Dynamic Group Key Agreement for Resource-Constrained Devices using Blockchains

Anonymous

No Institute Given

Abstract. Dynamic group key agreement (DGKA) protocols are one of the key security primitives to secure multiparty communications in decentralized and insecure environments while considering the instant changes in the communication group. However, with the ever-increasing number of connected devices, traditional DGKA protocols have performance challenges since each member in the group has to make several computationally intensive operations while verifying the keying materials in order to compute the resulting group key. To overcome this issue, we propose a new approach for DGKA protocols by utilizing Hyperledger Fabric framework as a blockchain platform. To this end, we migrate the communication and verification overhead of DGKA participants to the blockchain network in our developed scheme. This paradigm allows a flexible DGKA protocol that considers resource-constrained entities and trade-offs regarding distributed computation. According to our performance analysis, participants with low computing resources can efficiently utilize our protocol. Furthermore, we have demonstrated that our protocol has the same security features as other comparable protocols in the literature.

Keywords: Group Key Agreement · Blockchain · Hyperledger Fabric.

1 Introduction

The digitalization of daily life and human activities has become a reality via the emergence of more prevalent and high-performance communications and networking. With the advent of 5G networks, innovative and collective solutions which consist of different type of devices are now much more feasible. The envisaged use-cases and applications involve a huge number of devices and pervasive data sharing and dissemination such as massive Machine-Type Communications (mMTC) and Multi-access Edge Computing (MEC) scenarios. Although some of these systems require ultra-reliable and real-time connectivity in more preset conditions (e.g., telesurgery or industrial networks), many others require a dynamic environment where interaction between networked entities changes frequently and minor latency due to security functions can be tolerated (e.g., ad hoc data sharing or sensor-based monitoring scenarios). Moreover, IoT devices with low computing power and limited energy resources are expected to operate seamlessly and efficiently in future networks. In such systems, as the number of

connected devices increases rapidly, decentralized and efficient secure communication frameworks are essential to meet the service requirements.

As a secure communication facilitator, group key agreement protocols where participants can agree on a common secret key in an insecure channel have gained significant importance. Starting with Diffie-Hellman [8], where two parties can agree on a secret key, several protocols have been developed which enable multiple parties to agree on a common key [15,16]. However, such protocols were mostly designed for the static groups, where the members of the group do not change until the end of communication session. Therefore, if the members in the group change, the entire protocol should be executed from the beginning for all participants in the group. On the other hand, some protocols provide additional functionalities to handle the re-execution overhead. Such protocols are called Dynamic Group Key Agreement (DGKA) protocols [10, 12–14, 27]. Although DGKA protocols are more efficient compared to static ones, there exist some other several factors which affect their performance – the first one is the way of broadcasting key agreement parameters and the second being the validation of participant identities via verification of received parameters. To perform better in parameter distribution and verification stages, cluster-based approaches [11,14,18] and tree-based methods [10,17] have been proposed in the literature.

As a decentralized computing platform, blockchain technology has recently emerged starting with *Bitcoin* [21] as a monetary system based on cryptocurrency. The technology is later decoupled from cryptocurrencies and transformed into a wider domain with the concept of *Distributed Ledger Technology (DLT)* such as in Hyperledger Fabric (HF) [28]. Essentially, HF is a generic decentralized application development platform where transactions history is shared among peers as computation nodes in the network. DLT allows decentralization, greater transparency and easier auditability in a distributed setting.

In our work, we propose a dynamic group key agreement protocol called B-GKAP which is an improved version of KAP-PBC [13] protocol and integrates the HF platform to improve key computation performance while keeping important security properties of known DGKA protocols. The main rationale for using the blockchain technology is to offload computational burden in a trustworthy and distributed manner for resource-constrained environments. The HF provides capabilities, e.g., as Fabric Channels, of a permission-based blockchain platform to realize this extension in an efficient and secure way. The relevant benefits of our implemented approach are shown with the complexity analysis carried out for different aspects such as communication and computation in our experiments.

Our main contributions in this work are as follows:

1. To reduce the number of parameter transmissions in the protocol, DGKA participants communicate with the blockchain network instead of communicating with each other in our proposal. In this way, the amount of network transmissions is decreased significantly.

2. We employ a blockchain application in HF to perform verification of group key agreement parameters. Therefore, instead of each participant performing verification of every other participant, we migrate those operations to the blockchain network. Our performance evaluation show that B-GKAP provides better results in terms of scalabaility when compared to conventional DGKA protocols.
3. We propose a detailed security analysis for B-GKAP by considering the the well-known security attacks and properties. Our analysis show that B-GKAP achieves the same level of security with the existing DGKA protocols. Furthermore, we extend our analysis for the use of blockchain network in group key computation based on the *honest-but-curious* security model.

The outline of this paper is as follows: in Section 3, we discuss the group key agreement solutions and blockchain platforms in the technical domain. Section 4 explains our proposed model B-GKAP in terms of its system model, protocol flow and functions. Then, in Section 5, we prove that our protocol has the same security features with known group key agreement protocols. In Section 6, we discuss performance of our model in terms of communication cost and computational cost complexities analysis and simulation results. Finally, Section 8 summarizes our findings in this study, followed by a discussion of potential future work.

2 Preliminaries

In this section, we introduce general definitions and the security model of B-GKAP.

2.1 General Definitions

This section introduces the general definitions of B-GKAP based on [13].

Definition 1. ***Participants:***

- *Each participant is an entity and is represented as* U_i.
- *Each participant* U_i *who fully follows the protocol is called as "honest participant".*
- *The participant list is represented as* $\mathcal{U} = \langle U_1, U_2, \ldots, U_{N+M} \rangle$ *which consist of two subgroups, network participants as* $|\mathcal{U}_{net}| = M$*, and group participants as* $|\mathcal{U}_{grp}| = N$.

$$\mathcal{U} = \mathcal{U}_{grp} \cup \mathcal{U}_{net}$$

- *The participant group* $\mathcal{U}_{grp}$ *is circular so that* $U_{N+i} = U_i$ *for some positive* $1 \leqslant i \leqslant N$*. The order of the participants is known by each participant.*

Definition 2. ***Public Parameters:*** *B-GKAP uses the following public parameters based on the definitions in [13]:*

- $p = 2q + 1$*, where both* p *and* q *are large prime numbers.*

– g *is a generator for* $G_q = \{i^2 | i \in Z_p^*\}$*, where* G_q *is a cyclic subgroup of quadratic residues in* Z_p^**.*
– T *is the time-stamp against replay attack.*

Definition 3. ***Long-term Public Private Key Pair:*** *The protocol uses the following long-term key definitions based on [13]. Each entity in B-GKAP holds this key pair.*

– $x_i \in Z_q^*$ *is the private key and only the entity that holds the key knows it. This key is never shared with other entities in the network.*
– y_i *is the public key where* $y_i = g^{x_i} \bmod p$

Since our solution is based on KAP-PBC in [13], we assume that long-term public keys of each participant are issued via a Certification Authority (CA). Before the transmission, each variable is signed with long-term private key. Thus, during signature verification stage, identities of the participants are authenticated.

Definition 4. ***Schnorr Signature Scheme:*** *Based on the definition in [24], a message* M *can be signed as* $e, s = SS(x_i, y_i, M)$ *and the signature products* e, s *can be verified using* $SV(y_i, (e, s), M) \stackrel{?}{=} True$*. In these equations, SS stands for 'Schnorr Sign' and SV stands for 'Schnorr Verify'.*

Definition 5. ***Ledger Functions:*** *Each network participant* U_i *in* $\mathcal{U}_{net}$ *maintains its own blockchain ledger and has two functionalities called* $readLedger(\cdot)$ *and* $writeLedger(\cdot)$*. A variable* x *can be written to the ledger via* $writeLedger(x)$*, and read from the ledger via* $x = readLedger()$*.*

2.2 Security Models

We consider *malicious* and *honest-but-curious* security models for the group participants and the network participants of B-GKAP, respectively. We assume that the potential entities for the malicious security model are as follows:

– *Group participants:* Participants that actively involve in the group key computation.
– *B-GKAP users:* Users that have valid certificates received from CA. They do not need to participate every group key computation.
– *Non-members:* Standard Internet users without a valid certificate.

Based on the entity definitions above, a DGKA protocol should provide security against the following threats: (i) violation of security properties (authentication, fault-tolerance and forward secrecy), (ii) security attacks (impersonation, eavesdropping and replay attacks) and (iii) secure dynamic group operations (backward and forward confidentiality properties). We refer the reader to Section 5 for further details.

On the other hand, B-GKAP differs from conventional DGKA protocols by including entities of HF into group key computation process. The main idea is to outsource the verification overhead of temporary public keys and the exchanged secrets to the HF network for achieving a more scalable DGKA protocol. Network participants are powerful entities and they have access to all keying materials to compute the group key; in fact, these participants are computationally bounded by the B-GKAP algorithm and they cannot compute the group key. Therefore, we consider the *honest-but-curious* security model for the network participants. Based on the definition in [23], we define *honest-but-curious* adversary for B-GKAP as follows:

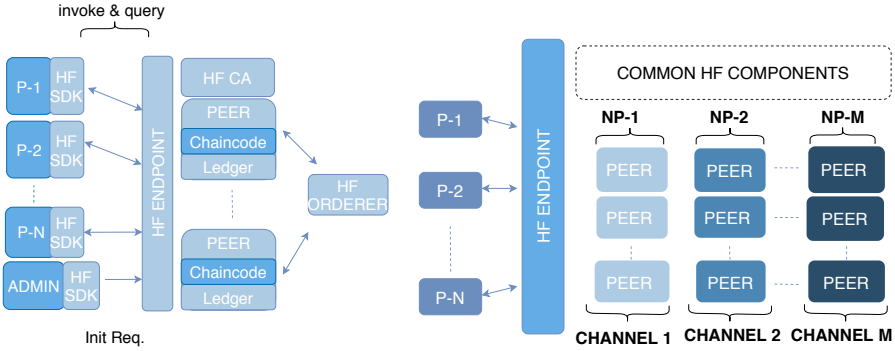
Definition (Honest-But-Curious (HBC) Adversary). *The Honest-But-Curious (HBC) adversary is a legitimate network participant in B-GKAP that tries to learn the group key while honestly following the protocol [23].*

3 Related Work and Technical Background

In this section, we overview the literature with respect to DGKA protocols and blockchain technologies. Particularly, we elaborate on the HF[1] platform which we have employed as the blockchain platform to implement and evaluate our protocol.

3.1 GKA Protocols

Diffie-Hellman key exchange protocol [8] is the first group key agreement protocol that is used for securing the communication among two parties using a common key computed by these participants. Later, the concept of two-party secure communication was extended by Ingamarsson et al. in [15] to multi-party setting. In addition, the protocol in [4] is also accepted as a pioneering work regarding the key agreement protocol research with the proposed efficient group key computation. Nevertheless, these protocols were designed specifically for static groups, which means any change in the set of communicating parties requires the re-execution of the group key agreement protocol for all communicating participants. In this work, our main focus is the dynamic group key agreement (DGKA) protocols that use some auxiliary functions to update the group key without re-executing the protocol from scratch for all participants as elaborated in [7, 9, 10, 12–14, 25]. These protocols have many application areas such as conference communication [12], secure file sharing systems [13], and secure communication in Mobile Ad Hoc Networks (MANETs) [14]. Although the listed protocols have proposed efficient group key computation approaches, novel solutions are necessary to overcome the overhead of verification operations during group key computation while dealing with large groups.



(a) B-GKAP block diagram.

(b) B-GKAP model. (P: Group Participant NP: Network Participant)

Fig. 1: B-GKAP block diagram and model.

[1] https://www.hyperledger.org/projects/fabric

3.2 HF Platform

HF is a permissioned blockchain platform that only allows identified participants [1]. Thus, with the identification of the network modules, Byzantine Fault Tolerant (BFT) [6] or Crash Fault Tolerant (CFT) [22] consensus protocols can be utilized. Another important feature of HF is that most of the HF components are designed to be modular such as Membership Service Provider (MSP) and consensus protocol. This modular design is made possible by its novel execute-order-validate architecture. In other applications [5,21], order-execute architecture is utilized where transactions are first ordered via a consensus protocol, and then they are executed by all peers sequentially. On the other hand, in HF, execution of the transactions is performed first to allow running non-deterministic applications and the ordering phase is separated from the validation step to isolate consensus logic from the peers. Therefore, the transactions can run in parallel without the necessity to keep the order. After the consensus is provided by ordering state, the final state of the transaction can be applied by all nodes individually. Therefore, in this work, our main motivation is to employ HF platform to perform necessary verification operations for increasing the group key computation performance.

4 B-GKAP: Blockchain-based Group Key Agreement Protocol

In this section, we introduce our Blockchain based Group Key Agreement Protocol (B-GKAP) which is deployed on HF as a blockchain platform. In the first section, we provide a system overview that explains the positioning of the HF components. Then, we introduce B-GKAP in more details.

4.1 System Overview

B-GKAP is based on the Key Agreement Protocol with Partial Backward Confidentiality, namely KAP-PBC [13], but extends and improves its performance with HF platform. Additionally, in B-GKAP, we migrate the communication among participants to communication between participants and the network, which in return reduces the communication cost during the group key computation in terms of the length of the transmitted messages. Moreover, to verify the variables of the participants, we utilize HF chaincodes. When a variable is received as an invoke request by the network, the chaincode first performs the verification operation depending on the variable type. Then if the verification succeeds, the chaincode approves the operation.

The overview of B-GKAP is shown in Fig. 1a, which consists of the following main components:

1. B-GKAP participants are the entities which compute the group key.
2. B-GKAP admin sends initialization command to start up the HF platform and setup initial variables as specified in Section 2.1. Both B-GKAP participants and admin use HF Software Development Kit (SDK) which enables them to communicate with the network.
3. The peers are responsible for simulating incoming transactions by utilizing B-GKAP chaincode. Additionally, each peer maintains a blockchain ledger and latest ledger state. We have utilized HF ledger to store B-GKAP parameters.
4. HF Endpoint is a logical endpoint which can correspond peers or orderers, according to the operation described in Section 3.2.

5. B-GKAP Chaincode handles all ledger read-write requests of the participants. The chaincode performs all the necessary verification operations and, if the request is valid, it produces read-write set.
6. HF Orderer performs the ordering of the produced transaction output sets as a block of transactions and it disseminates to all HF peers. Then peers update their ledger states.
7. HF CA maintains the identities of the HF components and B-GKAP participants.

4.2 B-GKAP Protocol

As shown in Fig. 2, first, each participant $U_i \in \mathcal{U}$ executes *Public Key Distribution* step to distribute temporary public keys. Then, each network participant executes *Public Key Verification* and *Fault Correction* steps to remove dishonest participants from the group. Later on, remaining honest participants execute *Public Key Query* to fetch the temporary public key of the next participant in the group. Once this step is completed, each participant executes the *Secret Key Distribution* to send the secret keys to the network participants. Afterward, network participants perform *Secret Key Verification* and *Fault Correction* to exclude malicious participants from the group. Finally, each participant performs *Secret Key Query* and *Group Key Computation* steps to compute the common group key. Additionally, when a new participant joins the group or leaves the group, *Participant Join* or *Participant Leave* steps can be executed.

As illustrated in Fig. 1b, there are network participants which are not involved in the group key computation phase. Instead, they produce B-GKAP parameters except for the secret key. Therefore, network participants can verify the temporary public and secret keys of the participants who compute the group key. Each network participant has multiple peers and an isolated ledger via HF channels. In this way, secret key variables can be stored and validated separately by each network participant.

4.3 B-GKAP Protocol Steps

In this section, we give details of the protocol steps.

Public Key Distribution (`sendPK(·)`). *Each participant $U_i \in U$ executes the following to distribute temporary public keys. Group participants $U_j \in U_{grp}$ distribute their temporary public keys to each network participant $U_t \in U_{net}$.*

1: randomly select $t \in Z_q^*$
2: $\omega = g^t mod p$
3: Sign ω: $e, s = SS(x, y, \omega)$
4: Send the message $M = \{\omega, e, s, T\}$

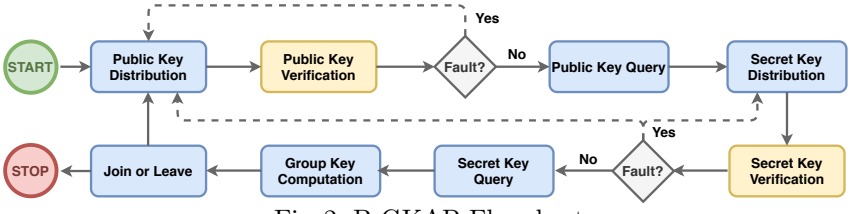


Fig. 2: B-GKAP Flowchart

Public Key Verification (`verifyPK(·)`). *Each network participant $U_j \in U_{net}$ executes following to verify temporary public key of each group participant $U_i \in \mathcal{U}_{grp}$. According to verification result, the key is written to the ledger of U_j.*

1: ***for all*** $U_i \in \mathcal{U}_{grp}$ ***do***
2: *Check the timestamp T*
3: ***if*** $SV(y_i, (e_{1,i}, s_{1,i}), \omega_i)$ ***then***
4: $writeLedger(\omega_i)$
5: ***end if***
6: ***end for***

Fault Correction (`faultCorr(·)`). *Each network participant $U_j \in U_{net}$ performs following to remove any group participant $U_i \in \mathcal{U}_{grp}$ whose verification of its temporary public key or secret key fails.*

1: ***for all*** $U_i \in \mathcal{U}_{grp}$ ***do***
2: ***if*** U_i *is faulty* ***then***
3: U_i *is removed from the participant group,* $\mathcal{U}' = \mathcal{U} - U_i$
4: *Execute* $participantLeave(\cdot)$
5: ***end if***
6: ***end for***

Public Key Query (`queryPK(·)`). *In this step, each group participant $U_i \in \mathcal{U}_{grp}$, requests for temporary public key of next group participant U_{i+1} in the group from the target network participant $U_j \in \mathcal{U}_{net}$.*

U_j performs the following:
1: $\omega_{i+1} = readLedger()$
2: Sign temporary public key of U_{i+1}, $e_j, s_j = SS(y_j, x_j, \omega_{i+1})$
3: Send message to U_i, $M = (\omega_{i+1}, e_j, s_j, T)$
U_i performs the following:
1: Receive the message M
2: Check timestamp T
3: Verify signature of U_j: $SV(y_j, (e_j, s_j), \omega_{i+1}))$

Secret Key Distribution (`sendSK(·)`). *Each group participant $U_i \in \mathcal{U}_{grp}$ performs following to generate and distribute secret key (CK_i) to the target network participant $U_j \in \mathcal{U}_{net}$. The selection of U_j is determined in a way to ensure equal distribution of secret keys.*

1: Generate CK_i: $CK_i = \omega_{(i+1)}^{t_i} \mod p = g^{t_i t_{i+1}} \mod p$
2: Randomly select an integer $a \in Z_q^*$
3: $k = (\omega_j^a mod p) \mod q$
4: Randomly select a line $L(x)$, $L(x) = xc_i + CK_i \mod q$, $c_i = g^a \mod p$
5: $d_i = L(k) \mod q$
6: $d_i' = k \oplus d_i$
7: $e_{2,i}, s_{2,i} = SS(x_i, y_i, CK_i)$
8: Send the message $M = \{s_{2,i}, e_{2,i}, c_i, d_i', T\}$

Secret Key Verification (`verifySK(·)`). *Each network participant $U_j \in \mathcal{U}_{net}$ performs following to verify secret keys of group participants $U_i \in \mathcal{U}_{grp}$.*

1: ***for all*** $U_i \in \mathcal{U}_{grp}$ ***do***
2: Receive message $M = \{s_{2,i}, e_{2,i}, c_i, d'_i, T\}$
3: Recover CK_i and check T
4: $k = (c_i^{t_j} \mod p) \mod q$
5: $d = d'_i \oplus k$
6: $CK_i = d - c_i * k \mod q$
7: Check the signature of U_i
8: ***if*** $SV(y_i, (e_{2,i}, s_{2,i}), CK_i)$ ***then***
9: $writeLedger(CK_i)$
10: ***end if***
11: ***end for***

Secret Key Query (`querySK(·)`). *After the fault correction step, each participant $U_i \in \mathcal{U}_{grp}$ performs following to query secret keys from each $U_j \in \mathcal{U}_{net}$ ($CK_{1...N} = CK_1||CK_2||\ldots||CK_N$).*

U_j *performs the following:*
1: Randomly select an integer $a \in Z_q^*$
2: $c_j = g^a \mod p$
3: $k_i = (\omega_i^a modp) modq$
4: ***for all*** $U_k \in \mathcal{U}_{grp} - U_i$ ***do***
5: Randomly select a line $L(x) = xc_j + CK_k \mod q$
6: $d_k = L(k_i) \mod q$
7: $d'_k = k_i \oplus d_k$
8: ***end for***
9: Sign $CK_{1...N}$: $e_{2,j}, s_{2,j} = SS(x_j, y_j, CK_{1...N})$
10: Send $M = \{s_{2,j}, e_{2,j}, c_j, \{d'_1, d'_2, \ldots, d'_N\}, T\}$

U_i *performs the following:*
1: $k_i = (c_j^{t_i} \mod p) \mod q$
2: ***for all*** $U_k \in \mathcal{U}_{grp} - U_i$ ***do***
3: $d_k = d'_k \oplus k_i$
4: $CK_k = d_k - c_j * k_i \mod q$
5: ***end for***
6: Check timestamp T
7: Check the signature of U_j: $SV(y_j, (e_{2,j}, s_{2,j}), CK_{1...N})$

Group Key Computation (`compute(·)`). *Each group participant $U_i \in \mathcal{U}_{grp}$, computes the group key.*

1: ***for all*** $U_i \in \mathcal{U}_{grp}$ ***do***
2: $CK = ((CK_1 CK_2 \cdots CK_{|\mathcal{U}_{grp}|}) modp) modq = (g^{t_1 t_2 + t_2 t_3 + \ldots + t_{n-1} t_n + t_n t_1} modp) modq$
3: ***end for***

Participant Join (`join(·)`). *Let U_i be the participant that wants to join the group $U_{grp} = \{U_1, U_2, ..., U_N\}$. The join operations operates as follows:*

1: ***if*** $U_i \in \{U_N, U_{N+1}, \ldots, U_{N+K}\}$ ***then***
2: U_i *performs* $queryPK(\cdot)$ *and* $querySK(\cdot)$
3: U_i *performs* $sendPK(\cdot)$ *function*
4: *Network participants perform* $faultCorr(\cdot)$
5: U_{i-1} *performs* $sendSK(\cdot)$ *function*
6: *Network participants perform* $faultCorr(\cdot)$
7: ***end if***
8: ***for all*** $U_i \in \{U_1, U_2, \ldots, U_{N+K}\}$ ***do***
9: U_i *performs* $querySKs(\cdot)$ *and* $groupKeyComputation(\cdot)$ *functions*
10: ***end for***

Participant Leave (`leave(·)`). *Let* $U_i, U_{i+1}, ..., U_{i+N}$ *be the set of leaving participants and* U'_{grp} *be the group participants after the leave operation. Then, the leave operation operates as follows:*

1: ***if*** $|\mathcal{U}_{grp}| - |\mathcal{U}'_{grp}| < 2$ ***then***
2: *The group key computation is terminated*
3: ***end if***
4: ***for each*** *leaving participant* $U_j \in \mathcal{U}'_{grp}$ ***do***
5: *non-leaving participant(s)* $U_{j-1} \in \mathcal{U}_{grp} - \mathcal{U}'_{grp}$, *performs* $sendPK(\cdot)$
6: *Network participants perform* $faultCorr(\cdot)$
7: U_{j-1} *and* $U_{j-2} \in \mathcal{U}_{grp} - \mathcal{U}'_{grp}$ *perform* $sendSK(\cdot)$
8: *Network participants perform* $faultCorr(\cdot)$
9: ***end for***
10: ***for all*** $U_i \in \mathcal{U}_{grp} - \mathcal{U}'_{grp}$ ***do***
11: U_i *performs* $querySK(\cdot)$ *and* $groupKeyComputation(\cdot)$ *functions*
12: ***end for***

5 Security Analysis

In this section, we provide a security analysis of B-GKAP. We consider the basic security properties as well as potential attacks for group key agreement protocols for our analysis.

5.1 Security Properties of GKA Protocols

In this section, we analyze B-GKAP with respect to the the basic security properties of group key agreement protocols such as authentication, fault tolerance and forward secrecy.

Authentication: This property is used for validating the identities of participants during the execution of the protocol. In B-GKAP, as an initial authentication mechanism, we assume that all participants are pre-identified with HF CA [28]. As a requirement of interacting with HF Network, all participants have to use TLS v1.3[2] certificate to provide identification. The TLS certificate is created by HF Admin prior to the network initialization. For the second level of authentication mechanism, long-term key pairs of the participants are used. All long-term public keys of the participants must be signed by a trusted CA. During variable exchange between the participants and the network,

[2] https://tools.ietf.org/html/rfc8446

all message payloads is signed with the long-term private key of the sender entity. Eventually, receiving entity verifies the signature of the payload by sender's long-term public key using Schnorr's signature [24].

Fault Tolerance: In the course of group key agreement processes, malicious participants should be immediately detected and removed from the group. In B-GKAP, detection and elimination of faulty participants occurs during the execution of $verifyPK(\cdot)$, $verifySK(\cdot)$ and $faultCorr(\cdot)$ functions. If a malicious participant is detected in $verifyPK(\cdot)$ or $verifySK(\cdot)$ functions, the key of the malicious participant is not written to the ledger. Later, the $faultCorr(\cdot)$ function is used for removing this participant from the group.

Forward Secrecy: This property is used for protecting the previous and subsequent group keys against the compromise of long-term private keys of participants in the group. Therefore, in B-GKAP, the long-term key pairs of participants are only used to authenticate these participants. Additionally, each entity in B-GKAP generates a new temporary public-private key pair using $sendPK(\cdot)$ function for each session. Thus, B-GKAP provides the forward secrecy property.

5.2 Protection Against Security Attacks

In this section, we provide the security analysis of B-GKAP against impersonation, eavesdropping and replay attacks.

Impersonation Attack: As mentioned in Section 2.2, the motivation of the impersonation attack is to take place of any group participant during the protocol execution. To do so, an attacker needs to be able to generate the signature of that entity. Since B-GKAP uses the Schnorr signature scheme [24] as an authentication mechanism and, as stated in [2] and [20], the Schnorr signature is secure against impersonation attacks, B-GKAP also provides security against the impersonation attack.

Eavesdropping Attack: The goal of the eavesdropping attack in group key agreement protocols is to capture the computed group key by eavesdropping the communication channels among entities. In order to generate a group key, the attacker must obtain secret keys (CK_i) of all participants ($CK_1, \ldots, CK_N$). In B-GKAP, entities exchange secret keys in $sendSK(\cdot)$ and $querySK(\cdot)$ functions. In those functions, all secret key values are extracted using c, d' variables of the sender and temporary private key (t) of the receiver. Since only the participants in the key agreement group $\mathcal{U}$ knows their temporary private key (t), to compute $k = (c^t mod p)$ equation, attacker should try to extract t_j from $\omega_j = g^{t_j} mod p$ for each participant U_j in $\mathcal{U}$. Therefore, because solving this equation is as hard as the discrete logarithm problem, B-GKAP is secure against eavesdropping attack.

Replay Attack: During the communication between the B-GKAP entities, an attacker might capture and re-transmit messages in the network to degrade the availability of recipient parties such as in Distributed Denial-of-Service (DDoS) attack. Therefore, to provide a protection against such attacks, we also append timestamp variable (T) into the communication messages among participants during the execution of the protocol.

5.3 Security of Join and Leave Operations

Dynamic group operations enables group key agreement protocols to be more efficient during re-generation of group keys. In order to overcome security weaknesses stated in

[19], a protocol must ensure forward and backward confidentiality properties. Forward confidentiality property assures that further group keys cannot be computed by a participant who has left the group. Conversely, backward confidentiality feature warrants that previous group keys cannot be computed by recently joined participants. In the following sections, we prove that Leave and Join operations in B-GKAP provide the same security features as [13] assures.

Prior to applying dynamic group operations, let the participant group be $\mathcal{U}_{grp} = \{U_1, U_2, U_3, \ldots, U_N\}$ and the group key be:

$$CK = ((g^{t_1 t_2 + \ldots + t_{i-1} t_i + t_i t_{i+1} + \ldots + t_N t_1}) mod p) mod q)$$

Lemma 1. *Under the difficulty of discrete logarithm problem, join operation does not violate backward confidentiality.*

Proof. The participant U_N and joining participants should run $join(\cdot)$ algorithm. Let the joining participants be $U' = U_{N+1}, U_{N+2}, \ldots, U_{N+k}$ and the new group key be $CK' = ((g^{t_1 t_2 + \ldots + t_{N-1} t'_N + t'_N t_{N+1} + \ldots + t_{N+k} t_1}) mod p) mod q$. We assume that any joining participant with malicious intent has the previous message exchanged during the computation of CK. In order to compute the previous group key, the malicious participant should obtain t_N from either using $g^{t_{N-1} t_N}$ or $g^{t_N t_1}$, which is as hard as solving the discrete logarithm problem. Thus, B-GKAP provides backward confidentiality under the difficulty of discrete logarithm problem.

Lemma 2. *Under the difficulty of discrete logarithm problem, leave operation does not violate forward confidentiality.*

Proof. Let $\mathcal{U}' = U_i, U_{i+1}, U_{i+2}, \ldots U_{i+k}$ be leaving participants where $U' \subseteq U$. As given in function $leave(\cdot)$ (Section 2.1), when participants in $\mathcal{U}'$ leaves the group, the group key is re-computed as $CK' = ((g^{t_1 t_2 + \ldots + t_{i-2} t'_{i-1} + t'_{i-1} t_{i+k+1} \cdots + t_N t_1}) mod p) mod q$. If there exist some malicious participants in the leaving ones and they want to compute the new group key CK' using the old keying materials, they have to obtain t'_{i-1} by either using $g^{t_{i-2} t'_{i-1}}$ or $g^{t'_{i-1} t_{i+k+1}}$. Since solving these equations is as hard as solving the discrete logarithm problem, B-GKAP provides forward confidentiality property.

5.4 Security of B-GKAP Hyperledger Fabric Network

In B-GKAP, the network participants store all keying material exhanged by the group participants. Such participants are involved in the execution of B-GKAP except for the secret key generation and key computation steps. During the network initialization step, ($sendPK(\cdot)$), each network participant U_m network entities generate temporary public-private key pair (t_m, ω_m). Participants use these public keys to encrypt their secret keys before the submission. Thus, the network can unveil the secret keys of the participants using its temporary private key (t_m). Additionally, the network participants also holds its own long-term key pair (x_m, y_m). These keys are used for signing the protocol variables that are sent to the participants. Hence, participants ensure that they are receiving variables from a trusted entity. Moreover, the secret key verification is distributed among the network entities. Since the ledger of each entity is isolated via the Fabric channels, the secret keys of the participants are also isolated[3]. Therefore, as

[3] https://developer.ibm.com/tutorials/cl-blockchain-private-confidential-transactions-hyperledger-fabric-zero-knowledge-proof/

described in Section 2.2, we consider the *honest-but-curious* security model for network participants. Although they have the keying material to compute the group key, all the network entities should work cooperatively to compute the key.

Furthermore, HF platform provides storage immutability via blockchain ledgers. The ledger is distributed among peers, and to change the ledger data, a subset of peers needs to generate the same output. This feature is forced by endorsement policies. Finally, our design guarantees that even if a peer is compromised or failed, the system remains operational.

In B-GKAP protocol, prior to secret key distribution, identities of network participants are verified via $U_j \in \mathcal{U}_{network}$, $SV(y_j, e_{2,j}, s_{2,j}, \omega_j) \neq false$, and secret key CK_i of each participant $U_i \in \mathcal{U}_{grp}$ is encrypted via temporary public key ω_j of the network participant $U_j \in \mathcal{U}_{net}$. Thus, network participants can only receive intended parameters which are $\omega_i, s_i, e_i, d'_i, c_i$ for $U_i \in \mathcal{U}_{grp}$. Moreover, in HF, the only way to interact with the ledger is via Fabric chaincodes. Because implemented chaincode has defined a set of functionality and is shared among the peers, the network has no other choice but to follow the B-GKAP protocol. Given these properties, B-GKAP fits the HBC adversary model.

6 Performance Analysis

In this section, communication cost and computational cost complexity of B-GKAP are analyzed and simulation results are presented. During our analysis, we only consider the participants which compute the group key. All simulations were carried out with a machine of Intel Core Broadwell Processor (2.5GHz x 8 cores), L1 Cache 32KB, L2 Cache 4MB, L3 Cache 16MB, and 32GB RAM. We have used Docker Engine v18.06.1-ce-mac73[4], and Hyperledger Fabric v1.4.3[5]. For both B-GKAP chaincode and B-GKAP participant implementations, we have used Go programming language v1.13.3[6]. Moreover, we have utilized the same environment for the implementation of other protocols for performance comparison.

Table 1: Transmission length of each B-GKAP function in bits.

Function	**Transmission length**
$sendPK(\cdot)$	$(2q + p)$
$queryPK(\cdot)$	$(2q + p)$
$sendSK(\cdot)$	$(3q + p)$
$querySK(\cdot)$	$(N + 1)q + p$

6.1 Communication Cost Complexity $\mathcal{C}_t$

B-GKAP Functions: In B-GKAP, variable transmission occurs between participants ($U_i \in \mathcal{U}_{grp}$) and network participants ($U_j \in \mathcal{U}_{net}$). Since all transmitted variables are modular base of p and q, length of a variable is equal to its modular base. Table 1 indicates network transmission length of each function in B-GKAP.

[4] https://docs.docker.com/engine/
[5] https://hyperledger-fabric.readthedocs.io/
[6] https://golang.org/project/

Table 2: Computational and communication complexities for N participants.

Protocol	$C_c \times T_{exp}$	C_t
Protocol in [10]	$\leq O(\log_3 N)$	-
Protocol in [26]	$O(N)$	$(N+2)\|q\| + 4\|p\|$
Protocol in [12]	$O(N)$	$(N+2)\|q\| + 4\|p\|$
GKAP-MANET [14]	$O(N)$	$2\|q\| + 5\|p\|$
KAP-PBC [13]	$O(N)$	$(N+4)\|q\| + 2\|p\|$
B-GKAP	$O(1)$	$5\|q\| + 2\|p\|$

Key Computation: During key computation, several variable transmissions between participants and the network occur. For each participant $U_i \in \mathcal{U}_{grp}$, network transmissions are performed in $sendPK(\cdot)$ and $sendSK(\cdot)$ functions. In total, for the number of network participants M, $|(2M+3)q+(M+1)p|$ bits are transmitted for each DGKA participant. Therefore communication complexity of key computation is $C_t = O(M)$.

Join Operation: For the join operation, when K participants join the group, $K+1$ participants perform network transmission in $sendPK(\cdot)$ function, and $K+2$ participants perform network transmission in $sendSK(\cdot)$. In total, for K joining participants, $|(2M(K+1)+3K+6)q+(M+K+2)p|$ bits are transmitted.

Leave Operation: In the leave operation, for the leaving participant U_i, participant U_{i-1} executes $sendPK(\cdot)$. Moreover, participants U_{i-1} and U_{i-2} execute $sendSK(\cdot)$. Therefore, $|(2M+6)q+(M+2)p|$ bits are transmitted.

6.2 Computational Cost Complexity C_c

In computational cost analysis, we consider modular exponential operations as the principal factor while calculating our results. The time cost of these operations can be stated as $T_{exp} = O(x^y \mod z)$. In B-GKAP, participants $U_i \in \mathcal{U}_{grp}$ performs verification for only network participants $U_j \in \mathcal{U}_{net}$. If we consider that M is negligible against the participant count N, the computational cost complexity of group key computation, join and leave operations is $C_c = O(1)T_{exp}$.

7 Discussion on the Performance of B-GKAP

In this section, we compare the communication cost and computational cost complexities of B-GKAP with the well-known GKA protocols by considering the the total computational and communication costs of a single participant as the benchmark.

Comparison for the computational and communications costs of B-GKAP with other well-known dynamic group key agreement protocols [10, 12–14, 26] is given in Table 2. For instance, GKAP-MANET protocol relies on the most efficient group key proposed by Burmester and Desmedt (BD) in [4]. BD protocol have many other variants that improves the security of the original work against active attacks while achieving a constant round of communication and less computational overhead as introduced in Katz-Yung protocol [16]. However, such static protocols are computationally more expensive than the dynamic ones. Therefore, we only compare B-GKAP with other dynamic group key agreement protocols.

As shown in Table 2, B-GKAP is more efficient than most of the protocols regarding the communication and computational complexities for each participant. In terms of total communication complexity, the other protocols perform network transmission to every other participant in the key agreement group. On the other hand, B-GKAP participants only transmit messages to a limited number of network participants. Moreover, in B-GKAP, participants only perform verification for the network participants instead of performing verification for incoming parameters from other participants. As a cost of utilizing HF SDK, the lightweight client code in a B-GKAP participant needs to establish several connections to a limited number of HF peers and orderers for data transmissions, which is a slight additional overhead compared to the mentioned protocols. Moreover, this number becomes more negligible when the number of participants increases. Given these reasons, the overhead for participants in B-GKAP is significantly lower than counterparts. Due to these complexity advantages compared to other protocols, B-GKAP can be a good candidate for resource-constrained devices.

As a case study, we further investigate B-GKAP performance and compare it to KAP-PBC counterpart as a baseline scheme. Fig. 3 depicts the group key computation performance of KAP-PBC and B-GKAP for N group participants. As the orderer parameters, we set batch size as N/M for B-GKAP where M is number of network participants and batch timeout as five seconds. Since the solution is designed for several network participants such as large organizations, this simulation is performed with one and two network participants, which are represented as B-GKAP and B-GKAP$^+$ respectively. Additionally, each network participant maintains two HF peers.

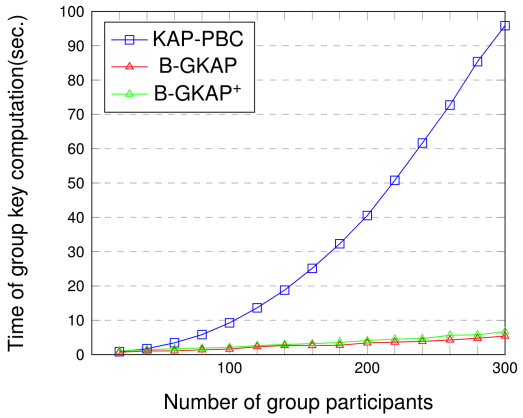


Fig. 3: B-GKAP and conventional model comparison.

The results show that, group key computation time of the conventional method increases exponentially as the participant count increases. In contrary, group key computation of B-GKAP increases linearly. First, the communication cost complexity of KAP-PBC is $O(N^2)$ whereas in B-GKAP, the communication cost complexity is $O(N)$. For the computational cost complexity, B-GKAP outperforms KAP-PBC with its constant computational cost $O(1)$. The reason of slight performance difference between B-GKAP and B-GKAP$^+$ is that in B-GKAP$^+$, the orderer needs to process submitted public variables in two transaction blocks. Moreover, in our simulations, we set batch size as the HF network can process without an issue. For more participants, the network should process the transactions for multiple batches, which would affect the performance slightly.

8 Conclusion

In this study, we present B-GKAP which employs Hyperledger Fabric (HF) blockchain platform to partially offload the group key computation and thus alleviate the performance burden for resource-constrained environments. Our approach is specifically geared towards non-real-time scenarios such as ad hoc data sharing or group communications considering potential latency due to blockchain operations, albeit being minimal. With our protocol, the computation overhead of the group key agreement participants is decreased significantly by migrating the verification of the distributed parameters to the network participants. Thus, participants with low computation power and energy resource can conveniently adopt our GKA protocol. Additionally, we have reduced the number of network transmissions for group key computation, leave and join operations. Hence, for network environments such as IoT where participant group changes frequently, our solution provides more efficient dynamic operations. Furthermore, we have distributed secret keys of the participants among the network participants via Fabric Channels. In this way, malicious network participants cannot generate group keys without colluding.

HF platform provides immutable storage property for stored variables via blockchain ledger. Additionally, in HF, not only valid but also invalid transactions are stored in the ledger. This feature makes our system auditable for deeper investigations. Another important feature of HF is its modularity. For instance, its consensus protocol can be replaced with more efficient methods as a future work. Moreover, with its modular membership service provider, various authentication schemes can be utilized depending on the usage area of the protocol.

Furthermore, to overcome the problem of colluding network participants, Fabric chaincode runtime environment, and ledger storage can be transferred to a Trusted Execution Environment (TEE) [3]. Consequently, even the network participants cannot access to secret keys of the participants.

References

1. Hyperledger Fabric, https://www.hyperledger.org/projects/fabric, accessed: 2019-09-30
2. Bellare, M., Palacio, A.: GQ and Schnorr identification schemes: Proofs of security against impersonation under active and concurrent attacks. In: Yung, M. (ed.) Advances in Cryptology — CRYPTO 2002. pp. 162–177. Springer Berlin Heidelberg, Berlin, Heidelberg (2002)
3. Brandenburger, M., Cachin, C., Kapitza, R., Sorniotti, A.: Blockchain and trusted computing: Problems, pitfalls, and a solution for hyperledger fabric. arXiv e-prints arXiv:1805.08541 (May 2018)
4. Burmester, M., Desmedt, Y.: A secure and efficient conference key distribution system. In: Workshop on the Theory and Application of of Cryptographic Techniques. pp. 275–286. Springer (1994)
5. Buterin, V.: A next-generation smart contract and decentralized application platforme, https://github.com/ethereum/wiki/wiki/White-Paper
6. Castro, M., Liskov, B.: Practical byzantine fault tolerance and proactive recovery. ACM Transactions on Computer Systems (TOCS) **20**(4), 398–461 (2002)
7. Chuang, Y.H., Tseng, Y.M.: An efficient dynamic group key agreement protocol for imbalanced wireless networks. Int. J. of Net. Man. **20**, 167–180 (2010)

8. Diffie, W., Hellman, M.: New directions in cryptography. IEEE Transactions on Information Theory **22**, 644 – 654 (12 1976). https://doi.org/10.1109/TIT.1976.1055638
9. Dutta, R., Barua, R.: Constant round dynamic group key agreement. In: Zhou, J., Lopez, J., Deng, R.H., Bao, F. (eds.) Information Security (2005)
10. Dutta, R., Barua, R.: Dynamic group key agreement in tree-based setting. In: Boyd, C., González Nieto, J.M. (eds.) Information Security and Privacy. pp. 101–112. Springer Berlin Heidelberg, Berlin, Heidelberg (2005)
11. Dutta, R., Dowling, T.: Secure and efficient group key agreements for cluster based networks. Transactions on Computational Science **4**, 87–116 (01 2009). https://doi.org/10.1007/978-3-642-01004-0_6
12. Ermiş, O., Bahtiyar, Ş., Anarim, E., Çağlayan, U.: An improved conference-key agreement protocol for dynamic groups with efficient fault correction. Security and Communication Networks **8**, 1347–1359 (05 2015)
13. Ermiş, O., Bahtiyar, Ş., Anarim, E., Çağlayan, U.: A key agreement protocol with partial backward confidentiality. Computer Networks **129**, 159–177 (2017)
14. Ermiş, O., Bahtiyar, Ş., Anarim, E., Çağlayan, U.: A secure and efficient group key agreement approach for mobile ad hoc networks. Ad Hoc Netw. **67**, 24–39 (2017)
15. Ingemarsson, I., Tang, D., Wong, C.: A conference key distribution system. IEEE Transactions on Information Theory **28**, 714–719 (1982)
16. Katz, J., Yung, M.: Scalable protocols for authenticated group key exchange. Journal of Cryptology **20**(1), 85–113 (2007)
17. Kim, Y., Perrig, A., Tsudik, G.: Tree-based group key agreement. ACM Transactions on Information and System Security **7**, 60–96 (02 2004)
18. Konstantinou, E.: Cluster-based group key agreement for wireless ad hoc networks. In: 2008 Third International Conference on Availability, Reliability and Security. pp. 550–557 (March 2008). https://doi.org/10.1109/ARES.2008.106
19. Lee, S., Kim, J., Hong, S.: Security weakness of Tseng's fault-tolerant conference key agreement protocol. Journal of Systems and Software **82**, 1163–1167 (2009)
20. Morita, H., Schuldt, J.C.N., Matsuda, T., Hanaoka, G., Iwata, T.: On the security of the Schnorr signature scheme and dsa against related-key attacks. In: Kwon, S., Yun, A. (eds.) Information Security and Cryptology - ICISC 2015. pp. 20–35. Springer International Publishing, Cham (2016)
21. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. Tech. rep., Manubot (2008)
22. Ongaro, D., Ousterhout, J.: In search of an understandable consensus algorithm. In: 2014 USENIX Annual Technical Conference (ATC). pp. 305–319 (2014)
23. Paverd, A., Martin, A., Brown, I.: Modelling and automatically analysing privacy properties for honest-but-curious adversaries. Uni. of Oxford, Tech. Rep (2014)
24. Schnorr, C.P.: Efficient identification and signatures for smart cards. In: Brassard, G. (ed.) Advances in Cryptology — CRYPTO' 89 Proceedings (1990)
25. Steiner, M., Tsudik, G., Waidner, M.: Key agreement in dynamic peer groups. IEEE Transactions on Parallel and Distributed Systems **11**(8), 769–780 (2000)
26. Tseng, Y.M.: An improved conference-key agreement protocol with forward secrecy. Informatica, Lith. Acad. Sci. **16**, 275–284 (01 2005)
27. Tseng, Y.M.: A communication-efficient and fault-tolerant conference-key agreement protocol with forward secrecy. J Syst. Software **80**, 1091–1101 (07 2007)
28. Vukolić, M.: Hyperledger fabric: towards scalable blockchain for business. Tech. rep., Tech. rep. Trust in Digital Life 2016. IBM Research, 2016. (2015)