Hindawi
Security and Communication Networks
Volume 2023, Article ID 7123175, 10 pages
https://doi.org/10.1155/2023/7123175

Research Article

High Efficiency Secure Channels for a Secure Multiparty Computation Protocol Based on Signal

Yunqi Yang[1,2] and Rui Zhang[1]

[1]*State Key Laboratory of Information Security, Institute of Information Engineering, CAS, Beijing, China*
[2]*School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China*

Correspondence should be addressed to Yunqi Yang; yangyunqi@iie.ac.cn

Received 6 February 2023; Revised 13 March 2023; Accepted 9 April 2023; Published 19 April 2023

Academic Editor: Chien-Ming Chen

Secure multiparty computation (MPC) requires the messages transmitted in secure channels which can provide encryption and authorization to the messages. To implement a secure channel for MPC protocols, researchers have tried some communication protocols, such as TLS and Noise. However, these methods have some limitations. These protocols need a trusted certification authority to provide identity authorization which is difficult for an MPC protocol, and how participants manage the key of each party and how to use the key to establish communication is also a problem. A Signal protocol is an end-to-end encryption communication protocol, which is known as the most secure communication protocol in the world. Based on the Signal protocol, we implemented a signal-based secure multiparty computation protocol, which can run the MPC protocol and transmit messages through a signal-based secure channel. Compared with previous research on the MPC protocol over Signal secure channels, the new MPC client adds group communication of the Signal protocol to transmit messages, which significantly improves the communication efficiency of broadcast messages of MPC protocols. To test the communication efficiency of the new MPC client, we implemented a concrete BLS threshold signature protocol on the new client, comparing the elapsed time of key generation and signing on the new client to that on the client only using Signal end-to-end communication. According to our experiment result, we found that the new client run at least 37.48% faster than the old client on the BLS threshold signature whose number of parties ranges from 3 to 5, if the parties have sent Signal group messages to each other. The more parties in the MPC protocol, the higher the proportion of broadcast messages and the more obvious the performance improvement of the new client. Our work improves the performance of MPC secure channels based on the Signal protocol, especially for complex MPC protocols with many participants.

1. Introduction

Secure multiparty computation (MPC) is a cryptographic problem, which can be expressed as a group of parties, who do not trust each other and need to protect their privacy information but want to perform collaborative computing without a trusted third party. MPC was first proposed by Yao in 1982, who raised a classic millionaire question. To define the security of MPC, researchers constructed an ideal world, in which there is a trusted third party (TTP), who receives information from other parties and then computes the agreed function and sends the result to other parties. However, in the real world, TTP does not exist. The security of MPC in the real world is to be the same as the MPC in the ideal world. If adversaries cannot complete the attack on the protocol in the real world that cannot be achieved in the ideal world, the protocol is considered secure.

Assume n indicates the number of parties and t indicates the upper limit of the number of possible adversaries. For $t < n/3$, it means that the number of adversaries does not reach 1/3 of the total number of parties. In this case, an MPC protocol with fairness and guaranteed output delivery is to be implemented for any function with computational security, provided that there is an authenticated secure channel in an end-to-end synchronous network. Moreover, the channel should be assumed that it is private and information-theoretic security. Therefore, to implement a MPC protocol, finding a secure channel is most important.

To implement a secure channel for MPC, previous researchers have tried many methods, such as TLS and the Noise framework [1]. However, these methods still have some disadvantages. For example, parties of MPC need a trusted certification authority (CA) if they want to build a secure channel with TLS. The Signal protocol is an end-to-end encryption communication method, which provides functions such as identity authorization, key agreement, and encrypted communication. The Signal protocol [2–4] is widely used in many applications, such as Signal and WhatsApp. Some previous studies tried to establish a secure channel for MPC using the Signal protocol. For example, the Zengo-X group implemented BLS threshold signature [5–8] based on the secure channel [9] established by the end-to-end communication function of the Signal protocol.

Our goal is to establish an effective secure channel with Signal for the MPC protocol, such as BLS threshold. Besides end-to-end communication, the Signal protocol also supports group communication, which can increase the communication efficiency of broadcast information of the MPC protocol. A previous secure channel of Signal for MPC only used the function of end-to-end encrypted communication, which transmitted a broadcast message by transmitting a broadcast message into many end-to-end encrypted messages for each receiver and transmitting them in turn. By using the function of group communication in the Signal protocol, we significantly improve the performance of the MPC secure channel with Signal in transmitting broadcast messages. In our new MPC clients based on Signal, end-to-end messages are still sent by Signal end-to-end encryption communication and broadcast messages are sent by Signal group encryption communication which sends messages of multireceiver to the server only once for one round of MPC protocol communication. To measure the communication efficiency of the new MPC client based on Signal, we implement the BLS threshold signature protocol on the client and compare the elapsed time of key generation and signing to the previous secure channels based on Signal. The communication efficiency of our client is much higher than the clients that the Zengo-X group implemented in most situations. In our experiment environment, our new client runs at least 37.48% faster than the old client only using Signal end-to-end communication. The advantages of our clients increase with the growth of number of parties and round of broadcast messages.

2. Background

2.1. Secure Channels for MPC. For almost all MPC protocols, establishing secure channels for round communications between each two parties is necessary. A secure channel provides encryption and authorization, which means that messages transmitted in it will not be obtained by adversaries, nor tempered with or forged by adversaries. Existing MPC protocols have used some method to establish the secure channels, such as TLS and Noise [9]. For example, SCALE-MAMBA [10] is a general purpose MPC framework, which used OpenSSL implementation of TLS to establish secure channels. SCALE-MAMBA sets up a small public key infrastructure (PKI) to implement the authorization of the secure channels. Each party needs to create keys and certificates for the main CA. However, using TLS for the secure channels of MPC still has some problems to be solved. Firstly, the CA in SCALE-MAMBA should be trusted by other parties. In an MPC protocol, it is a problem that who setups the trusted CA. Secondly, every party need to store a lot of public keys and certificates of other parties. If some private keys or certificates lost or change, it is difficult to deal with these accidents. Thirdly, TLS often used in secure client-server communication in Internet, such as the HTTPS protocol. For two-party secure computation, one party can play the role of a web server and the other party plays the client. However, it is not to be applied on a multiparty case.

The Noise framework [1] is also able to establish an end-to-end secure channel for the MPC protocol, which is flexible and allow users to choose a set of security properties. The handshake of Noise is based on Diffie–Hellman (DH) [11] key exchange. Noise has been used in some popular applications to provide end-to-end encryption, such as the Lightning Network and WireGuard. It is feasible to use the Noise framework to establish a secure channel for an MPC protocol. However, the developers need to choose appropriate patterns from a set of handshake patterns based on DH key exchange which enables a specific combination of security properties and implements it from a low-level library which provides the basic crypto algorithm. In addition, the Noise framework still does not solve the problem for-which party plays the role of CA, and how to choose the Noise pattern is a new problem. Choosing an unsuitable pattern may impact the performance of the MPC protocol.

2.2. Signal Protocol. Signal is an end-to-end encryption communication application, in which the Signal protocol used is known as the most secure communication protocol in the world. No third party, including the server, can view the communication content. Signal protocol includes key generation, key agreement, symmetric encryption, digital signature, and other functions. It can be applied on two-party and group communications to ensure the encryption transmission of text, image, voice, video and other data. The Signal protocol has good forward and backward security, ensuring that even if the key used in single communication is leaked, the content of messages sent before or after will not be obtained by adversaries. The Signal protocol uses a server to route messages among clients, which will not participate in MPC protocol if we use Signal to establish the secure channels in it. While users start using Signal, they first need to register on the server with their phone number. Phone number and UUIDwill be used as the identity of the user.

Users generate identity keys and some prekeys for the key agreement when they register on the server and store the public keys on the server. The Signal protocol uses the Double Ratchet algorithm [3] as the key agreement algorithm, which is divided into two parts, DH ratchet and key derivation function (KDF) ratchet. DH ratchet uses the X3DH protocol [4, 12–14] to generate shared secret between two parties. The sender requests the public keys of the receiver from the server and executes three times the elliptic curve Diffie–Hellman (ECDH) key exchange protocol to combine the result of ECDH into a share secret. Then, the KDF ratchet generates message keys used in symmetric encryption from the secret key with SHA-256. For the Signal protocol, DH ratchet provides forward security and KDF ratchet provides backward security. In the process of symmetric encryption, the Signal protocol uses AES256-CBC as the symmetric encryption algorithm and uses SHA256 to add the message authentication code (MAC) for ensuring the integrity of the message.

2.3. BLS Signature. The BLS signature algorithm was delivered by Boneh et al. [6]. The BLS signature uses new hash functions whose results are a point on the elliptic curve and uses a bilinear mapping function e, which maps two points (P, Q) on a bilinear elliptic curve to a number and for a given number x, $e(x \bullet P, Q) = e(P, x \bullet Q)$. An important property of the BLS signature is that signatures can be aggregated. The signatures of multiple signers on the same message can be aggregated into one signature. The aggregated BLS signature algorithm [6–8] is as follows:

Assume the parameters par $= (q, G_1, G_2, G_t, e, g_1, g_2)$ is a bilinear group, where G_1, G_2, G_t are groups of prime order q, e is a bilinear map $e\colon G_1 \times G_2 \longrightarrow G_t$, and g_1, g_2 are generators of groups G_1, G_2. Assume H_0 and H_1 are two collision-resistant hash functions $H_0\colon \{0,1\}^* \longrightarrow G_2$ and $H_1\colon \{0,1\}^* \longrightarrow Z_q$.

(1) Key Generation and Aggregation: Each signer i chooses sk_i from Z_q, compute $pk_i \leftarrow g_2^{sk_i}$. For $\{pk_1, \ldots, pk_n\}$, compute $apk \leftarrow \prod_{i=1}^{n} pk_i^{H_1(pk_i, \{pk_1, \ldots, pk_n\})}$.

(2) Signing: Each signer computes $s_i \leftarrow H_0(m)^{a_i \bullet sk_i}$, where $a_i \leftarrow H_1(pk_i, \{pk_1, \ldots, pk_n\})$. Signers send the signatures to other signers and compute the aggregated signature $\sigma \leftarrow \prod_{j=1}^{n} s_j$.

(3) Verification: The verifier computes the following formula to verify the signature. If $e(\sigma, g_2^{-1}) \bullet e(H_0(m), apk) = 1$, output 1. The signature is valid.

Based on the BLS signature, an MPC protocol of the BLS threshold signature [6, 7, 15] is to be constructed. The number of the parties joining the protocol is n, and $t+1$ $(1 < t \leq n-1)$ signers of the group can sign a signature on behalf of the group. Any $t+1$ parties in the group sign the same signature for the same message. The main communication process of the BLS threshold signature protocol to be used in this paper is introduced as follows:

(1) Key generation: the process of key generation needs four rounds of communication.

  (i) Each party generates his key pair *sk* and *pk* and sends the hash commitment of *pk* to other parties. This round uses broadcast communication.

  (ii) Each party sends his *pk* and the commitment parameter to other parties. This round uses broadcast communication.

  (iii) Each party shares his *sk* by Feldman verifiable secret sharing (VSS). Other parties will receive different shared secret. So, this round uses end-to-end communication.

  (iv) Each party sends the proof of *sk* and generates the keys. This round uses broadcast communication.

(2) Signing: The process of signing only needs one round communication.

  (i) Each party generates his partial signatures and sends to other parties. This round uses broadcast communication.

3. Materials and Methods

3.1. Secure Channels Based on Signal. Based on the rust library of the Signal protocol, we implement an MPC client which can run an MPC protocol as one of the parties of the protocol and transmit the messages among the clients using secure channels. Compared with previous research [9], our client adds support for the group communication in the Signal protocol, which can improve the performance of the secure channels in transmitting broadcast messages.

Before using the MPC client, the user needs to register a Signal account on the server. This step needs an Android or iOS Signal client which connects the same Signal server as the MPC client. Signal app opened its source codes on GitHub, so it is easy to change the server address in the open-source codes of the Signal client and compile them to obtain the client. When the user registers a Signal account on the Signal server, the client generates an identity key and some prekeys with the signatures [2, 16, 17] signed by the identity key, and then stores their public key on the server. The phone number and UUID are the identifications of the user. A Signal account can be bound with one phone client and several desktop clients. The MPC client requests registration from the server as same as the desktop client, and the server returns a message including some necessary information for registration. The phone client scans the QR code showed by the MPC client using the function of adding new device and establishes a temporary encrypted communication to the MPC client. The MPC client obtains the identity key and some other information, and then requests the server for creating a new device, generates its own prekeys, and stores the public key on the server. Then, the MPC client finishes logging in the Signal account.

To run an MPC protocol with the MPC clients who have logged in the Signal account, a group JavaScript Object Notation (JSON) file with Signal addresses and public keys of all the parties is necessary. After logging in a Signal account, the MPC client can show this information of the device in the JSON format. Users combine them into a file and add the same file to every client to join the MPC client. Differing from the official Signal client sending messages to all the device of the signal account of the sender and the receiver, the MPC client only sends messages to the target device which is the MPC client joining the MPC protocol. Each party in the group file only represents a device of a Signal account. When the MPC client sends messages to the server, the packages only include the ciphertext of the target device, and the ciphertexts of other devices in the packages are empty. This design can simplify the information sending process in the Signal protocol and reduce the amount of the process of requesting receiver information from the server.

When the MPC client running an MPC protocol, the client establishes a secure WebSocket connection to send outcoming messages and listen for incoming messages. The key agreement and symmetric encryption algorithm are implemented in the rust library of Signal. When the client wants to send an end-to-end message through the secure Signal channel, it checks if the key session exists. If the sender and the receiver have not sent messages before, the sender needs to request for the prekeys of the receiver from the server and runs the key agreement protocol (X3DH) [3, 4] to build a new session. Then, the message will be encrypted by the message key generated by X3DH and be sent to the server using the Signal server API. When the receiver obtains the encrypted message, it handles the message in a process similar to that of the sender, transmitting the decrypted message to the MPC protocol as the incoming message. To filter the MPC protocol messages, the cipher messages sent by the MPC client are added a fixed tag at the beginning. When the MPC client receives Signal messages, it checks the tag and drops the messages without the tag.

3.2. Group Communication. Group communication is a new function in the Signal protocol. Compared with previous research of secure channels of Signal for MPC, we implement the group communication function of Signal on the MPC client for handling broadcast messages. In fact, early version of Signal APP also has the group function (Group v1). Group v1 is similar to the secure channels for MPC of the ZenGo-X team [9], which stores the group member information locally and sends group messages by several end-to-end messages. New version group communication of Signal (Group v2) is based on the function of the sealed sender. The sealed sender [18] is to hide sender's information from the server. After the encryption of the normal Signal protocol, the Signal APP client encrypts the ciphertext once more with the identity key of the sender and the receiver. To prevent spoofing of sender's identity, the sender certificates are added and be encrypted with the ciphertext of normal Signal encryption. The Signal client requests the sender certificates from the server. When a Signal client receives a sealed sender message, it can check the sender certificate's validity. Another important data for sealed sender are the unidentified access key which prevents the abusing of the server. Without identity authentication, some clients may request to send messages frequently. The unidentified access keys are derived from profile keys, which are exchanged with contacts or other people that the user have conversation with. A signal account can be set on the sender to allow anyone to send sealed sender messages to it, which set the unidentified access key of the account to all zero.

Group v2 [19, 20] is first to solve the problem of the store of group member's information. Store the information locally in every party's device is hard to manage the information. In Group v2, the group member's information is encrypted by the master key and stored on the server [20, 21]. When a new member is invited in the group, the inviter shares the group master key via Signal end-to-end message. In fact, we do not focus on the group member management of Group v2 in this paper.

Another important function with Group v2 is sending multireceiver messages, which is based on the sealed sender. In Group v2 conversations of Signal APP, encrypted messages are sent by this function. When the client wants to send multireceiver messages, it needs to generate a distribution id, which is a random UUID. The distribution id is used to identify the sender key. The sender key is similar to the chain key in normal encryption of the Signal protocol, which can generate message keys by KDF ratchet. After encrypted by the sender key, the messages will be encrypted with multireceiver sealed sender encryption. The pseudocode of multireceiver sealed sender encryption is shown in Algorithm1. Firstly, the client generates a random number M, and an ephemeral asymmetric key pair E and a one-time symmetric key K are derived from M. The messages encrypted by sender keys before will be encrypted with AES-GCM-SIV (AEAD_Encrypt in Algorithm 1) by the symmetric key for the second-round encryption. For each receiver, the shared secret is generated with key agreement between the ephemeral key and the identity key of the receiver. Then, the encrypted message keys C_i are obtained by XOR M and shared secret. The authentication tags of the encrypted message keys AT_i are obtained by HMAC (hash-based message authorization code)-based KDF (HKDF) [22, 23] of the keys and the shared secret that the key agreement is the result of two parties' identity key. Finally, the results of multireceiver sealed sender encryption contain the UUID, encrypted message key, authentication tag for each receiver, the public key of the ephemeral asymmetric key, and the ciphertext.

In our MPC client, we implement the group communication of the Signal protocol for the MPC protocol. Before sending group communication messages, the MPC client checks if the available distribution id and sender key session exist. If they do not exist, the MPC client generates them and sends a sender key distribution message to other parties in the MPC protocol, ensuring every party has the sender key of the sender device and the distribution id. Sender certificates are necessary information in sealed sender messages. The MPC client requests the sender certificates before sending the group communication messages by the same server API that the Signal APP uses. The Signal accounts of the MPC clients are set to allow anyone to send sealed sender messages to them on the server, so the values of unidentified access keys of them are all zero. For simplifying the process of communication, we ignore the abusing of sealed sender messages so that the MPC clients need not to exchange their profile keys.

```
ENCRYPT(message, R_i):
  M = Random(32)
  r = KDF(label_r, M, len = 64)
  K = KDF(label_K, M, len = 32)
  E = DeriveKeyPair(r)
  for i in num_recipients:
    C_i = KDF(label_DH, DH(E, R_i) ||E.public ||R_i.public, len = 32) XOR M
    AT_i = KDF(label_DH_s, DH(S, R_i) ||E.public ||C_i ||S.public ||R_i.public, len = 16)
  ciphertext = AEAD_Encrypt(K, message)
  return E.public, C_i, AT_i, ciphertext
DECRYPT(E.public, C, AT, ciphertext):
  M = KDF(label_DH, DH(E, R) ||E.public ||R.public, len = 32) xor C
  r = KDF(label_r, M, len = 64)
  K = KDF(label_K, M, len = 32)
  E′ = DeriveKeyPair(r)
  if E.public ! = E′.public:
    return DecryptionError
  message = AEAD_Decrypt(K, ciphertext)//includes S.public
  AT′ = KDF(label_DH_s, DH(S, R) ||E.public ||C ||S.public ||R.public, len = 16)
  if AT ! = AT′:
    return DecryptionError
  return message
```

Algorithm 1: The pseudocode of the multireceiver sealed sender (from the rust library code of the Signal protocol).

3.3. *BLS Threshold Signature on Secure Channels of Signal.*

BLS threshold signature includes key generation, signing, and verifying three steps, in which key generation and signing need to send end-to-end or broadcast messages to other parties. Key generation has four rounds of communication, in which round 1, 2, and 4 are broadcast communication and round 3 is end-to-end communication. Signing has only one round broadcast communication. In our MPC client for the BLS threshold signature, the end-to-end messages are sent by normal end-to-end encrypted communication of the Signal protocol and broadcast messages are sent by group communication of the Signal protocol. We did not build a real group of Group v2, only using the multireceiver messages function of Group v2 to send broadcast messages.

The flow chart of the MPC client is shown in Figure 1. The first two steps of both two kinds of communications are the initial steps which register the client on the server and collect the information of the devices of parties. Completing these steps, the MPC clients can send secure messages through the secure channel base on the Signal protocol and know other parties' addresses in the Signal protocol of the MPC protocol. Then, the BLS threshold signature can run by the MPC clients. In BLS threshold signature module, the receivers of messages are expressed as the order in member list, and for broadcast messages the receiver is none. When the messages are transmitted into the Signal protocol module, the client replaces the receivers of the messages with the real Signal address. For broadcast messages, the receiver addresses are lists of all the parties except the sender. Then, according to the type of the messages, the MPC client sends these messages by end-to-end secure channels or group secure channels.

Depending on the message type, the client sends messages in different ways. If the message is an end-to-end message, such as the round 3 of the key generation of the BLS threshold signature, the message is sent as shown on the right side of Figure 1. When a Signal client sends messages to someone at first time, it needs to require the receiver's keys and some necessary information from the server. Then, the client generates new chain key sessions using two parties' keys if the sessions does not exist or gets the chain key sessions from the local store if the available sessions exist. The messages to be sent will be encrypted by the message key, which is generated by the chain key session, and then the encapsulated ciphertext is sent to the server.

If the message is a broadcast message, such as the round 1 of signing and round 1, 3, and 4 of key generation of the BLS threshold signature, the messages are sent as shown on the left side of Figure 1. In previous research of MPC over Signal [9], the broadcast messages are divided into several end-to-end messages and sent successively. However, in our MPC client, the broadcast messages are sent by group communication of the Signal protocol. The sender certificate is used to verify the identity of the sender in group communication of the Signal protocol, which is generated by the server and is valid for a short time. The MPC client requires the sender certificate of the sender from the server before sending a Signal group message. When the client sends Signal group messages to new parties, it also needs the keys of the receivers from the server. Compared to the end-to-end communication, group communication needs the information of all the devices of the receivers instead of the devices of the MPC clients. Then, the MPC client generates a sender key session if no sender key session is available or loads a sender key session from the local store. The information of the sender key session is distributed to other parties of the MPC protocol by end-to-end messages before sending group messages at the first time of sending to them. One-time steps in the sending process are marked with

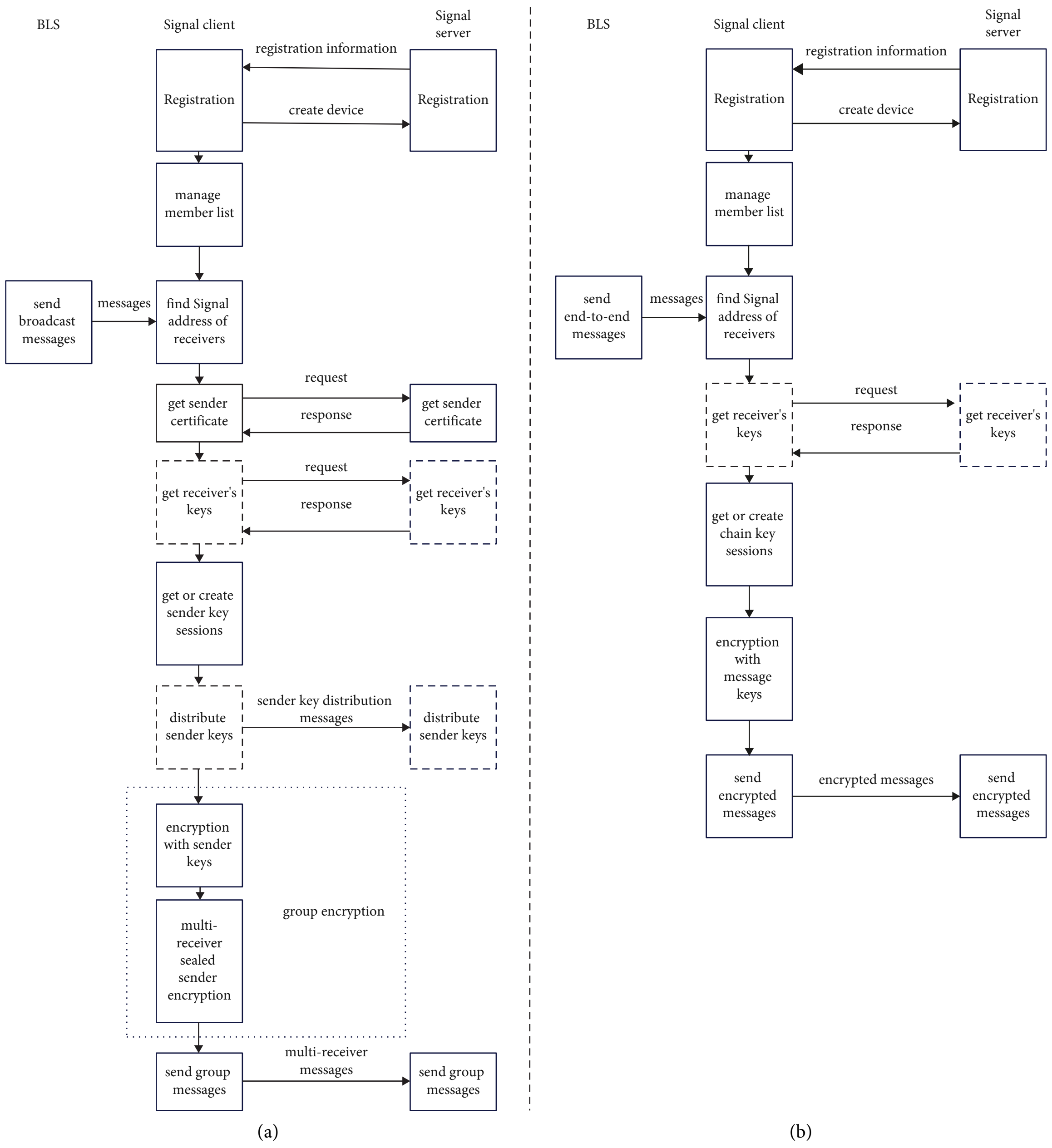

Figure 1: Flowchart of the MPC client, including (a) group communication and (b) end-to-end communication.

a dashed border in Figure 1 and will not be executed after the first time of sending messages. The plaintext of the group message is encrypted two rounds, as shown in Figure 1. Then, the ciphertext is sent to the server using the API of multireceiver message.

The MPC client handles the received messages as the type of the messages. End-to-end messages are decrypted using the chain key sessions and the plaintexts are sent to the MPC protocol module for calculation. The group messages are received as the sealed sender messages. The client needs to decrypt the sealed sender messages to obtain the information of the sender of the message, and then decrypts the messages using corresponding sender key sessions to obtain the plaintext for the MPC protocol.

4. Results and Discussion

4.1. Experiment Environment. The MPC clients are installed on the Aliyun Cloud Server, which connects the Signal official staging server (https://chat.staging.signal.org). As shown in Table 1, the operation system of the cloud server is Ubuntu 20.04 and the server is located in Singapore. The size of the memory of the cloud server is 4 GB and the system disk is 80 GB. The Rust version to compile the MPC client is nightly-2021-09-16. The original version of the Signal protocol library is 0.17.0, and we implement our MPC client based on the original version of the Signal protocol.

In our experiment, our MPC clients connected the Signal official staging server. In fact, we do not recommend connecting the clients to the official server in practical applications. The version of the official server is uncontrolled and the server may monitor and prohibit abnormal use. For purposes other than research, we recommend using the server source code to build the server.

4.2. Experiment Method. The goal of the experiments is to compare the operational efficiency between the MPC clients using Signal group communication and the MPC clients only using end-to-end communication. In the BLS threshold signature, key generation and signing need to use the secure channels to transmit information. In the process of key generation, clients send and receive 3 rounds broadcast messages and 1 round end-to-end messages. In the process of signing, clients only send and receive 1 round of message. By using Signal group communication, we significantly improve the communication efficiency of broadcast messages. To study the impact of the number of participants on the operational efficiency, we, respectively, tested three kinds of the BLS threshold signature protocol, that is, we set "$t = 2$, $n = 3$," "$t = 3$, $n = 4$," and "$t = 4$, $n = 5$."

To run the MPC protocol, multiple clients need to run at the same time. Different starting time of different clients will lead to different elapsed times because the MPC client send next round messages after receiving all the messages of the previous round. As a result, the elapsed time of the last started client is the shortest because it needs not to wait for other parties' messages at the first round. We choose the elapsed time of the last started client as the experiment result. At least 5 tests will be conducted for each environment, and the average value will be taken to evaluate the operating efficiency. When the client sends messages to other parties for the first time, whether it is group communication or end-to-end communication, it needs to perform some additional steps, such as requiring receivers' keys and distributing sender keys. These steps will not be performed when subsequent messages are sent and these steps in group communication are more complex than these in end-to-end communication. In this paper, we mainly focus on the operational efficiency of subsequent rounds communication. For the first-time communication, we tested some typical situation of MPC protocols to show the impact on the communication efficiency of new and old clients. Before running the MPC protocol, the MPC clients have acquired necessary information. The time of the preparation steps will not be considered.

4.3. Experiment Result

4.3.1. Key Generation. The elapsed time of key generation of the BLS threshold signature is shown in Figure 2. Group represents the MPC clients using both Signal end-to-end communication and group communication, and Zengo represents the MPC clients that the Zengo-X team implemented [9] using only end-to-end communication. Signal group communication requires at least 3 parties, so the experiments start with 3 party protocols. In Figure 2, the elapsed times do not include the time of one-time steps and the JSON file of parities information has been added into the clients.

Table 1: Experiment environment.

Operation system	Ubuntu 20.04
Server location	Singapore
Memory	4 GB
System disk	80 GB
CPU cores	2
Rust version	Nightly-2021-09-16
Libsignal-client version	0.17.0 (edited)

According to the experiment result of key generation, compared to the Zengo client, the Group client has significant performance advantages, even if the number of parties is only three. In "$t = 2$, $n = 3$" protocol, compared with the Zengo client, the operational efficiency of the Group client is improved by 37.48%. With the growth of number of parties, the elapsed time of both Group and Zengo increase, and compared with Zengo client, the advantages of the operational efficiency of Group increases more significantly.

4.3.2. Signing. The elapsed time of the signing of BLS threshold signature is shown in Figure 3. In Figure 3, the elapsed times do not include the time of one-time steps and the JSON file of parities information has been added into the clients. Compared with key generation, the process of signing is simpler, only transmitting one round broadcast messages. Therefore, the elapsed time of signing is much shorter than that of key generation, and the elapsed time of group hardly increase with the growth of the number of parties. For an t+1 parties signing protocol, if only using end-to-end communication, every client needs to send and receive t end-to-end messages. The encryption of group messages is more complex than end-to-end messages, but the encryption time of group messages increases little with the growth of the number of parties. In "$t = 2$, $n = 3$" protocol, compared with the Zengo client, the operational efficiency of the Group client is improved by 68.83%. With the growth of number of parties, compared with Zengo client, the efficiency advantage of Group increases rapidly.

4.3.3. First-Time Communication. The elapsed time of first-time communication of some typical situation is shown in Table 2. The three situations are "$t = 2$, $n = 3$, signing," "$t = 2$, $n = 3$, key generation," and "$t = 4$, $n = 5$, key generation." According to the result, we can see that the communication efficiency of the new MPC client using group communication is not as good as that of the Zengo client in some situations. 3 parties signing includes only one round of broadcast messages and the elapsed time of Group is nearly twice as that of the Zengo client. With the growth of the number of parties and the rounds of broadcast messages, the MPC client of Group performs better than the Zengo client. 5 parties key generation include 3 rounds broadcast messages and the elapsed time of Group is shorter than that of the Zengo client even if the clients have not sent group messages to other parties.

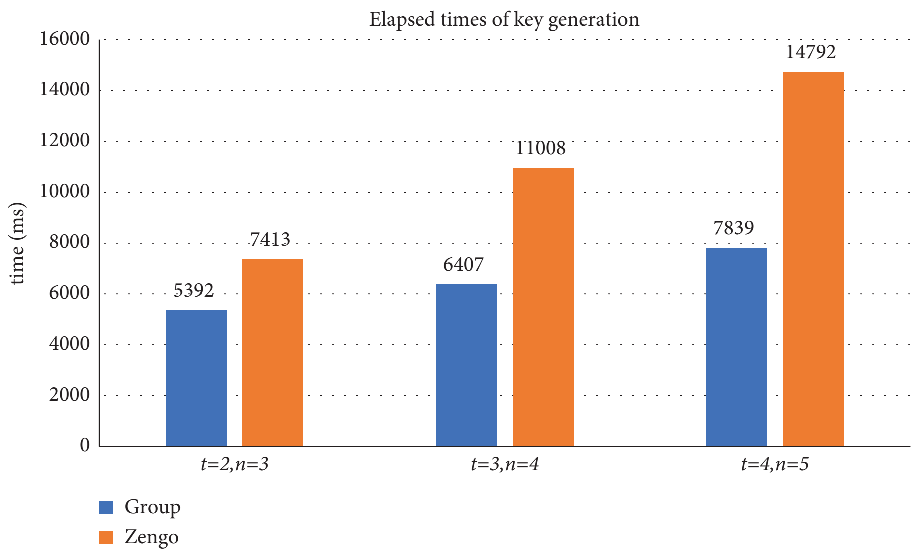

Figure 2: Elapsed time of key generation, including "$t = 2$, $n = 3$," "$t = 3$, $n = 4$," and "$t = 4$, $n = 5$."

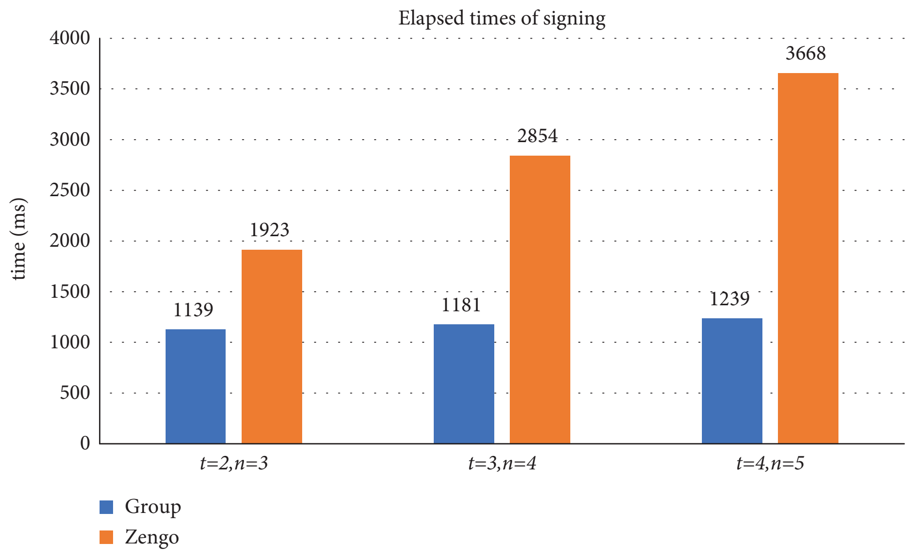

Figure 3: Elapsed time of signing, including "$t = 2$, $n = 3$," "$t = 3$, $n = 4$," and "$t = 4$, $n = 5$."

Table 2: Elapsed time (ms) of first-time communication.

Clients	$t = 2$, $n = 3$ signing	$t = 2$, $n = 3$ key generation	$t = 4$, $n = 5$ key generation
Group	4226	8759	13356
Zengo	2145	7747	15300

4.4. Discussion. According to the experimental result, the MPC client using both Signal end-to-end communication and group communication has significant advantages over the MPC client by only using Signal end-to-end communication in the BLS threshold signature protocol. Actually, compared with other steps, the encryption of multireceiver sealed sender and receiving messages costs little time because in the signing process the number of parties has little impact on the elapsed time of group clients. To send a broadcast message using Signal end-to-end communication, the client needs to send the messages to the server several times, one receiver at a time, and the elapsed time of old clients increases rapidly with the growth of the number of parties. As a result, sending requests to the server takes a lot of time. Although generating a group sealed sender message is more complex than generating a normal Signal message, as its process mainly runs locally. Using Signal group communication, the MPC client only sends a message to the server once, no matter how many receivers there are. For any MPC protocol sending broadcast messages, using Signal group communication can effectively reduce the elapsed time of the protocol if the parties have sent group messages to each other before.

Our experiment still has some limitations. We assume that the MPC clients have sent group messages before so that some one-time steps were not included in performance comparison, such as sending sender key distribution messages and getting receivers' keys from the server. Although Signal end-to-end communication also has some one-time steps, the one-time steps of Signal group communication obviously take more time than those of end-to-end communication. The MPC clients distribute sender keys using Signal end-to-end messages, which actually add an additional round of end-to-end messages. In addition, Signal group communication requires the keys of all the devices of receivers, which means the clients need to request at least twice the keys from the server. As a result, in some extreme cases, using group communication may reduce communication efficiency: (1) the parties of the MPC protocol have not sent Signal group messages to each other, (2) the protocol is simple, including only one round broadcast messages to be sent, and (3) the number of the parties of the MPC protocol is very small, such as 3 parties. When the above three points are met, the use of group communication may not improve communication efficiency of the MPC protocol. We tested three typical first-time communication situations of MPC protocols, and the group client actually performs worse in some cases where there are few parties and rounds situation. However, when there are many parties and rounds of broadcast, the Group client performs better than the Zengo client even in first-time communication.

5. Conclusions

Our research established secure channels of MPC protocols based on Signal group communication and end-to-end communication and tested the communication efficiency of the secure channels in running the BLS threshold signature protocol. Compared with the MPC client only using Signal end-to-end communication, our new MPC client has significant improvement of communication efficiency in the key generation and signing of the BLS threshold signature when the number of parties is greater than 3 and they have previously sent Signal group messages to each other. In our experiment environment, our new client runs at least 37.48% faster than the old client only using Signal end-to-end communication. The secure channels based on the Signal protocol provide encryption and authorization to the messages which are required by the MPC protocols. Our research improves the performance of the MPC secure channels based on the Signal protocol on the complex protocols of multiple parties. The improvement of communication efficiency increases with the number of parties and the rounds of broadcast messages.

There are also some points that can be improved for our research. Our MPC clients still need to collect the information manually. We mentioned that the Group v2 in the Signal protocol can store the encrypted members' information on the server which can manage the members' information easily. Combining the function and MPC can simplify the preparation steps. Our research studies have not considered the time cost of the sender key update. For long-term use, sometimes, the process of generating new sender keys may increase the elapsed time of the MPC protocol. These points can be studied in future research.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by the National Key R&D Program of China under Grant no. 2017YFB1400700.

References

[1] T. Perrin, "The Noise protocol framework," 2018, https://noiseprotocol.org/noise.pdf.

[2] T. Perrin, "The XEdDSA and VXEdDSA signature schemes," 2016, https://signal.org/docs/specifications/xeddsa/xeddsa.pdf.

[3] T. Perrin and M. Marlinspike, "The Double ratchet algorithm," 2016, https://signal.org/docs/specifications/doubleratchet/doubleratchet.pdf.

[4] M. Marlinspike and T. Perrin, "The X3DH key agreement protocol," 2016, https://signal.org/docs/specifications/x3dh/x3dh.pdf.

[5] Stepan, "BLS signatures: better than Schnorr," 2018, https://medium.com/cryptoadvance/bls-signatures-better-than-schnorr-5a7fe30ea716.

[6] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the weil pairing," *Journal of Cryptology*, vol. 17, no. 4, pp. 297–319, 2004.

[7] B. Dan, M. Drijvers, and G. Neven, "Compact multi-signatures for smaller blockchains," in *Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security*, Springer, Berlin, Heidelberg, October, 2018.

[8] D. Galindo, L. Jia, M. Ordean, and J.-M. Wong, "Fully distributed verifiable random functions and their application to decentralised random beacons," in *Proceedings of the 2021 IEEE European Symposium on Security and Privacy (EuroS&P)*, vol. 96, Vienna, Austria, September 2020.

[9] O. Shlomovits, "MPC-Over-Signal," 2021, https://medium.com/zengo/mpc-over-signal-977db599de66.

[10] A. Aly, K. Cong, and D. Cozzo et, "SCALE–MAMBA v1.14: documentation," 2021, https://homes.esat.kuleuven.be/%7Ensmart/SCALE/.

[11] T. Okamoto and D. Pointcheval, "The gap-problems: a new class of problems for the security of cryptographic schemes," in *Proceedings of the 4th International Workshop on Practice and Theory in Public Key Cryptography*, Public Key Cryptography, Berlin, Heidelberg, June 2005.

[12] A. Langley, M. Hamburg, and S. Turner, "Elliptic curves for security," Jan-2016, http://www.ietf.org/rfc/rfc7748.txt;Internet Engineering Task Force; RFC 7748 (Informational); IETF.

[13] C. Kudla and K. G. Paterson, "Modular security proofs for key agreement protocols," in *Proceedings of the Advances in Cryptology - ASIACRYPT 2005: 11th International Conference on the Theory and Application of Cryptology and Information Security*, Berlin, Heidelberg, December 2005.

[14] S. Blake-Wilson, D. Johnson, and A. Menezes, "Key agreement protocols and their security analysis," in *Proceedings of*

the Crytography and Coding: 6th IMA International Conference Cirencester, London, UK, December 1997.

[15] S. Agrawal, P. Mohassel, P. Mukherjee, and P. Rindal, “DiSE: distributed symmetric-key encryption,” in *CCS*, pp. 1993–2010, ACM, New York, NY, USA, 2018.

[16] C. Cremers and M. Feltz, “One-round strongly secure key exchange with perfect forward secrecy and deniability,” 2011, https://eprint.iacr.org/2011/300.

[17] J. P. Degabriele, A. Lehmann, K. G. Paterson, N. P. Smart, and M. Strefler, “On the joint security of encryption and signature in EMV”, cryptology ePrint archive,” 2011, https://eprint.iacr.org/2011/615.

[18] jlund, “Technology preview: sealed sender for signal,” 2018, https://www.signal.org/blog/sealed-sender/.

[19] jimio, “Technology preview: signal private group system,” 2019, https://www.signal.org/blog/signal-private-group-system/.

[20] M. Chase, T. Perrin, and G. Zaverucha, “The signal private group system and anonymous credentials supporting efficient verifiable encryption,” 2019, https://eprint.iacr.org/2019/1416.

[21] M. Chase, S. Meiklejohn, and G. Zaverucha, “Algebraic MACs and keyed-verification anonymous credentials,” in *ACM CCS 2014*, G.-J. Ahn, M. Yung, and N. Li, Eds., pp. 1205–1216, ACM Press, New York, NY, USA, 2014.

[22] H. Krawczyk and P. Eronen, “HMAC-Based extract-and-expand key derivation function (HKDF),” 2010, http://www.ietf.org/rfc/rfc5869.txt.

[23] H. Krawczyk, M. Bellare, and R. Canetti, “HMAC: keyed-hashing for message authentication,” 1997, http://www.ietf.org/rfc/rfc2104.txt.