Secure P2PSIP-based conference system with dynamic scalability

SHI Yuan-ying

School of Telecommunications and Information Engineering, Xi’an University of Posts and Telecommunications, Xi’an, China

Abstract. P2PSIP (Peer-to-Peer SIP)-based conference system suffers from more security threats than SIP-based conference systems. Stronger security measures are required for such system, but they often incur heavy burden on low-capability peers and the conference size will be limited. Thus, we propose a dynamic scalable and secure architecture for P2PSIP-based conference system. In this architecture, most security operations as well as conference control are performed by a number of high-performance peers known as CCs (conference controllers). High scalability is achieved by adding and deleting CCs as needed. Corresponding to this system model, a distributed conference key management scheme based on one-way accumulators is presented and described in detail. Security and performance analysis shows that this scheme can provide forward/backward secrecy while requiring low computation and storage cost for resource-restricted devices.

Keywords: P2PSIP, conference, security, scalability, one-way accumulators, key management

1. Introduction

SIP (Session Initiation Protocol)[1] is an application-layer signaling protocol designed for establishing media sessions. It is now used for a variety of multimedia applications, such as VoIP, instant messages and multiparty conferences. Since traditional SIP is in client-server mode, most SIP-based conference systems are based on centralized architecture. However, such architecture places a heavy CPU and network bandwidth burden on the central server, which gives rise to many problems such as single-point failure and weak scalability.

P2PSIP has emerged in the last few years as a promising approach to solve the performance and scalability problems of traditional SIP. It eliminates the need for central servers and utilizes P2P overlay to locate users and to route messages. If conferencing application is built on the P2PSIP architecture model, conference management, media mixing and distribution functions can be assigned to terminal devices. Thus, users can set up conferences without relying on a conference server deployed by a service provider. The system flexibility will be improved.

Security is of primary concern for conference systems. In P2PSIP-based conference system, the security problems become more complicated. On one hand, because the system is organized in a distributed mode and lack of central control, malicious peers are hard to be identified. Then the threats of eavesdropping, impersonation, unauthorized data modification, and denial-of-service are increased. Therefore, stronger security measures such as authentication, access control and data encryption are required for the system. On the other hand, P2PSIP-based conference system often consists of heterogeneous peers with different processing power, network bandwidth and online duration. Above security measures will incur heavy storage, computation and communication burden on low capacity peers. Consequently, such peers may become system bottleneck and limit the conference size. Furthermore, in P2P environments, participants join and leave the conference frequently. Whenever the conference composition changes, secret information of the conference

* E-mail address: shi_yuanying@163.com
needs to be updated accordingly to provide forward and backward secrecy. For large-scale conferences with hundreds of members, how to reduce the update overhead is also an important issue.

In order to provide strong security while keeping good scalability, this paper presents a hierarchical architecture for P2PSIP-based conference system. In this architecture, most security operations as well as conference control are performed by some high performance nodes, while security cost for low capacity nodes is quite small. We describe this system architecture in section II. Section III demonstrates the security mechanism in detail, which focuses on a new decentralized conference key management scheme. Then, security and performance of the new scheme is analyzed in section IV. Finally, some conclusions are given in section V.

2. Architecture Of P2PSIP-based Conference System

Our P2PSIP-based conference system is shown in Fig.1. In the system, nodes wishing to obtain conferencing service must first join a DHT (Distributed Hash Table) network. For simplicity, we take Chord[2] as example, but other DHT structures such as CAN[3], Pastry[4] can also be applied to our system. According to their capability, peer nodes are classified into two types: superior nodes and ordinary nodes. Superior nodes have powerful processing capability, high bandwidth and long online time. They can not only act as conference participants, but also conference controllers (CCs) to offer such services as conference management, streams mixture and distribution, etc. Ordinary nodes are resource-restricted devices, such as PDA and smart phone. They can only act as conference participants. To join a conference, ordinary nodes must first use P2PSIP protocols such as SOSIMPLE[5] to establish sessions with a CC. We suggest using locality-aware P2P schemes such as[6] in our system, so that ordinary nodes can connect to an adjacent CC to reduce communication delay.

A conference always has one primary CC (PCC) and zero or more secondary CCs (SCC). The number of SCCs depends on the conference scale. When a node initiates a conference, it must choose an adjacent superior node (or itself if it is a superior node) as PCC. Then the initiator sends a list of initial conferees to PCC using the scheme defined in RFC5366[7]. After getting the list, PCC will perform the following operations.

Step 1: PCC generates a unique conference URI (Conf-ID) and adds its own address (SIP URI or IP address) to the conference contact list. Then it stores the Conf-ID with the contact list in the overlay. Hence, new participants can retrieve PCC’s address by searching the Conf-ID and join the conference through PCC.

Step 2 (optional): If the number of initial conferees is too large, PCC may add one or more superior nodes (SCCs) as its child nodes. Chosen SCCs are preferably located in participants-intensive areas so as to reduce the communication delay. For each SCC, PCC selects an appropriate subset from the list of initial conferees and sends it to the SCC. Then, PCC adds all SCCs’ URIs to the conference contact list.

Step 3: In order to add participants to the conference, PCC, as well as all existing SCCs, sends INVITE messages to the participants of whom they are in charge.

Once the initialization process described above is completed, the conference may be divided into several groups. Each group is controlled by a CC, and all CCs cooperate to manage the conference. It is the CC that is responsible for group maintenance, including participants’ management, audio/video mixing, conference key establishment, and interaction with other groups.
As the conference goes on, new participants may join, and some participants may leave. To join an ongoing conference, a new participant must get the conference contact list first. Then, it chooses an adjacent CC from the list and establishes connection with the CC. With the increase in conference size, some CC might carry a heavy load. To keep load balance, every CC should record the processing load of itself and that of its child nodes (SCCs). Then a heavy-loaded CC can transfer new arrivals to a light-loaded child node by sending REFER messages. If the CC is a leaf node or all its child nodes are also heavy-loaded, it may add a new child node and transfer new participants to it. Thus, all CCs in the conference may form a tree structure and PCC is the root. Each node in the CC tree must store information about its parent node and all its child nodes.

When a member is leaving a conference, it only needs to send a BYE message to its group CC. Similarly, if all group members of a leaf CC have gone, it may leave by sending a BYE message to its parent node. Other CCs will then be informed by update messages, and PCC can inform the initiator of the changes and update the conference contact list accordingly.

3. Security Mechanism

In our P2PSIP-based conference system, security algorithm is composed of three modules: access control module, data protection module and conference key management module.

3.1 Access Control Module

This module ensures that only authorized users can join a conference. It is achieved by using the authentication procedure and an access control list (ACL) which is maintained by the conference initiator. When a member joins a conference, it must carry out mutual authentication with the CC that it connects to. Similarly, when a SCC joins, it must perform mutual authentication with its parent node. We suggest the use of identity-based or certificateless-based authentication mechanisms[8][9]. Because of no need for an expensive PKI (Public Key Infrastructure), such mechanisms are viable for large scale implementation.

In our system, a conference-aware user can participate in dial-in mode[10]. Even if the user is authenticated successfully, he may be an unwanted conferee. For example, a student, who has the right to use the conferencing service in the campus network, should be prevented from joining a discussion only among teachers. Thus, when the initiator is informed of a newcomer, he must check the ACL to examine whether the user has the right to join. If false, the user must be rejected.

3.2 Data Protection Module

This module is used to ensure the confidentiality and integrity of the signaling data and media data exchanged in a conference. To gain higher security, signaling data used to create a conference and to add a new participant/SCC are protected by asymmetric cryptography. After the shared conference key is established, symmetric cryptography, such as RC5, DES, is used to protect the privacy data due to its fast speed and low cost.

3.3 Conference Key Management Module

This module is used to construct a shared conference key and to update the key when the membership changes. It is designed based on collision-free accumulators defined in[11]. Accumulators are functions to summarize a large number of values in one value. In our system, each participant is required to generate a random value. The conference key is constructed based on values of all participants. The detailed algorithm is explained below.

1) Notations

We use the following notations in the rest of this paper.

\[
\begin{align*}
SCC_i & & \text{i}^{\text{th}} \text{secondary CC} \\
G_i & & \text{i}^{\text{th}} \text{group} \\
u_{ij} & & \text{j}^{\text{th}} \text{member of i}^{\text{th}} \text{group} \\
y_{ij} & & \text{representative of} \ u_{ij} \\
\omega_{ij} & & \text{witness of} \ u_{ij}
\end{align*}
\]
Y_{SCC_i}/Y_{PCC} \quad \text{representative of } SCC_i/PCC

BY_{SCC_i}/BY_{PCC} \quad \text{branch representative of } SCC_i/PCC

B_{\omega_{SCC_i}} \quad \text{branch witness of } SCC_i

2) System Initialization

When a superior node becomes PCC, it chooses two large strong prime \( p, q \) and computes \( N=pq \) and \( \Phi(N)=(p-1)(q-1) \). It then randomly chooses a suitably-large base \( x \) that is relatively prime to \( N \). We call \( x \) the conference seed.

3) Conference Key Establishment

Without loss of generality, we let \( SCC_1, SCC_2, \cdots, SCC_k \) be the SCCs invited by PCC during conference initialization. Thus, the conference is divided into \( k+1 \) groups, i.e. \( G_{PCC}, G_{SCC_1}, \cdots, G_{SCC_k} \). After authentication procedure, PCC sends \( N, \Phi(N) \) and \( x \) securely (i.e. encrypts the data with \( SCC_i \)'s public key) to \( SCC_i \) \( (i\in\{1,2,\cdots,k\}) \), and all CCs multicast \( N \) to their group members.

Suppose there are \( l \) members in group \( G_i \) \( (G_i \in \{G_{PCC}, G_{SCC_1}, \cdots, G_{SCC_k}\}) \). When joining the group, member \( u_j \) \( (j\in\{1,2,\cdots,l\}) \) chooses a random prime \( y_j \) \( (y_j<N) \) as its representative and sends it securely to the group CC. After receiving and storing \( y_j \) of all members, the CC computes and stores its own representative \( \omega = \prod_{y_j} \Phi \mod N \), which is product of representatives of all members in \( G_i \). Moreover, each \( SCC_i \) \( (i\in\{1,2,\cdots,k\}) \) should calculate and store its branch representative \( BY_{SCC_i} \), which is product of representatives of all conferees in the subtree rooted at \( SCC_i \). Since all SCCs are leaf nodes at present, we can obtain that \( BY_{SCC_i} = Y_{SCC_i} \) \( (i\in\{1,2,\cdots,k\}) \).

Next, \( SCC_i \) sends \( BY_{SCC_i} \) securely to PCC. PCC stores \( BY_{SCC_i} \), then computes \( SCC_i \)'s branch witness \( B_{\omega_{SCC_i}} = \prod_{y_{ij}} BY_{SCC_i} \mod N \) and sends it back to \( SCC_i \). \( B_{\omega_{SCC_i}} \) is the accumulation of representatives of all members that are not in the subtree rooted at \( SCC_i \). After calculating branch witnesses for all \( SCC_i \), PCC computes and stores \( BY_{PCC} = Y_{PCC} BY_{SCC_1} \cdots BY_{SCC_k} \mod \Phi(N) \), which is product of representatives of all members in the conference.

When \( SCC_i \) receives \( B_{\omega_{SCC_i}} \), it must store and use \( B_{\omega_{SCC_i}} \) to generate the witnesses of all its group members. For member \( u_j, SCC_i \) computes \( o_j = y_j \Phi \mod \Phi(N) \) and \( \omega_j = B_{\omega_{SCC_i}} o_j \mod N \), then sends \( \omega_j \) to \( u_j \). It is obvious that \( \omega_j \) is the accumulation of representatives of all conferees except \( u_j \). PCC computes the witnesses of its group members in a similar way, except that \( \omega_j \) is calculated as \( \omega_j = x^{BY_{PCC}} \mod N \).

Now, \( u_j \) can calculate the conference key \( CK \) as

\[
CK = \omega_j x_j \mod N.
\]

(1)

\( SCC_i \) obtains \( CK \) as

\[
CK = B_{\omega_{SCC_i}} BY_{SCC_i} \mod N.
\]

(2)

\( PCC \) calculates \( CK \) using

\[
CK = x^{BY_{PCC}} \mod N.
\]

(3)

4) Conference Key Update

\( CK \) must be updated when members or CCs join/leave a conference. The update procedure is slightly different between the two cases, which we will discuss separately.

Case 1: Conferee’s join/leave

Suppose of a conference with 11 CCs which form a tree structure as Fig.2. When a new user \( u_{10m} \) joins and sends its representative \( y_{10m}(y_{10m}<N) \) securely to \( SCC_{10} \), \( CK \) will be updated as follows.

Step 1: \( SCC_{10} \) stores \( y_{10m} \), then computes and sends \( \omega_{10m} = B_{\omega_{SCC_{10}}} \) to \( u_{10m} \), where \( r \) is a randomly selected prime number. Next, \( SCC_{10} \) updates \( Y_{SCC_{10}} \) with \( Y_{SCC_{10}} = Y_{SCC_{10}} r y_{10m} \) and \( BY_{SCC_{10}} \) with...
Finally, SCC_{10} encrypts r_{10} with current CK and multicasts it to all its group members (except \( u_{10} \)) and to all its neighbor nodes (i.e. CCs), including parent node and child nodes.

![Fig.2: Example of a CC tree](image)

**Step 2:** When a CC receives \( r_{10} \) from a neighbor node, it will continue to send the datum to its group members and all the other neighbor nodes. This process is executed until all CCs get \( r_{10} \). If a CC receives \( r_{10} \) from its child node, it must update both the child nodes’ and its own branch representatives. For example, when SCC_{2} receives \( r_{10} \) from SCC_{6}, it must recalculate \( r_{10} \) as \( BY_{SCC_{2}} = BY_{SCC_{6}} \cdot r_{10} \). If a CC receives \( r_{10} \) from its parent node, it only needs to update its branch witness. For example, when SCC_{7} receives \( r_{10} \) from SCC_{2}, it updates \( r_{10} \) as \( BY_{SCC_{7}} = BY_{SCC_{2}} \cdot r_{10} \).

**Step 3:** After receiving \( r_{10} \), member \( u_{ij} \) (except \( u_{10} \)) must update its witness as \( \omega = \omega \cdot r_{10} \).

**Step 4:** Depending on its role in the conference, every member/CC calculates the new CK using (1), (2) or (3).

The update process for member’s departure is similar to above, except that in step 1, the group CC must use product of the modular inversion of the leaving member’s representative and a randomly selected prime (e.g. \( r_{10}^{-1} \)) to update corresponding data and current CK.

**Case 2:** CC’s join/leave

Suppose SCC_{10} is a newly joined CC in Fig.2. After computing \( Y_{SCC_{10}} \) and \( BY_{SCC_{10}} \), SCC_{10} sends \( BY_{SCC_{10}} \) securely to its parent node (SCC_{6}) to get the conference key. SCC_{6} stores \( BY_{SCC_{10}} \) and computes \( \omega \cdot r_{10} \) using \( \omega \cdot r_{10} = \omega \cdot \omega = \omega \cdot \omega = \omega \cdot \omega \), where \( r \) is a randomly selected prime number. Then, SCC_{6} sends \( \omega \cdot r_{10} \) to SCC_{10} and updates \( BY_{SCC_{10}} \) with \( BY_{SCC_{10}} = BY_{SCC_{6}} \cdot r_{10} \). Finally, SCC_{10} encrypts \( BY_{SCC_{10}} \) with current CK and sends it to the parent node SCC_{2}, the child node SCC_{9} and all its group members. After receiving \( \omega \cdot r_{10} \), SCC_{10} calculates witnesses for all its group members. The following procedure is the same as above steps 2-4.

When a SCC leaves the conference, CK must be updated using product of the modular inversion of the leaving SCC’s branch representative and a randomly selected prime (e.g. \( r_{10}^{-1} \)).

### 4. Security and performance analysis

This section mainly evaluates the security and performance of our conference key management scheme.

Firstly, we prove correctness of the scheme. According to the quasi-commutative property of one-way accumulators[11] and (1)-(3), we can obtain:

\[
CK = \omega_{i}^{y_{i}} = B_{\omega_{SCC_{i}}}^{BY_{SCC_{i}}} = \omega_{i}^{y_{i}} \cdot \omega_{i}^{y_{i}} = B_{\omega_{SCC_{i}}}^{BY_{SCC_{i}} + \omega_{i}^{y_{i}} \cdot \omega_{i}^{y_{i}}} = B_{\omega_{SCC_{i}}}^{BY_{SCC_{i}} \cdot \omega_{i}^{y_{i}}} = \omega_{i}^{y_{i}} \mod N
\]

Thus, all conference and CCs will have a shared conference key.

Then, we prove the proposed scheme meets the security requirement for forward and backward secrecy. Whenever the conference composition changes, a random prime \( r \) will be selected to update CK. For example, when SCC_{10} joins, SCC_{6} sends \( \omega \cdot r_{10} \) to SCC_{10} instead of \( \omega \cdot r_{10} \). Because \( \omega \cdot r_{10} \) equals current CK, the use of \( r \) is equivalent to changing CK to \( CK' = CK' \). Thus, a newly joined SCC/members will be prevented from accessing past secure keys. Likewise, if SCC_{10} leaves, current CK will be
updated with $CK' = CK^r_{SCC u_j}$. Without knowledge of $r$, the leaving SCC/member is hard to calculate the new $CK$. By this way, both forward and backward secrecy are achieved.

Finally, we examine the complexity of the proposed scheme. In our system, each participant $u_j$ only needs to store $(y_{ij}, \omega_{ij})$ and to perform 1 modular exponentiation to establish $CK$, whilst each CC requires to store representatives of all its group members, branch representatives of all child nodes, its own representative, its branch representative and branch witness. If a conference has $M$ participants and $k$ CCs in the beginning, each CC roughly performs $M/k$ modular exponentiations, $M/k$ modular inversions and $(2M/k-1)$ modular multiplications to establish $CK$. In addition, PCC must perform $k-1$ modular exponentiations and $k$ modular multiplications to calculate representatives of all SCCs. When a member joins a group, the group CC needs to perform 2 modular exponentiations and 4 modular multiplications to update the stored data and $CK$. When a member quits, the group CC performs 1 modular exponentiation, 3 modular multiplications and 1 modular inversion. Other CCs must execute at most 1 modular exponentiation and 2 modular multiplications to respond to such changes. In contrast, conference only need to perform 2 modular exponentiations to update $CK$.

From above analysis, we can see that most of computation and storage tasks are executed by CCs, and that key update cost is quite low for most nodes.

5. Conclusion

In this paper, we introduce a hierarchical architecture for P2PSIP-based conferencing services, which can meet dynamic scalability and security need for large-scale Internet conferences. In this architecture, members of a conference are gathered in groups around some powerful nodes (CCs), which control the conference in a distributed manner. A matched key management scheme based on one-way accumulators is responsible for conference key generation and update. When conferees or CCs change, the shared conference key is changed and the forward/backward secrecy of conference is guaranteed. A preliminary performance analysis shows that the key management scheme has low storage and computation requirements and is suitable for resource-restricted devices. For further study, we are planning to conduct simulation experiments under various network environments (e.g. different network size, various number of CCs) and to deeply evaluate the performance of our scheme.

6. References


492

