input  : idx //the index of node in $\mathcal{G}$
   Return: $ssk$ //The group key
1  begin
2      $K, \Phi, \mathcal{Y} \leftarrow \mathcal{F}_{idl}\_\mathcal{S}_C$.getKeyMaterials($\mathcal{G}$.ID)
3      Get $EK_{idx}$ and $EK_0$ from $\mathcal{Y}$
4      $\lambda \leftarrow$ 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      $ssk_{pre} \leftarrow \mathcal{F}_{idl}\_\mathcal{S}_C$.getPreGrpKey($\mathcal{G}$.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     $ssk \leftarrow$ KDF($s || \theta.x$)
15     Delete $\theta, s$
16     return $ssk$
17 end
```

$EK_{ij} = ek_{ij}.P$. Then U_i posts Y_i to $\mathcal{F}_{idl}\_\mathcal{S}_C$ by sending the transaction $\mathcal{F}_{idl}\_\mathcal{S}_C$.postEphK($Y_i$) to $\mathcal{F}_{idl}$. It is worth nothing that EP_i should be frequently replaced.

C. Initiation

During the initiation phase, the initiator U_0 computes the essential key materials that will be used to generate the group key and sends those to $\mathcal{F}_{idl}\_\mathcal{S}_C$. 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 $\mathcal{F}_{idl}\_\mathcal{S}_C$ 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 $\mathcal{F}_{idl}\_\mathcal{S}_C$.createGroup() to $\mathcal{F}_{idl}$.

D. Smart Contract Computation

Once U_0 terminates the initiation process, the function $\mathcal{F}_{idl}\_\mathcal{S}_C$.createGroup() is executed by $\mathcal{F}_{idl}$. 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: $ssk_{pre} = \sum_{i=0}^{m}[H_1(\lambda_i) + k(EK_i)]$

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 $\mathcal{G}$.

E. Acquisition of the group key

Once the Smart Contract Computation step is terminated, responders can derive the group key by executing Algorithm 2. Firstly, each $U_i \in R$ needs to get the key materials from $\mathcal{F}_{idl}\_\mathcal{S}_C$. 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 $\mathcal{F}_{idl}\_\mathcal{S}_C$ and compute θ as follows: $\theta = \sum_{i=0}^{m} H_1(\lambda_i)$.

Finally, using KDF, the responders obtain ssk from θ and s. Then, they delete θ and s. The initiator also executes Algorithm 2 to get the group key. However, it starts the execution at line 7 because it already has the key materials stored in local memory.

F. Group Key Update

To satisfy PCS's requirements, our scheme needs to provide a way 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 justified by Cohn-Gordon *et al.* [3].

The key update process is two-fold. Let $U_{idx} \in \mathcal{G}$ be a node that wants to update the group key. Firstly, U_{idx} executes the Initiate Key Update process outlined in Algorithm 3. 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 $\mathcal{F}_{idl}\_\mathcal{S}_C$. Then, using X3DH protocol, it computes λ' for others. After that, it encrypts k' using λ'. Finally it computes α and sends the transaction $\mathcal{F}_{idl}\_\mathcal{S}_C$.updateGrpK() to $\mathcal{F}_{idl}$. 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 3 and EK_i represents the old ephemeral keys.

Secondly, once $\mathcal{F}_{idl}$ receives $\mathcal{F}_{idl}\_\mathcal{S}_C$.updateGrpK(), it updates ssk_{pre} to ssk'_{pre} by computing $ssk_{pre} + \alpha$:

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

After this, other nodes in $\mathcal{G}$ can obtain the new group key according to Algorithm 4. Each node gets the new key materials from $\mathcal{F}_{idl}\_\mathcal{S}_C$, and using X3DH protocol, it computes λ'. Then,

Algorithm 3: 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
        $\mathcal{Y}[i] \leftarrow \mathcal{F}_{idl}\_\mathcal{S}_C$.getEphK$(LK_i)$
        $\lambda' \leftarrow$ X3DH$(lk_{idx}, LK_i, ek_{idx}, \mathcal{Y}[i])$
        $K'[i] \leftarrow k' \oplus H_2(\lambda)$
        Delete $\lambda'$
    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
      $\mathcal{F}_{idl}\_\mathcal{S}_C$.updateGrpK$(\mathcal{G}$.ID$, K', \mathcal{Y}, \alpha)$ to $\mathcal{F}_{idl}$
end
```

each node deciphers the encrypted value of k' by using $H_1(\lambda')$. Next, each node gets ssk'_{pre} from $\mathcal{F}_{idl}\_\mathcal{S}_C$, 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 4, but starting at line 5 since it already has the new key materials stored in local memory.

V. Security Analysis

Man-in-the-Middle attack: Given the current state of our scheme, 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 impersonate U_i from the point of view of U_0. To prevent this attack, one can use a Decentralized Public Key Infrastructure (DPKI) on top of our scheme to authenticate public keys. For instance, a DPKI based blockchain can be suitable since it can be directly incorporated with 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 2, $ssk = \text{KDF}(s||\theta.x)$. 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 ECDLP, $\mathcal{A}$ cannot find k and compute $w = \sum_{i=0}^{m}[k(EK_i)]$ from $\mathcal{Y}$. Without w, $\mathcal{A}$ 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. □

Algorithm 4: Acquisition of New Key

```
Input  : j //the index of node in $\mathcal{G}$ getting in the new key
Return: $ssk'$//the new group key
begin
    $K', \Phi, \mathcal{Y} \leftarrow \mathcal{F}_{idl}\_\mathcal{S}_C$.getKeyMaterials$(\mathcal{G}$.ID$)$
    $\lambda' \leftarrow$ X3DH$(lk_j, LK_{idx}, ek_j, EK_{idx})$
    $k' \leftarrow K'[j] \oplus H_2(\lambda)$
    $ssk'_{pre} \leftarrow \mathcal{F}_{idl}\_\mathcal{S}_C$.getPreGrpKey$(\mathcal{G}$.ID$)$
    $w' \leftarrow 0$
    for $i \leftarrow 0$ to $(|\mathcal{G}|)$ do
        $w' \leftarrow w' + \mathcal{Y}[i]$
    end
    $w' \leftarrow k' \times w'$
    $\theta' \leftarrow ssk'_{pre} - w'$
    $ssk' \leftarrow$ KDF$(ssk||\theta'.x)$
    Delete $\theta'$
    Return: $ssk'$
end
```

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. 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 variables generated uniformly at random, and s is generated uniformly at random, ssk is random. Furthermore, since a KDF is collision-resistant [11], ssk is unique. Thus, all keys in $\mathcal{D}$ are uncorrelated. Therefore, $\mathcal{A}$ cannot use keys in $\mathcal{D}$ to reveal ssk_{n+1} or ssk_0.

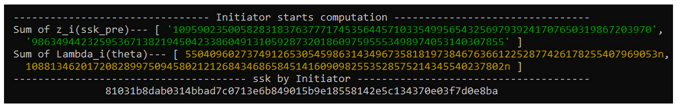
Second, let's suppose some keys in $\mathcal{D}$ were derived after a key update, i.e., $ssk_{i+1} = \text{KDF}(ssk_i||\theta'.x)$. In this case, knowing ssk_i is not sufficient to reveal ssk_{i+1}.

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.*

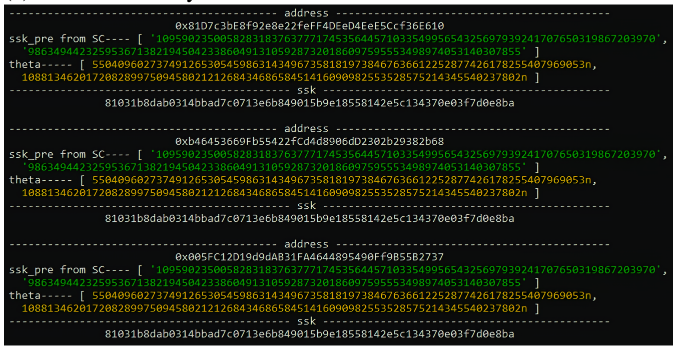
Proof. Once U_i is compromised, it is possible for $\mathcal{A}$ to obtains $k, \lambda_i,$ and ssk. However, if after the compromised U_i performs the Update Key process,

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



```
------------------------------- Initiator starts computation --------------------------------
Sum of z_i(ssk_pre)--- [ '10959023500582831837637771745356445710335499565432569793924170765031986720397 0',
  '98634944232595367138219450423386049131059287320186097595553498974053140307855' ]
Sum of Lambda_i(theta)--- [ 55040960273749126530545986314349673581819738467636612252877426178255407969053n,
  108813462017208289975094580212126843468658451416090982553528575214345540237802n ]
------------------------------------- ssk by Initiator --------------------------------------
                81031b8dab0314bbad7c0713e6b849015b9e18558142e5c134370e03f7d0e8ba
```

(a) ssk obtained by the initiator



```
------------------------------------------ address ------------------------------------------
                          0x81D7c3bE8f92e8e22feFF4DEeD4EeE5Ccf36E610
ssk_pre from SC---- [ '109590235005828318376377717453564457103354995654325697939241707650319867203970',
  '98634944232595367138219450423386049131059287320186097595553498974053140307855' ]
theta----- [ 55040960273749126530545986314349673581819738467636612252877426178255407969053n,
  108813462017208289975094580212126843468658451416090982553528575214345540237802n ]
-------------------------------------------- ssk --------------------------------------------
                81031b8dab0314bbad7c0713e6b849015b9e18558142e5c134370e03f7d0e8ba

------------------------------------------ address ------------------------------------------
                          0xb46453669Fb55422fCd4d8906dD2302b29382b68
ssk_pre from SC---- [ '109590235005828318376377717453564457103354995654325697939241707650319867203970',
  '98634944232595367138219450423386049131059287320186097595553498974053140307855' ]
theta----- [ 55040960273749126530545986314349673581819738467636612252877426178255407969053n,
  108813462017208289975094580212126843468658451416090982553528575214345540237802n ]
-------------------------------------------- ssk --------------------------------------------
                81031b8dab0314bbad7c0713e6b849015b9e18558142e5c134370e03f7d0e8ba

------------------------------------------ address ------------------------------------------
                          0x005FC12D19d9dAB31FA4644895490Ff9B55B2737
ssk_pre from SC---- [ '109590235005828318376377717453564457103354995654325697939241707650319867203970',
  '98634944232595367138219450423386049131059287320186097595553498974053140307855' ]
theta----- [ 55040960273749126530545986314349673581819738467636612252877426178255407969053n,
  108813462017208289975094580212126843468658451416090982553528575214345540237802n ]
-------------------------------------------- ssk --------------------------------------------
                81031b8dab0314bbad7c0713e6b849015b9e18558142e5c134370e03f7d0e8ba
```

(b) ssk obtained by responders

Fig. 2: Experiment results

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. $\square$

VI. Experimental Results and Discussion

For the simulation, we used Rinkeby Testnet[1], an Ethereum's testnet. Due to the page limit, we only present the group creation and the key update process in this paper. However, we have successfully implemented a join (add member) and a leave (remove member) operations as well. A non-compiled version of $\mathcal{F}_{idl}\_\mathcal{S}_C$ is available at [13]. $\mathcal{F}_{idl}\_\mathcal{S}_C$ was written and compiled using Solidity v0.5.16. Then, it was deployed on Rinkeby[2]. Nodes used Web3.js to interact with Rinkeby and Node.js v10.16.3 for local computations.

Figure 2 shows the result after the execution of the group creation process between an initiator and three responders. As we can see, all members obtained the same group key ssk. Figure 3 shows the gas consumption of $\mathcal{F}_{idl}\_\mathcal{S}_C$'s functions for different group size. *postEphK* gas consumption varies in function of the number of keys posted. The member addition function is implemented as *addMber*. The functions *getEphK, getPreGrpKey, getKeyMaterials* do not consume gas since they only read data stored in $\mathcal{F}_{idl}\_\mathcal{S}_C$.The member leave operation is implemented as a replay of the group creation process minus the group member that was removed.

[1]https://www.rinkeby.io/

[2]Address: ”0xC48f9bb74ebaD1EEf59967b6e2Ba245f4D37F89C”

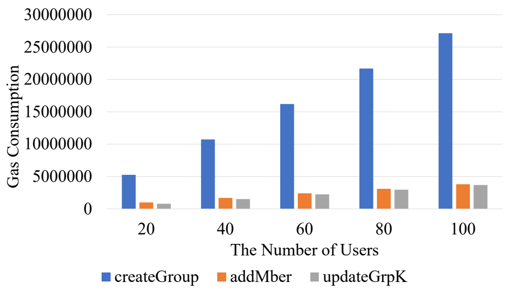


Fig. 3: Amount of Gas Consumption in response to change in the number of users.

VII. Conclusion

In this paper, we proposed a smart contract-based asynchronous group key agreement mechanism for IoT. The proposed scheme does not need a TTP, and it can support efficient group member addition and removal. Additionally, it provides post-compromised security under a passive attacker. We proved that the proposed scheme is secure under the ECDLP and CDHEC problems. The result of our experiments demonstrates that the proposed scheme is highly practical. As future work, we aim to reduce the gas consumption and provide better efficiency.

References

[1] C.-S. Park, “Security architecture for secure multicast coap applications,” *IEEE Internet of Things Journal*, vol. 7, no. 4, pp. 3441–3452, April 2020.

[2] F. Meneghello, M. Calore, D. Zucchetto, M. Polese, and A. Zanella, “Iot: Internet of threats? a survey of practical security vulnerabilities in real iot devices,” *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8182–8201, October 2019.

[3] K. Cohn-Gorden, C. Cremers, and L. Garratt, “On post-compromise security,” in *Proceedings of IEEE 29th Computer Security Foundations Symposium (CSF)*, June 2016, pp. 164–178.

[4] J. Cui, X. Tao, J. Zhang, Y. Xu, and H. Zhong, “Hcpa-gka: A hash function-based conditional privacy-preserving authentication and group-key agreement scheme for vanets,” *Vehicular communications*, vol. 14, pp. 15–25, 2018.

[5] G. Sharma, R. A. Sahu, V. Kuchta, O. Markowitch, and S. Bala, “Authenticated group key agreement protocol without pairing,” in *Proceedings of International Conference on Information and Communications Security (ICICS)*, 2017, pp. 606–618.

[6] A. Kumar and S. Tripathi, “Anonymous id-based group key agreement protocol without pairing.” *IJ Network Security*, vol. 18, no. 2, pp. 263–273, 2016.

[7] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” pp. 1–9, 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[8] V. Buterin, “A next generation smart contract & decentralized application platform,” *Ethereum White Paper*, pp. 1–36, 2014.

[9] T. El Gamal, “A public key cryptosystem and a signature scheme based on discrete logarithms,” in *Proceedings of CRYPTO 84 on Advances in Cryptology*, 1985, p. 10–18.

[10] M. Marlinspike and T. Perrin, “The x3dh key agreement protocol,” pp. 1–11, 2016. [Online]. Available: https://https://www.signal.org/docs/specifications/x3dh/x3dh.pdf

[11] H. Krawczyk, “Cryptographic extraction and key derivation: The hkdf scheme,” in *Proceedings of Advances in Cryptology-CRYPTO 2010*, August 2010, pp. 631–648.

[12] V. Denis, “The different types of blockchains,” 2018, accessed: 2020-02-8. [Online]. Available: https://medium.com/the-capital/the-different-types-of-blockchains-456968398559

[13] [Online]. Available: http://i2s.kennesaw.edu/resources/