MULTI-PARTY AUTHENTICATION IN DYNAMIC SETTINGS

Abstract

Emerging technologies are introducing new ways of using applications performing transactions, and processing information in global and open networks. Modern computer networks are a complex assembly of databases, web and other application servers, web browsers, and various network devices. In such an environment transactions are usually not simple client-server arrangements, but complex actions involving multiple participants.

In distributed, multi-party environments, there is usually no clear distinction as to who initiated a transaction, which parties should participate in a transaction, which components are used to perform the transaction, and finally which recipient(s) complete the transaction. Many components and aspects of a transaction are often determined dynamically on an ad hoc basis, even after the transaction has been initiated. In such circumstances the reliability of transactions, authentication of participating parties, creation and verification of roles and authorization schemes, non-repudiation, and other security issues are becoming increasingly difficult to establish and enforce. Simple extension of client-server security protocols to multi-party security protocols is not possible. An important security requirement for multi-party transactions is multiparty authentication mechanisms. Most of the existing protocols and mechanisms provide only authentication between two parties and do not provide authentication between multiple participants [Hada 2002]. In addition, they do not provide single sign-on (SSO). Thus new authentication architectures, mechanisms and protocols are required. In such an authentication architecture, once an entity authenticates itself to the local computer system, that system should represent it and perform authentication protocol with other parties until a user logs–out. Not only that, but such authentication architecture should also support authentication sessions with remote parties in different security domains without any pre-registration. Additionally such architecture should be able to support delegated and delayed authentication, which is necessary to conduct secure multi-party transactions. This report gives a survey of security requirements and key management schemes of dynamic peer groups.

Introduction

The increasing use of collaborative applications on the Internet and new Information technologies, is leading to a rapid change from simple client server model towards a multi-party model. Many modern computing environments involve dynamic peer groups. Distributed simulation, multi-user games, conferencing applications and replicated servers are just a few examples. Security is crucial for such distributed and collaborative applications that operate in a dynamic network environment and communicate over insecure networks such as the Internet. Basic security services needed in such a group setting are largely the same as in point-to-point communication: data secrecy and
integrity, and entity authentication. These services cannot be attained without secure, efficient, and robust group key management.

The specific security requirements and needs of dynamic peer groups, in particular key management are still considered as open network challenges. Several group key agreement techniques have been proposed in the last decade, all assuming the existence of an underlying group communication infrastructure for reliable message delivery.

Desired Properties for a Group Key Agreement Protocol

1. Fundamental security goals

   (i) **Key authentication.** Implicit key authentication to entity A implies that only B may be able to compute a particular key, while explicit key authentication to entity A implies that only B has the ability to compute a particular key and has actually done so.

   (ii) **Key confirmation.** Key confirmation to entity A is the assurance that entity B has actually computed the shared session key $K$. A key agreement protocol which provides implicit key authentication to both participating entities is called an authenticated key agreement protocol, while one providing explicit key agreement with key confirmation protocol.

2. Other desirable security attributes

   (i) **Known session key security.** A protocol is called known session key secure if it still achieves its goal in the face of an adversary who has learned some previous session keys.

   (ii) **(Perfect) forward secrecy.** A protocol provides forward secrecy if, when the long-term secrets of one or more entities are compromised, the secrecy of previous session keys is not affected. Perfect forward secrecy refers to the scenario when the private keys of all the participating entities are compromised.

   (iii) **No key-compromise impersonation.** Suppose A’s long-term private key is disclosed. Clearly an adversary that knows this value can impersonate A in any protocol. We say that a protocol resists key-compromise impersonation when this loss does not enable an adversary to impersonate other entities as well.

   (iv) **Unknown key-share attribute.** Entity B cannot be coerced into sharing a key with entity A without B’s knowledge, i.e., when B believes the key is shared with some entity C ≠ A, and A (correctly) believes the key is shared with B.

   (v) **Key control.** Neither entity should be able to force the session key to be a pre-selected value. That is, the session key is determined by all the entities and no one can influence the generation of the session key.
3. **Desirable performance attributes**

(i) Number of message exchanges (passes) required between entities;
(ii) Bandwidth required by messages (total number of bits transmitted);
(iii) Complexity of computation by each entity (as it affects execution time); and
(iv) Possibility of pre-computation to reduce on-line computational complexity.

**Key Agreement in Peer Groups**

Key management has a great impact not only on the security and fault tolerance of the overall system, but also on its performance. Group key management techniques generally fall into three categories

- centralized
- distributed
- contributory.

**Centralized Group key management.**

This involves a single entity or a group of entities which function as a centralized server to distribute keys to group members via a pair-wise secure channel established with each group member. The centralized key server must however be continuously available and present in every possible subset of a group in order to support continued operation. It works well in one-to-many multicast scenarios.

Features of centralized key distribution that makes it unsuitable for Dynamic Peer Groups:

- Centralized key server which acts as a trusted third party for generating and distributing for a multitude of groups is a single point of failure and a likely performance bottleneck.
- Since all group secrets are generated in one place, a TTP presents a very attractive attack target for adversaries.
- Environments with no hierarchy of trust are a poor match for centralized key transport. (For example, consider a peer group composed of members in different, and perhaps competing, organizations or countries.)
- Some DPG environments (such as *ad hoc* wireless networks) are highly dynamic and no group member can be assumed to be present all the time. However, most key distribution protocols assume fixed centers.
- It might be simply unacceptable for a single party to generate the group key. For example, each party may need assurance that the resulting group key is *fresh* and *random* (e.g., in case the key is later used for computing digital signatures).
- Achieving perfect forward secrecy and resistance to known-key attacks in an efficient manner is very difficult in the centralized key distribution setting.
Distributed group key management

This involves dynamically selecting a group member to act as the key distribution server. This approach is more suitable to peer group communication over unreliable networks. Though a robust approach it has the drawback that it requires the key server should maintain long-term pair-wise secure channels with all group members in order to distribute group keys. When a new key server is selected all data structures generated when a key changes also need to be recreated. CKD is a centralized key distribution technique with the key server dynamically chosen among the group members [Amir et al. 2004]. The key server uses pair-wise Diffie–Hellman key exchange to distribute keys. CKD is comparable to GDH in terms of both computation and bandwidth costs.

CKD Protocol

The CKD protocol is a simple group key management scheme. In CKD, the group key is not contributory; it is always generated by the current controller. The controller establishes a separate secure channel with each current group member by using authenticated two-party Diffie–Hellman key exchange. Each such key stays unchanged as long as both parties (controller and regular group member) remain in the group. The controller is always the oldest member of the group. (The oldest member is picked in order to reduce expensive establishment of pair-wise secure channels necessary upon each controller change.)

- Each group member and the controller agree on a unique pairwise key using authenticated two-party Diffie-Hellman. This key does not need to change as long as both users remain in the group. If the controller leaves the group, the new controller has to perform this operation with every member. If a regular member leaves, the controller simply discards this pairwise key.
- The group controller unilaterally generates and distributes the group secret.

CKD Protocol

Let \((x, \alpha^{x_1})\) and \((x_{n+1}, \alpha^{x(n+1)})\) be secret and public keys of \(M_1\) (group controller) and \(M_{n+1}\) respectively.

Let \(K_{1n+1} = \alpha^{x_1 x_{n+1}} \mod p\). \(M_{n+1}\) wants to join the group.

**Round 1:**

\(M_1\) selects random \(r_1 \mod q\).

\(M_1 \rightarrow M_{n+1} : \alpha^{r_1} \mod p\)
**Round 2:**

\[ M_{n+1} \] selects random \( r_{n+1} \mod q \).

\[ M_{n+1} \rightarrow M_i : \alpha^{r_{n+1}} \mod p. \]

**Round 3:**

\[ M_1 \] selects group secret \( K_s \) and computes

\[ M_1 \rightarrow M_i : K_s^{a_{ri}} \mod p \text{ for all } i \text{ belongs to } [2, n+1] \]

Whenever group membership changes, the controller generates a new secret and distributes it to every member encrypted under the long-term pairwise key it shares with that member. In case of a join or merge event (see Figure 3), the controller initially establishes a secure channel with each incoming member. Note that an efficient symmetric cipher can be used to securely distribute the group key. However, the security would differ from that of our group key agreement protocol which relies solely on the Decision Diffie–Hellman assumption [Boneh 1998] and the Discrete Logarithm problem [Menezes et al. 1996]. Therefore, to provide equivalent level of security, we encrypt the group key via modular exponentiation.

---

**Table 1:**

**Step 1:** \( M_1 \) selects random \( r_1 \in \mathbb{Z}_q \) (this selection is performed only once),

\[ M_1 \longrightarrow \{ M_{n+j} \mid j \in [1, k] \} : g^{r_1} \mod p \]

**Step 2:** For each \( j \in [1, k] \), \( M_{n+j} \) selects random \( r_{n+j} \in \mathbb{Z}_q \),

\[ M_1 \leftarrow M_{n+j} : g^{r_{n+j}} \mod p \]

**Step 3:** \( M_1 \) selects a random group secret \( K_s \) and computes

\[ M_1 \rightarrow M_i : K_s^{a_{ri}} \mod p \ \forall i \in [2, n+k]. \]

**Step 4:** From the broadcast message, every member can compute the group key.

---

Fig. 3. CKD—Merge protocol.

When a partition occurs (see Figure 4), in addition to refreshing and distributing the group key, the controller discards the long-term key it shared with each leaving member. A special case is when the controller itself leaves the group. In this case, the oldest remaining member becomes the new controller. Significant additional cost is incurred, since, before distributing the new key, the new controller must first establish a secure channel with every remaining group member.
Contributory Group Key Agreement

Contributory group key agreement protocols compute a group key as a (usually, one-way) function of individual contributions from all members and can provide both key independence and PFS properties. This is appropriate for dynamic peer groups. This approach avoids problems with single point of trust and failure, and also does not always require the establishment of pair-wise secure channels between group members. At the same time, contributory group key agreement presents a tough practical challenge: Its multi-round nature must be reconciled with the possibility of crashes, partitions, and other events affecting group membership that can occur during the execution of the group key agreement.

We give here an overview of four contributory group key management protocols followed by a detailed explanation and comparison.

GDH is a protocol based on group extensions of the two-party Diffie–Hellman key exchange [Steiner et al. 2000] and provides fully contributory key agreement. GDH is fairly computation-intensive, requiring $O(n)$ cryptographic operations for each key change. It is, however, bandwidth efficient

TGDH combines a binary key tree structure with the GDH technique [Kim et al. 2004]. TGDH is efficient in terms of computation as most membership changes require $O(\log n)$ cryptographic operations.

STR [Kim et al. 2004a] is a special case of TGDH with a so-called “skinny” or imbalanced tree. It is based on the protocol by Steer et al. [1990]. STR is more efficient than the above protocols in terms of communication; whereas, its computation costs are comparable to those of GDH and CKD.

BD is another variation of GDH proposed by Burmester and Desmedt [1994]. It is efficient in computation, requiring a constant number of exponentiations (three) upon any membership change. However, communication costs are significant

Centralized and distributed key management rely on symmetric encryption to distribute group keys, while contributory protocols rely on modular exponentiations. Therefore, the former do not provide Perfect Forward Secrecy. However, such protocols scale to large groups and have a lighter overhead than contributory ones.
Group AKE protocols are essential for applications such as secure video or teleconferencing, and also for collaborative (peer-to-peer) applications which are likely to involve a large number of users. A number of solutions to the fundamental problem of authenticated group key exchange (AGKE) among n parties within a larger and insecure public network have been proposed; however, all provably secure solutions thus far are not scalable and, in particular, require O(n) rounds.

The initial contributory group key agreement protocols were obtained by extending Diffie-Hellman exchange method to groups of n-parties[1, 1996]. Drawback was that it did not provide authentication of the parties involved and hence was prone to man-in-the-middle attacks.

Giuseppe Ateniese, Michael Steiner, and Gene Tsudik[2,2000], studied the security services in the context of dynamic peer groups (DPG’s). They leveraged the results of [1] to develop practical and secure authenticated key agreement protocols for DPG’s viz Authenticated Diffie-Hellman Key Exchange protocol for groups A-GDH. Other relevant security features such as key confirmation, key integrity and entity authentication were also considered here.

AKE protocols perform an initial key agreement IKA within a group. Once a group is formed and initial key agreed upon, members may leave or new members may join. Any membership change causes a corresponding group key change. Re-running the full IKA for each membership change is expensive. Auxiliary key agreement (AKA) protocols handle issues like member addition, member deletion etc.

Michael Steiner et al. in their work [3,2000] discuss all group key agreement operations and presents a concrete protocol suite, CLIQUES, which offers complete key agreement services. CLIQUES is based on multi-party extensions of the Diffie-Hellman key exchange method. It is a cryptographic toolkit that supports a menu of key management techniques for dynamic peer groups. Currently, Cliques includes five group key agreement protocols: GDH, CKD, TGDH, STR, and BD. All these protocols provide key independence and PFS.

Oliver Periera, Jean-Jacques in [4,2002] analysed the protocol suites extending Diffie-Hellman key exchange to a group setting and proposed a simple machinery that allowed them to manually pinpoint several unpublished attacks against the main security properties claimed in the definition of these protocols i.e implicit key agreement, perfect forward secrecy and resistance to known-key attacks.

Yair Amir, Y. Kim and Gene Tsudik in [5, 2004] present a thorough performance evaluation of five notable distributed key management techniques (for collaborative peer groups) integrated with a reliable group communication system. An in-depth comparison and analysis of the five techniques is presented based on experimental results obtained in actual local- and wide-area networks. It also gives a taxonomy of application scenarios for secure group communication systems and a mapping between broad application classes and appropriate group key management protocols.
Boneh and Silverberg [6,2002] studied the problem of finding efficiently computable non-degenerate multi-linear maps and presented several applications to cryptography using multi-linear forms. The efficiently computable multi-linear forms would enable one round multi-party key exchange, a unique signature scheme and secure broadcast encryption with very short broadcasts. They proposed a one round multi-party key agreement protocol from multi-linear forms, where security was based on the hardness of multi-linear Diffie Hellman problems and the n-parties shared a session key after one round of broadcast.

The disadvantage of their protocol is that this key is not authenticated and hence is subject to the classic man-in-the-middle attacks.

Lee et.al.[7,2002] presented multi-party authenticated key agreement protocols. Their protocols are also based on multilinear forms. They presented a single protocol with (n+1) different methods for deriving a session key (MAK protocols). Their protocols overcome the disadvantage of lack of authentication in Boneh and Silverberg’s protocol by incorporating authentication into the protocol using certificates. A certification authority (CA) is used in the initial set-up stage to provide the certificates which bind user’s identities to long-term keys. Each entity verifies the authenticity of the certificates he receives. If any check fails, then the protocol is aborted. When no check fails, one of the possible session keys are computed. Consequently, this system requires a large amount of computing time and storage.

Kenneth G. Paterson [8,2004] proved that, contrary to the claim of Lee et al in [7], (n - 2) of the n+1 protocols are easily broken using a standard man-in-the-middle attack. Thus these protocols do not meet even the most basic security requirement for a key agreement protocol. Their paper presents some attacks on the MAK protocols suggested by Lee et al.

GDH Protocol

GDH is a contributory group key agreement protocol which generalizes upon the 2-party Diffie–Hellman key exchange and provides fully contributory key agreement. The shared key is never transmitted over the network (in the clear or otherwise). Instead, a list of partial keys, one for each member, is generated and sent. Upon receipt, each member uses its partial key to compute the group secret. One particular member of the group (controller) is charged with the task of building and distributing this list. The controller is not fixed and has no special security privileges. The protocol runs as follows.

Protocol GDH.2:
Let \( M = \{M_1, \ldots, M_n\} \) be a set of users wishing to share a key \( S_n \). The GDH.2 protocol executes in \( n \) rounds. In the first stage \((n - 1)\) rounds), contributions are collected from individual group members and then, in the second stage \((n\text{-th round})\), the group keying material is broadcast. The actual protocol is as follows:
Initialization:
Let $p$ be a prime and $q$ a prime divisor of $p - 1$. Let $G$ be the unique cyclic subgroup of $\mathbb{Z}_p^*$ of order $q$, and let $\alpha$ be a generator of $G$.

**Round** $i$ ($0 < i < n$):
1. $M_i$ selects $r_i \in \mathbb{Z}_q$.
2. $M_i \rightarrow M_{i+1}$: \{ $a$ \}

**Round** $n$:
1. $M_n$ selects $r_n \in \mathbb{Z}_q$.
2. $M_n \rightarrow$ ALL $M_i$: \{ $r_n$ \}

When a merge event occurs (see Figure 1), the current controller generates a new key token by refreshing its contribution to the group key and passes the token to one of the new members. When the new member receives the token, it adds its own contribution and passes the token to the next new member. Eventually, the token reaches the last new member. This new member, who is slated to become the new controller, broadcasts the token to the group without adding its contribution. Upon receiving the broadcast token, each group member (old and new) factors out its contribution and unicasts the result (called a factor-out token) to the new controller. The new controller collects all the factor-out tokens and adds its own contribution to each. Every thus modified factor-out token (to which the new controller added its contribution) represents a partial key. Once all the partial keys are computed, the list of partial keys is broadcast to the group. Every member obtains the group key by factoring in its contribution into the corresponding partial key in the broadcasted list.

**Merge Protocol**

Assume that $k$ members are added to a group of size $n$.

**Step 1**: $M_n$ generates a new exponent $r'_n \in \mathbb{Z}_q$, computes $g^{r_1 \ldots r_n - r'_n}$ mod $p$, and unicasts the message to $M_{n+1}$.

**Step $j + 1$ for $j \in [1, k - 1]$**: New merging member $M_{n+j}$ generates an exponent $r_{n+j} \in \mathbb{Z}_q$, computes $g^{r_1 \ldots r_n \ldots r_{n+j}}$ mod $p$ and forwards the result to $M_{n+j+1}$.

**Step $k + 1$**: Upon receipt of the accumulated value, $M_{n+k}$ broadcasts it to the entire group.

**Step $k + 2$**: Upon receipt of the broadcast, every member $M_i$, $\forall i \in [1, n + k - 1]$, computes $g^{r_1 \ldots r_n \ldots r_{n+k-1}/r_i}$ mod $p$ and sends it back to $M_{n+k}$.

**Step $k+3$**: After collecting all the responses $M_{n+k}$ generates a new exponent $r_{n+k}$, produces the set $S = \{ g^{r_1 \ldots r_n \ldots r_{n+k}/r_i}$ mod $p | \forall i \in [1, n + k - 1] \}$ and broadcasts it to the group.

**Step $k+4$**: Upon receipt of the broadcast, every member $M_i$, $\forall i \in [1, n + k]$ computes the key $K = (g^{r_1 \ldots r_n \ldots r_{n+k}/r_i})^{r_i}$ mod $p = g^{r_1 \ldots r_n \ldots r_{n+k}}$ mod $p$. 
When some of the members leave the group (Figure 2), the controller (who, at all times, is the most recent remaining group member) removes their corresponding partial keys from the list of partial keys, refreshes each partial key in the list and broadcasts the list to the group. Each remaining member can then compute the shared key. If the current controller leaves, the last remaining member becomes the controller of the group.

TGDH Protocol

TGDH is an adaptation of key trees and computes a group key derived from individual contributions of all group members using a logical binary key tree [Kim et al. 2000, 2004b].

The key tree is organized as follows: each node $(l, v)$ is associated with a key $K(l, v)$ and a corresponding blinded key $B K(l, v) = g^{K(l, v)} \pmod{p}$. The root is associated with the group and each leaf with a distinct member. The root key represents the group key shared by all members, and a leaf key represents the random contribution by of a group member. Each internal node has an associated secret key and a public blinded key. The secret key is the result of a Diffie–Hellman key agreement between the node’s two children. Every member knows all keys on the path from its leaf node to the root as well as all blinded keys of the entire key tree. The protocol relies on the fact that every member can compute a group key if it knows all blinded keys in the key tree. Following every group membership change, each member independently and unambiguously modifies its view of the key tree. Depending on the type of the event, it adds or removes tree nodes related to the event, and invalidates all keys and blinded keys related with the affected nodes (always including the root node). As a result, some nodes may not be able to compute the

---

**Assume that a set $L$ of members is leaving a group of size $n$.**

**Step 1:** The controller $M_d$ generates a new exponent $r'_d \in \mathbb{Z}_q$, produces the set $S = \{ g^{r_1\ldots r_d}/ r_i \pmod{p} | M_i \notin L \}$ and broadcasts it to the remaining group.

**Step 2:** Upon receipt of $S$, every remaining member $M_i$, $\forall i \notin L$ computes the key $K = (g^{r_1\ldots r_d}/ r_i)^{r_i} = g^{r_1\ldots r_d} \pmod{p}$.
root key by themselves. However, the protocol guarantees that at least one member can compute at least one new key corresponding to either an internal node or to the root. Every such member (called a sponsor) computes all keys and blinded keys as far up the tree as possible and then broadcasts its key tree (only blinded keys) to the group. If a sponsor cannot compute the root key, the protocol guarantees the existence of at least one member which can proceed further up the tree, and so on. After at most two rounds (in case of a merge) or \( \log(n) \) rounds (in case of a worst-case partition), the protocol terminates with all members computing the same new group (root) key. After a partition, the protocol operates as follows. First, each remaining member updates its view of the tree by deleting all leaf nodes associated with the partitioned members and (recursively) their respective parent nodes. To prevent reuse of old group keys, one of the remaining members (the shallowest rightmost sponsor) changes its key share. Each sponsor computes all keys and blinded keys as far up the tree as possible and then broadcasts its view of the key tree with the new blinded keys. Upon receiving the broadcast, each member checks whether the message contains a new blinded key. This procedure iterates until all members obtain the new group key.

![Diagram](image)

**Fig. 5. TGDH—Partition operation.**

Figure 5 shows an example where members \( M_1 \) and \( M_4 \) are partitioned out of the group. When a merge happens, the sponsor (the rightmost member of each merging group) broadcasts its tree view to the merging subgroup after refreshing its key share (leaf key) and recomputing all affected blinded keys. Upon receiving this message, all members uniquely and independently determine the merge position of the two trees. As described above, all keys and blinded keys on the path from the merge point to the root are invalidated. The rightmost member of the subtree rooted at the merge point becomes the sponsor. The sponsor computes all keys and blinded keys and broadcasts the key tree (with only blinded keys) to the group. All members now have the complete set of new blinded keys which allows them to compute all keys on their key path. Figure 6 shows an
example of the merge protocol. Members $M_6$ and $M_7$ are added to the group consisting of members $M_1, M_2, \ldots, M_5$.

![Fig. 6. TGDH—Merge operation.](image)

**STR Protocol**

The STR protocol [Steer et al. 1990; Kim et al. 2004a] is an “extreme” version of TGDH with the underlying key tree completely unbalanced or stretched out. In other words, the height of the key tree is always $(n-1)$, as opposed to (roughly) $\log(n)$ in TGDH. All other features of the key tree are the same as in TGDH. After a partition, the sponsor is defined as the member corresponding to the leaf node just below the lowest leaving member. After deleting all leaving nodes (see Figure 7), the sponsor refreshes its key share, computes all (key, blinded key) pairs up to the level just below the root node. Finally, the sponsor
broadcasts the updated key tree thus allowing each member to compute the new group key. STR merge runs in two rounds. In the first round, each sponsor (topmost leaf node in each of the two merging trees) first refreshes its key share and computes the new root key and root blinded key. Then, the sponsors exchange their respective key tree views containing all blinded keys. The topmost leaf of the larger tree becomes the sole sponsor in the second round in the protocol (see Figure 8). Using the blinded keys from the key tree it received in the first round, the sponsor computes every (key, blinded key) pair up to the level just below the root node. It then broadcasts the new key tree to the entire group. All members now have the complete set of blinded keys which allows them to compute the new group key.

BD Protocol

Unlike other protocols discussed thus far, the BD protocol [Burmester and Desmedt 1994] is stateless. Therefore, the same key management protocol is performed regardless of the type of group membership change. Furthermore, BD is completely decentralized and has no sponsors, controllers, or any other members charged with any special duties. The main idea in BD is to distribute the computation among members, such that each member performs only three exponentiations. This is performed in two communication rounds, each consisting of \( n \) broadcasts. Figure 9 depicts the protocol.

Burmester and Desmedt protocol:
Burmester and Desmedt [9] proposed a very efficient protocol that executes in only three rounds:
1. Each member $M_i$ generates its random exponent $r_i$ and broadcasts $Z_i = \alpha^{r_i} \ mod \ p$ to the group.
2. Each member $M_i$ computes and broadcasts $X_i = (Z_{i+1}/Z_{i-1})^{r_i}$;
3. Each member $M_i$ can now compute the key $K_n = Z_{n-1}^r X_{n-1} \cdots X_2 \ mod \ p$.

The group key calculated by each member is then $K_n = \alpha N_1 N_2 + N_2 N_3 + \ldots + N_n N_1$. This protocol requires $n+1$ exponentiations per member and in all but one the exponent is at most $n-1$. The drawback is the requirement of $2n$ broadcast messages.

Assume a group of size $n$.

**Step 1:** Each member $M_i$ selects random $r_i \in \mathbb{Z}_p$, computes $Z_i = g^{r_i} \ mod \ p$ and broadcasts the message to the group.

**Step 2:** Each member $M_i$, after receiving $Z_{i-1}$ and $Z_{i+1}$, computes $X_i = (Z_{i+1}/Z_{i-1})^{r_i} = g^{r_{i+1} r_i - r_{i-1} r_i} \ mod \ p$ and broadcasts it to the group.

**Step 3:** Each member $M_j$, after receiving all $X_i, i \neq j$, computes $K = K_j = (Z_{j-1})^{r_1 + r_2 + \ldots + r_{n-1}} X_{n-1} \cdots X_{j+(n-1)} = g^{r_1 r_2 \cdots r_{n-1}} \ mod \ p$.

Fig. 9. BD protocol.

**Authenticated 2-party Key Agreement**

There exist secure protocols for authenticated DH-based key agreement. However, some are not contributory (such as El Gamal), some require more messages or assume a priori access to certified long-term keys, while others do not offer PFS or are vulnerable to so-called known-key attacks. Steiner et al [ ] developed an extension to the Diffie-Hellman (DH) [ ] key agreement method that provides key authentication

**A-GDH.2 Contributory Key Agreement Protocol**

Michael Steiner and Gene Tsudik [ ] developed an extension to the Diffie-Hellman key agreement method for $n$-parties which provides key authentication. Their protocol is a contributory protocol that provides PFS and is resistant to known key attacks.

**Protocol A-GDH.2:**

**Round n:**

```
.........................
```
The above protocol (A-GDH.2) achieves implicit key authentication in a relatively weak form since the key is not directly authenticated between an arbitrary Mi and Mj (i ≠ j). Instead, all key authentication is performed through Mn. This may suffice in some environments, e.g., when the exact membership of the group is not divulged to the individual Mi's. Another reason may be that Mn is an entity trusted by all other members, e.g., Mn is an authentication server.

A-GDH.2 will result in all participants agreeing on the same key if Mn behaves correctly. However, no one, including Mn, can be sure of other members' participation. In fact, one or more of the intended group members may be "skipped" without detection. Also, a dishonest Mn could partition the group into two without detection by group members. On the one hand, we assume a certain degree of trust in all group members (including Mn), e.g., not to reveal the group key to outsiders. On the other hand, when it comes to group membership, Mn might not be universally trusted to faithfully include all (and only) group members.

**Security Properties of the Protocol**

This protocol provided PFS, Implicit Key Authentication and was resistant to known key attacks.

**Multi-linear forms: Definition**

A map e : G^n → G is an n-multilinear map if it satisfies the following properties:

1. G_1 and G_2 are groups of the same prime order;
2. If a_1, ..., a_n ∈ Z and x_1, ..., x_n ∈ G_1 then e(x_1^{a_1}, ..., x_n^{a_n}) = e(x_1, ..., x_n)^{a_1⋯a_n};
3. The map e is non-degenerate in the following sense: if g ∈ G_1 is a generator of G_1 then e(g, ..., g) is a generator of G_2.

The efficiently computable multilinear forms would enable secure broadcast encryption with very short broadcasts and private keys, a unique signature scheme, and one round multi-party key exchange.

**The multi-linear Diffie-Hellman assumption.**

This assumption says that given
g, g^{a_1}, \ldots, g^{an+1} in G_1, it is hard to compute e(g, \ldots, g)^{a_1 \ldots an+1} in G_2. The multilinear Diffie-Hellman assumption means the multilinear Diffie-Hellman problem is hard.

A one round multi-party key agreement using multilinear forms:

Boneh and Silverberg [ ] proposed a simple one-round multi-party Diffie-Hellman key exchange protocol using multi-linear forms in which the secret session key for n-parties could be created using just one broadcast per entity.

Setup

- Let G_1 and G_2 be two finite cyclic groups of the same prime order p and let g be a generator of G_1.
- Let A_1, \ldots, A_n be n participants who want to share a key.
- Let e_{n-1} : G_1^{n-1} \rightarrow G_2 be an (n-1) multilinear map.

Publish

- Each A_i selects a uniformly random integer a_i \in [1, p-1] and computes g^{a_i}
- Each A_i broadcasts g^{a_i} to all others and keeps a_i secret.

Key Generation:

Each A_i computes conference key KA_i as follows:

\[ KA_i = e_{n-1}(g^{a_1}, g^{a_{i-1}}, g^{a_{i+1}}, \ldots, g^{a_n})^{a_i} = e_{n-1}(g^{a_1}, \ldots, g^{a_n})^{a_1 \ldots an} \in G_2 \]

Hence all participants obtain the same key K = KA_1 = \ldots = Kan

The security of this protocol is based on the hardness of the multilinear Diffie-Hellman problem. More precisely, the session key should be derived by applying a suitable key derivation function to the quantity \( e(g^{a_1}, \ldots, g^{an})^{a_1 \ldots an} \). Otherwise, an attacker might be able to get partial information about session keys even if the MDHP is hard.

This protocol lacked authentication and was hence prone to man-in-the-middle attack.

Man-in-the-Middle Attack on the protocol:
Lee et al (2004) presented authenticated multi-party key agreement protocols with n+1 methods for deriving a session key. Their protocols were also based on multi-linear forms.

A certification authority (CA) is used in the initial set-up stage to provide certificates which bind user’s identities to long-term keys. The certificate for entity Ai will be of the form:

$$Cet_{Ai} = (I_{Ai} || \mu_{Ai} || g || S_{CA}(I_{Ai} || \mu_{Ai} || g))$$

$I_{Ai}$ denotes the identity string of $Ai$, $||$ denotes the concatenation of data items, and $S_{CA}$ denotes the CA’s signature. Entity $Ai$’s long-term public key is $\mu_{Ai} = g^x$, where $x \in Z_p$ is the long-term secret key of $Ai$. Element $g$ is the public value and is induced in order to specify which element is used to construct $\mu_{Ai}$ and the short term public values.

II. Multi-party authenticated key agreement protocols (MAK) ($n > 2$)

Setup:
- Let $G_1$ and $G_2$ be two finite cyclic groups of the same prime order $p$ and let $g$ be a generator of $G_1$.
- Let $A_1, \ldots, A_n$ be $n$ participants who want to share a key.
- Let $e_{n-1} : G_1^{n-1} \rightarrow G_2$ be an (n-1) multilinear map.

Publish:
- Each $A_i$ selects a uniformly random integer $a_i \in [1, p-1]$ and computes $g^{a_i}$.
- Each $A_i$ broadcasts to all other entities the short-term public value $g^{a_i}$ along with a certificate $Cert_{Ai}$ containing his long-term public key and each $A_i$ keeps $a_i$ secret. The ordering of protocol messages is unimportant and any of the other entities can initiate the protocol.

MAK Key generation:
- Each $A_i$ verifies the certificate he receives. If any check fails, the protocol should be aborted. When no check fails, one of the following possible session keys described below should be computed.

1. Type A (MAK-A)

$$KA_i = e_{n-1}(g^{a_1} \ldots g^{a_{i-1}}, g^{a_{i+1}}, \ldots, g^{a_n})^{a_i} \cdot e_{n-1}(g^{x_1} \ldots g^{x_{i-1}}, g^{x_{i+1}}, \ldots g^{x_n})^{x_i}$$

$$= e_{n-1}(g, \ldots, g)^{a_1 \ldots a_{n+1}} \ldots x_1 \ldots x_n$$
2. Type B-j (MAK B-j), (j = 1……n-1)

    KA_i = e_{n-1}(g .... g)^{n}

3. Type C (MAK-C)

Attacks on Lee et al’s protocols.

ID-based authentication: A Different Approach

Another direction of research on key agreement protocols is identity based protocols. In traditional PKI settings, key agreement protocols relies on the parties obtaining each other’s certificates, extracting each other’s public keys, checking certificate chains (which may involve many signature verifications) and finally generating a shared secret. The technique of identity-based encryption (IBE) greatly simplifies this process. Shamir in 1984 [ ] first formulated the concept of Identity-Based Cryptography (IBC) in which a public and private key pair is set up in a special way, i.e., the public key is the identifier (an arbitrary string) of an entity, and the corresponding private key is created by using an identity-based key extraction algorithm, which binds the identifier with a master secret of a trusted authority. In the same paper, Shamir provided the first key extraction algorithm that was based on the RSA problem, and presented an identity-based signature scheme.
By using varieties of the Shamir key extraction algorithm, more identity-based signature schemes and key agreement schemes were proposed. The idea was further streamlined by Sakai and Kasahara [13] in 2003, and is currently an area of very active research. The advantage of ID-based cryptosystems is that it simplifies the key management process which is a heavy burden in PKI based cryptosystems.

**ID-based multi party authenticated key agreement protocols:**

Thus it remains to be an open problem to design efficient secure identity based and authenticated key agreement protocols.

---

**References**

9. "ID-based Multi-party Authenticated Key Agreement Protocols from Multilinear Forms” Hyung Mok Lee, Kyung Ju HaP, and Kyo Min Ku1 J, Zhou et al. (Eds.):


15.