2011 International Conference on Computer Science and Information Technology (ICCSIT 2011)
IPCSIT vol. 51 (2012) © (2012) IACSIT Press, Singapore
DOI: 10.7763/IPCSIT.2012.V51.81

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 operat-ions as well as conference control are performed by a number of high-performance peers known as CCs (conference controll-ers). 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 forw-ard/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 modifica-tion, 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, particip-ants 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 perf-ormed 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

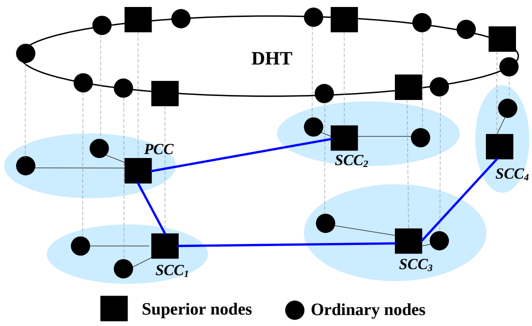

Fig.1:Architecture of the conference system

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 *SCC*s 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 (*SCC*s) as its child nodes. Chosen *SCC*s are preferably located in participants-intensive areas so as to reduce the communicat-ion 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 *SCC*s' URIs to the conference contact list.

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

Once the initialization process described above is comp-leted, 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 confer-ence, 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 (*SCC*s). 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 algori-thm 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 proce-dure 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 successf-ully, 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.

SCC_i	ith secondary CC
G_i	ith group
u_{ij}	jth member of ith group
y_{ij}	representative of u_{ij}
ω_{ij}	witness of u_{ij}

Y_{SCC_i} / Y_{PCC}	representative of SCC_i /*PCC*
BY_{SCC_i} / BY_{PCC}	branch representative of SCC_i /*PCC*
$B\omega_{SCC_i}$	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, ⋯, SCC_k be the *SCC*s 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_{ij} ($j \in \{1,2,\cdots,l\}$) chooses a random prime y_{ij} ($y_{ij}<N$) as its representative and sends it securely to the group CC. After receiving and storing y_{ij} of all members, the CC computes and stores its own representative $Y_i = y_{i1}y_{i2}\cdots y_{il} \bmod \Phi(N)$, 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 *SCC*s 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 $B\omega_{SCC_i} = x^{Y_{PCC}BY_{SCC_1}\dots BY_{SCC_{i-1}}BY_{SCC_{i+1}}\dots BY_{SCC_k}} \bmod N$ 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*s, *PCC* computes and stores $BY_{PCC} = Y_{PCC}BY_{SCC_1}\cdots BY_{SCC_k} \bmod \Phi(N)$, 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_{ij}, SCC_i computes $y_{ij}' = y_{ij}^{-1} \bmod \Phi(N)$ and $\omega_{ij} = B\omega_{SCC_i}^{BY_{SCC_i}y_{ij}'} \bmod N$, then sends ω_{ij} to u_{ij}. It is obvious that ω_{ij} is the accumulation of representatives of all conferees except u_{ij}. *PCC* computes the witnesses of its group members in a similar way, except that ω_{ij} is calculated as $\omega_{ij} = x^{BY_{PCC}y_{ij}'} \bmod N$.

Now, u_{ij} can calculate the conference key *CK* as

$$CK = \omega_{ij}^{\;y_{ij}} \bmod N . \tag{1}$$

SCC_i obtains *CK* as

$$CK = B\omega_{SCC_i}^{\;BY_{SCC_i}} \bmod N . \tag{2}$$

PCC calculates *CK* using

$$CK = x^{BY_{PCC}} \bmod N . \tag{3}$$

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 $\omega_{10m} = B\omega_{SCC_{10}}^{rBY_{SCC_{10}}}$ to u_{10m}, where *r* is a randomly selected prime number. Next, SCC_{10} updates $Y_{SCC_{10}}$ with $Y'_{SCC_{10}} = Y_{SCC_{10}}ry_{10m}$ and $BY_{SCC_{10}}$ with

$BY'_{SCC_{10}} = BY_{SCC_{10}} ry_{10m}$. Finally, SCC_{10} encrypts ry_{10m} with current *CK* and multicasts it to all its group members (except u_{10m}) and to all its neighbor nodes (i.e. CCs), including parent node and child nodes.

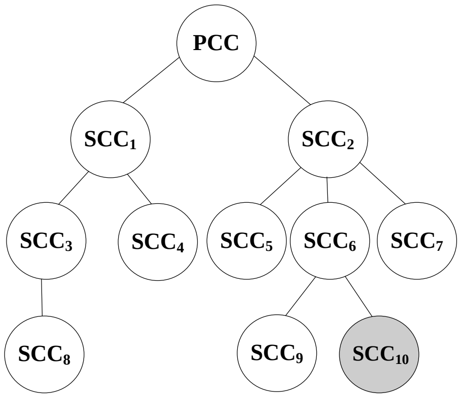

Fig.2: Example of a CC tree

Step 2: When a CC receives ry_{10m} 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 ry_{10m}. If a CC receives ry_{10m} from its child node, it must update both the child nodes' and its own branch representatives. For example, when SCC_2 receives ry_{10m} from SCC_6, it must recalculate BY_{SCC_6} as $BY'_{SCC_6} = BY_{SCC_6} ry_{10m}$ and BY_{SCC_2} as $BY'_{SCC_2} = BY_{SCC_2} ry_{10m}$. If a CC receives ry_{10m} from its parent node, it only needs to update its branch witness. For example, when SCC_7 receives ry_{10m} from SCC_2, it updates $B\omega_{SCC_7}$ with $B\omega'_{SCC_7} = B\omega_{SCC_7}^{ry_{10m}}$.

Step 3: After receiving ry_{10m}, member u_{ij} (except u_{10m}) must update its witness ω_{ij} as $\omega_{ij}' = \omega_{ij}^{ry_{10m}}$.

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 represent-ative and a randomly selected prime (e.g. ry_{10m}^{-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 $B\omega_{SCC_{10}}$ using $B\omega_{SCC_{10}} = B\omega_{SCC_6}^{rBY_{SCC_6}}$, where *r* is a randomly selected prime number. Then, SCC_6 sends $B\omega_{SCC_{10}}$ to SCC_{10} and updates BY_{SCC_6} with $BY'_{SCC_6} = BY_{SCC_6} rBY_{SCC_{10}}$. Finally, SCC_6 encrypts $rBY_{SCC_{10}}$ with current *CK* and sends it to the parent node SCC_2, the child node SCC_9 and all its group members. After receiving $B\omega_{SCC_{10}}$, SCC_{10} calculates witn-esses 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. $rBY_{SCC_{10}}^{-1}$).

4. Security and performance analysis

This section mainly evaluates the security and performa-nce 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:

$$CK = \omega_{ij}^{y_{ij}} = B\omega_{SCC_i}^{BY_{SCC_i} y_{ij}^{-1} y_{ij}} = B\omega_{SCC_i}^{BY_{SCC_i}} = x^{Y_{PCC} BY_{SCC_1} \cdots BY_{SCC_{i-1}} BY_{SCC_{i+1}} \cdots BY_{SCC_k} BY_{SCC_i}}$$
$$= x^{Y_{PCC} BY_{SCC_1} \cdots BY_{SCC_{i-1}} BY_{SCC_i} BY_{SCC_{i=1}} \cdots BY_{SCC_k}} = x^{BY_{PCC}} \pmod N$$

Thus, all conferees 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 $B\omega_{SCC_{10}} = B\omega_{SCC_6}^{rBY_{SCC_6}}$ to it instead of $B\omega_{SCC_{10}} = B\omega_{SCC_6}^{BY_{SCC_6}}$. Because $B\omega_{SCC_6}^{BY_{SCC_6}}$ equals current *CK*, the use of *r* is equivalent to changing *CK* to $CK' = CK^r$. Thus, a newly joined *SCC*/member will be prevented from accessing past secure keys. Likewise, if SCC_{10} leaves, current *CK* will be updated with $CK' = CK^{rBY_{SCC_{10}}^{-1}}$. 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_{ij} only needs to store (y_{ij}, ω_{ij}) and to perform 1 modular exponentiation to establish *CK*, whilst each CC requires to store representat-ives 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 *SCC*s. When a member joins a group, the group CC needs to perform 2 modular exponentiations and 4 modular multipli-cations 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 mod-ular multiplications to respond to such changes. In contrast, conferees 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 dyna-mic 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 plan-ning to conduct simulation experiments under various net-work environments (e.g. different network size, various number of CCs) and to deeply evaluate the performance of our scheme.

6. References

[1] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, SIP: Session Initiation Protocol, RFC3261, IETF, June 2002.

[2] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, “Chord: A scalable peer-to-peer lookup service for internet applications”, Proc. ACM SIGCOMM’01, ACM Press, Oct. 2001, pp.149–160, doi: 10.1145/383059.383071.

[3] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, “A scalable content addressable network”, Proc. ACM SIGCOMM’01, ACM Press, Oct. 2001, pp.161-172, doi: 10.1145/383059.383072.

[4] A. Rowstron and P. Druschel, “Pastry: Scalable, decentralized object location and routing for large-scale Peer-to-Peer systems”, Proc. IFIP/ACM Middleware’01, Springer-Verlag, 2001, pp.329-350.

[5] D.A. Bryan, B.B. Lowekamp, and C. Jennings, “SOSIMPLE: A serverless, standards-based P2PSIP communication system”, Proc. IEEE AAA-IDEA 05, IEEE press, June 2005, pp.42-49, doi: 10.1109/AAA-IDEA.2005.15.

[6] Lichun Li, Yang Ji, Tao Ma, Lanzhi Gu, Chunhong Zhang, “Locality-aware Peer-to-Peer SIP”, Proc. IEEE International Conference on Parallel and Distributed Systems(ICPADS’08), IEEE Computer Society Press, Dec. 2008, pp 295-302, doi:10.1109/ICPADS.2008.92.

[7] G. Camarillo and A. Johnston, Conference Establishment Using Request-Contained Lists in the Session Initiation Protocol (SIP), RFC5366, IETF, Oct. 2008.

[8] Ring J, Choo K-K R, Foo E, and Looi M, “A new authentication mechanism and key agreement protocol for SIP using identity-based cryptography”. Proc. AusCERT Asia Pacific Information Technology Security

Conference(AusCERT2006), University of Queensland Publication, May 2006, pp.57-72, doi:10.1016/j.comcom.2008.01.054.

[9] Shi Yuan-ying, “Efficient authentication and key agreement mechani-sm for P2PSIP”, Application research of computers, vol.28, Jan. 2011, pp.235-237, doi:10.3969/j.issn.1001-3695.2011.01.065.

[10] P. Koskelainen, H. Schulzrinne, and Wu Xiao-tao, “A SIP-based Conference Control Framework”. Proc. NOSSDAV '02, ACM Press, 2002, pp.53 – 61, doi:10.1145/507670.507679.

[11] N. Baric, and B Pfitzmann, “Collision-free accumulators and fail-stop signature schemes without trees”, Proc. EUROCRYPT’97, Springer-Verlag, Feb. 1997, pp.480-494, doi:10.1007/3-540-69053-0_33.