Communication-Efficient Group Key Agreement

Yongdae Kim
Dept. of Information and Computer Science
University of California Irvine
kyongdae@ics.uci.edu

Adrian Perrig
Computer Science Division
University of California Berkeley
perrig@cs.berkeley.edu

Gene Tsudik
Dept. of Information and Computer Science
University of California Irvine
gts@ics.uci.edu

Abstract Traditionally, research in secure group key agreement focuses on minimizing the computational overhead for cryptographic operations, and minimizing the communication overhead and the number of protocol rounds is of secondary concern.

The dramatic increase in computation power that we witnessed during the past years exposed network delay in WANs as the primary culprit for a negative performance impact on key agreement protocols.

The majority of previously proposed protocols optimize the cryptographic overhead of the protocol. However, high WAN delay negatively impacts their efficiency.

The goal of this work is to construct a new protocol that trades off computation with communication efficiency. We resurrect a key agreement protocol previously proposed by Steer et al. We extend it to handle dynamic groups and network failures such as network partitions and merges. The resulting protocol suite is provably secure against passive adversaries and provides key independence, i.e. a passive adversary who knows any proper subset of group keys cannot discover any other group key not included in the subset. Furthermore, the protocol is simple, fault-tolerant, and well-suited for high-delay wide area network.

Keywords: Peer group key agreement, fault-tolerant protocol

1. INTRODUCTION

The proliferation of applications, protocols and services that rely on group communication prompts the need for group-oriented security mechanisms (in addition to the traditional requirements of fault-tolerance, scalability, and reliability). Current group-oriented applications include IP telephony, video conferencing, collaborative workspaces, interactive chats and multi-user games. The security requirements of these applications are fairly typical, e.g., confidentiality, data integrity, authentication and access control. These are achieved through some form of group key management.

The peer nature of many group applications results in certain unique properties and requirements. First, every member in a peer group is both a sender and a receiver. Second, peer groups tend to be small, with fewer than a hundred members. Also, peer groups have no hierarchy and all members enjoy the same status. Therefore, solutions that assign greater importance to some group members are undesirable, since privileged members might behave maliciously; they are also attractive targets of attacks. This essentially rules out the traditional key distribution paradigm as it calls for higher trust in the group member who generates and distributes keys. Finally, since all networks are prone to faults and congestion, any subset of group members must be prepared to function as a group in its own right. In other words, if a network partition splits the members into multiple subgroups, each subgroup must quickly recover and continue to function independently.

In the last two decades a lot of research has been conducted with the aim of minimizing cryptographic overhead in security protocols. It has been long held as an incontrovertible fact that heavy-weight computation – such as large number arithmetic which is the basis of many modern cryptographic algorithms – is the greatest burden imposed by security protocols. We believe that, although this has been the case in the past, rapid advances in computing have resulted in drastic improvements in large-number arithmetic computations. For example, three years ago, a top-of-the-line RISC workstation performed a 512-bit modular exponentiation in around 24 ms. Today, an 850 Mhz Pentium III PC (priced at 1/5-th of the old RISC workstation) performs the same operation in under 1 ms.

In contrast, communication latency has not improved appreciably. Network devices and communication lines have become significantly faster and cheaper. However, the communication (especially via the Internet) has become both accessible and affordable which resulted in drastic increase in the demand for network bandwidth. Consequently, the explosion in the number of users and their devices often causes network congestion and outages. Moreover, while computation power and bandwidth are increasing, network delay is still faced with a fundamental limit dictated by the speed of light.

The bottleneck shift from computation to communication latency leads us to start looking at cryptographic protocols in a different light: allowing more liberal use of cryptographic operations while attempting to reduce the communication overhead. The latter includes both round and message complexity. Communication overhead is especially relevant in a peer group setting since group members can be spread throughout a large network, e.g., the global Internet.

We consider a protocol suggested by Steer et al. in 1988 [SSDW88], one of the first group key agreement protocols. Their protocol is based on the Diffie-Hellman key exchange and assumes the formation of a secure **static group**. We extend their protocol to deal with dynamic groups and network failures. This protocol – referred to as STR hereafter – was neglected due to its heavy computation and communication requirements: $O(n)$ communication rounds and $O(n)$ cryptographic operations are necessary to establish a shared key in a group of n members. However, we extend STR and construct new communication-efficient protocols that support dynamic groups. More concretely, we construct an entire group key management protocol suite, that is particularly efficient in a WAN environment where moderate to high network delays dominate. An extended version of this paper that provides more detail of our algorithms and security is available from the authors.

2. RELIABLE GROUP COMMUNICATION AND GROUP KEY AGREEMENT

In this section, we set the stage for the rest of the paper with a brief overview of the notable features of reliable group communication and group key agreement.

2.1. RELIABLE GROUP COMMUNICATION SEMANTICS

Many modern collaborative and distributed applications require a reliable group communication platform. Current reliable group communication toolkits generally provide one (or both) of two strong group communication semantics: Extended Virtual Synchrony (EVS) [MAMSA94] and View Synchrony (VS) [FLS97]. Both semantics guarantee that: 1) group members see the same set of messages between two sequential group membership events, and, 2) the sender's requested message order (e.g., FIFO, Causal, or Total) is preserved. VS offers a stricter guarantee than EVS: Messages are delivered to all recipients in the same membership as viewed by the sender application when it originally sent the message. In the context of this paper we require the underlying group communication to provide VS. However, we stress that VS is needed for the sake of fault-tolerance and robustness; the security of our protocols is in no way affected by the lack of VS. More details on the interaction of key agreement protocols and reliable group communication are addressed in [AAH+00].

2.2. COMMUNICATION DELAY

Due to the reliable group communication platform, network delay is amplified by the necessary acknowledgments between the group members. The speed of light puts a lower bound on the minimum network delay. For example, a laser pulse that travels through a fiber takes ≈ 10 ms between New York and San Francisco, ≈ 21 ms between Paris and San Francisco, and ≈ 40 ms from London to Sydney. In practice the networks today are slower than the lower bound by about a factor of 4 (due to switching overhead, etc.).

To put this into perspective, an 850MHz Pentium III PC performs a single 512-bit modular exponentiation (one of the most expensive, but most basic public key primitives) in under 1 ms. Moreover, the speed of computers continue to increase. Comparing this with the WAN network delay, it is clear that reducing the number of communication rounds is much more important in the long run for an efficient group key agreement scheme than reducing the computation overhead.

2.3. GROUP KEY AGREEMENT

A comprehensive group key agreement solution must handle adjustments to group secrets subsequent to all membership change operations in the underlying group communication system. The following membership changes are considered:

Join occurs when a prospective member wants to join a group.

Leave occurs when a member wants to leave (or is forced to leave) a group. There might be different reasons for member deletion such as voluntary leave, involuntary disconnect or forced expulsion.

Partition occurs when a group is split into smaller groups. A group partition can take place for several reasons, two of which are fairly common:

- Network failure – this occurs when a network event causes disconnectivity within the group. Consequently, a group is split into fragments.

- Explicit partition – this occurs when the application decides to split the group into multiple components or simply exclude multiple members at once.

Merge occurs when two or more groups merge to form a single group:

- Network fault heal – this occurs when a network event causes previously disconnected network partitions to reconnect.
- Explicit merge – this occurs when the application decides to merge multiple pre-existing groups into a single group.

At first glance, events such as network partitions and fault heals might appear infrequent and dealing with them might seem to be a purely academic exercise. In practice, however, such events are common owing to network misconfigurations and router failures. In addition, in mobile ad hoc (and other wireless) networks, partitions are both common and expected. Moser et al. present compelling arguments in support of these claims [MAMSA94]. Hence, dealing with group partitions and merges is a crucial component of group key agreement.

3. CRYPTOGRAPHIC PROPERTIES

In this section we summarize the desired properties for a secure group key agreement protocol. Following the model of [KPT00], we define six such properties:

- *Weak Backward Secrecy* guarantees that previously used group keys must not be discovered by new group members.
- *Weak Forward Secrecy* guarantees that new keys must remain out of reach of former group members.
- *Group Key Secrecy* guarantees that it is computationally infeasible for a passive adversary to discover any group key.
- *Forward Secrecy (Not to be confused with Perfect Forward Secrecy or PFS)* guarantees that a passive adversary who knows a contiguous subset of old group keys cannot discover subsequent group keys.
- *Backward Secrecy* guarantees that a passive adversary who knows a contiguous subset of group keys cannot discover preceding group keys.
- *Key Independence* guarantees that a passive adversary who knows any proper subset of group keys cannot discover any other group key.

The relationship among the properties is intuitive. The first two (often typically called Forward and Backward Secrecy in the literature) are different from the others in the sense that the adversary is assumed to be a current or a former group member. The other properties additionally include the cases of inadvertently leaked or otherwise compromised group keys. Forward and Backward Secrecy is a stronger condition than Weak Forward and Backward Secrecy. Either of Backward or Forward Secrecy subsumes Group Key Secrecy and Key Independence subsumes the rest. Finally, the combination of Backward and Forward Secrecy yields Key Independence.

In this paper we do not assume key authentication as part of the group key management protocols. All communication channels are public but authentic. The latter means that all messages are digitally signed by the sender using some sufficiently strong public key signature method such as DSA or RSA. All receivers are required to verify signatures on all received messages. Since no other long-term secrets or keys are used, we are not concerned with Perfect Forward Secrecy (PFS) as it is achieved trivially.

4. PROTOCOLS

We now describe the protocols that make up the STR key management suite: join, leave, merge, and partition. All protocols share a common framework with the following features:

- Each group member contributes an equal share to the group key; this share is kept secret by each group member.
- The group key is computed as a function of all current group members' shares.
- As the group grows, new members' shares are factored into the group key while remaining members' shares stay unchanged.
- As the group shrinks, departing members' shares are removed from the new group key and at least one remaining member changes its share.
- All protocol messages are signed by the sender, i.e., we assume an authenticated broadcast channel.

Before describing the protocols in detail, we review the basic STR key agreement protocol and the notation used in the rest of the paper.

4.1. NOTATION

We use the following notation:

n, N	number of protocol parties (group members)
i, j	group member indices: $i, j \in \{1, \ldots, N\}$
M_i	i-th group member; $i \in \{1, \ldots, N\}$
r_i	M_i's session random (secret key of leaf node M_i)
br_i	M_i's blinded session random, i.e. $\alpha^{r_i} \bmod p$
k_j	secret key shared among $M_1 ... M_j$
bk_j	blinded k_j, i.e. $\alpha^{k_j} \bmod p$
p	large prime number
α	exponentiation base

Tree-specific notation

$\mathsf{N}_{\langle j\rangle}$	Tree node j
$\mathsf{IN}_{\langle l\rangle}$	Internal tree node at level l
$\mathsf{LN}_{\langle i\rangle}$	Leaf node associated with member M_i
$T_{\langle i\rangle}$	Tree of member M_i
$BT_{\langle i\rangle}$	Tree of member M_i including all of its blinded keys

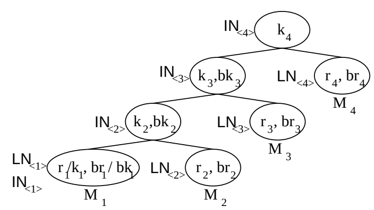

Figure 1 Notation for STR

Figure 1 shows an example of an STR key tree. The tree has two types of nodes: leaf and internal. Each leaf node is associated with a specific group member. An internal node $\mathsf{IN}_{\langle i\rangle}$ always has two children: another (lower) internal node $\mathsf{IN}_{\langle i-1\rangle}$ and a leaf node $\mathsf{LN}_{\langle i+1\rangle}$. The exception is $\mathsf{IN}_{\langle 1\rangle}$ which is also a leaf node corresponding to M_1. (Note that, consequently, $r_1 = k_1$.)

Each leaf node $\mathsf{LN}_{\langle i\rangle}$ has a *session random* r_i chosen and kept secret by M_i. The blinded version thereof is $br_i = \alpha^{r_i} \bmod p$.

Every internal node $\mathsf{IN}_{\langle j\rangle}$ has an associated secret key k_j and a public blinded key $bk_j = \alpha^{k_j} \bmod p$. The secret key k_i $(i > 1)$ is the result of a Diffie-Hellman key agreement between the node's two children. (k_1 is an exception and is equivalent to r_i.) k_i $(i > 1)$ is computed recursively as follows:

$$k_i \quad = \quad (bk_{i-1})^{r_i} \bmod p = (br_i)^{k_{i-1}} \bmod p = \alpha^{r_i k_{i-1}} \bmod p \quad \text{if } i > 1.$$

The group key in Figure 1 is the key associated with the root node:

$$k_4 = \alpha^{r_4 \alpha^{r_3 \alpha^{r_2 r_1}}}$$

We note that the root (group) key is never used directly for the purposes of encryption, authentication or integrity. Instead, such sub-keys are derived from the root key, e.g., by applying a cryptographically secure hash function to the root key. All blinded keys bk_i are assumed to be public.

The basic key agreement protocol is as follows. We assume that all members know the structure of the key tree and their initial position within the tree. (It is simple to have an ordering that uniquely determines the location of each member in a key tree.) Furthermore, each member knows its session random and the blinded session randoms of all other members. The two members M_1 and M_2 can first compute the group key corresponding to $\mathsf{IN}_{\langle 2 \rangle}$. M_1 computes:

$$\begin{array}{lllllll} k_2 & = & (br_2)^{r_1} \bmod p = \alpha^{r_1 r_2} \bmod p, & bk_2 & = & \alpha^{k_2} \bmod p \\ k_3 & = & (br_3)^{k_2} \bmod p, & bk_3 & = & \alpha^{k_3} \bmod p \\ \ldots & & & & & \\ k_N & = & (br_N)^{k_{N-1}} \bmod p & & & \end{array}$$

Next, M_1 broadcasts all blinded keys bk_i with $1 \leq i \leq N-1$. Armed with this message, every member then computes k_N as follows. (As mentioned above, members M_1 and M_2 derive the group key without additional broadcasts.) Any M_i (with $i > 2$) knows its session random r_i and bk_{i-1} from the broadcast message. Hence, it can derive $k_i = {bk_{i-1}}^{r_i} \bmod p$. It can then compute all remaining keys recursively up to the group key from the public blinded session randoms: $k_i = {br_{i+1}}^{k_i} \bmod p$ $(i \leq N)$.

Following every membership change, all members independently update the key tree. Since we assume that the underlying group communication system provides *view synchrony* (see Section 2.1), all members who correctly execute the protocol recompute an identical key tree after any membership event. The following fact describes the minimal requirement for a group member to compute the group key:

Remark 1. *If all members know all blinded session randoms of all other members, there exist at least two members who can compute the group key.*

Proof. This follows directly from the recursive definition of the group key. In other words, both M_1 and M_2 (the member at the lowest leaf nodes) can obtain the group key by computing pairwise keys recursively and using blinded session randoms of other members. □

Remark 2. *Any member can compute the group key, if it knows: 1) its own secret share, 2) the blinded key of its sibling subtree, and, 3) blinded session randoms of members higher in the tree.*

Proof. This also follows from the definition of the group key. To compute the group key, member M_i needs 1) r_i, 2) bk_{i-1}, and 3) $br_{i+1}, br_{i+2}, \ldots, br_N$. □

The protocols described below benefit from a special role (called sponsor) assigned to a certain group member following each membership change. A sponsor reduces communication overhead by performing "housekeeping" tasks that vary depending on the type of membership change. The criteria for selecting a sponsor varies as described below.

4.2. MEMBER JOIN PROTOCOL

We assume the group has n users ($\{M_1, \ldots, M_n\}$), when the group communication system announces the arrival of a new member. Both the new member and the prior group receive this notification simultaneously. The new member M_{n+1} broadcasts a join request message that contains its own blinded key bk_{n+1}. (which is the same as its blinded session random br_{n+1}) At the same time, the current group's sponsor (M_n) computes a blinded version of the current group key (bk_n) and sends the current tree $BT_{\langle n \rangle}$ to M_{n+1} with all blinded keys and blinded session randoms.

Next, each M_i first increments $n = n + 1$ and creates a new root key node $\mathsf{IN}_{\langle n \rangle}$ with two children: the root node $\mathsf{IN}_{\langle n-1 \rangle}$ of the prior tree $T_{\langle i \rangle}$ on the left and the new leaf node $\mathsf{LN}_{\langle n \rangle}$ corresponding to the new member on the right. Note that every member can compute the group key (see Remark 2):

- All existing members only need the new member's blinded session random
- The new member needs the blinded group key of the prior group

In a join operation, the sponsor is always the topmost leaf node, i.e., the most recent member in the current group.

As described, the join protocol takes one communication round and two cryptographic operations to compute the new group key (one before the message exchange and one after.)

The join protocol provides backward secrecy since a new member is only given a blinded key of the existing group. However, the protocol does not provide key independence since knowledge of a group key used before the join can be used to compute the group key used after the join. To remedy the situation, we can modify the protocol to require the sponsor to change its session random and the corresponding blinded value, br_n.

4.3. MEMBER LEAVE PROTOCOL

We again have a group of n members when a member M_d ($d \leq n$) leaves the group. If $d > 1$, the sponsor M_s is the leaf node directly below the leaving member, i.e., M_{d-1}. Otherwise, the sponsor is M_2. Upon hearing about the leave event from the group communication system, each remaining member updates its key tree by deleting the nodes $\mathsf{LN}_{\langle d \rangle}$ corresponding to M_d and its parent node $\mathsf{IN}_{\langle d \rangle}$. The nodes above the leaving node are also renumbered. The former sibling $\mathsf{IN}_{\langle d-1\rangle}$ of M_d is promoted to replace (former) M_d's parent. The sponsor M_s selects a new secret session random, computes all keys (and blinded keys) up to the root, and broadcasts $BT_{\langle s\rangle}$ to the group. This information allows all members to recompute the new group key.

In summary, the leave protocol takes one communication round and involves a single broadcast. The cryptographic cost varies depending on two factors: 1) the position of the departed member, and 2) the position of the remaining member who needs to compute the new key.

The total number of serial cryptographic operations in the leave protocol can be expressed as (assuming n is the original group size):

- $2(n-d)+1+(n-d)+1 = 3n-3d+2$ when $d > 2$
- $3n-7$ when $d = 1, 2$

In the worst case, M_1 or M_2 leave the group. The cost for this leave operation is equal to the leave of member M_3, which is $3n-7$. The average leave cost is $3n/2+2$.

The leave protocol provides forward secrecy since a former member cannot compute the new key owing to the sponsor's changing the session random. The protocol also provides key independence since knowledge of the new key cannot be used to derive the previous keys; this is, again, due to the sponsor refreshing its session random.

4.4. GROUP PARTITION PROTOCOL

A network fault can cause a partition of the group. To the remaining members, this actually appears as a concurrent leave of multiple members. With a minor modification, the leave protocol can handle multiple leaving members in a single round. The only difference is the sponsor selection. In case of a partition, the sponsor is the leaf node directly below the lowest-numbered leaving member. (If M_1 is the lowest-numbered leaving member, the sponsor is the lowest-numbered surviving member.)

After deleting all leaving nodes, the sponsor M_s refreshes its session random (key share), computes keys and blinded keys going up the tree – as in the plain leave protocol – terminating with the computation of $\alpha^{k_{n-1}} \bmod p$. It then broadcasts the updated key tree $BT_{\langle s\rangle}$ containing only blinded values. Each member including M_s can now compute the group key.

The computation and communication complexity of the partition protocol is identical to that of the leave protocol. The same holds for its security properties.

4.5. GROUP MERGE PROTOCOL

We now describe the STR merge protocol for two groups. (A more general protocol for merging larger number of groups is a straightforward extension.) We assume that, as in the case of join, the communication system simultaneously notifies all group members (in both groups) about the merge event. Moreover, reliable group communication toolkits typically include a list of all members that are about to merge in the merge notification. More specifically, we require that each member be able to distinguish between the group it was in from the group that it is merging with. This assumption is not unreasonable, e.g, it is satisfied in SPREAD [AAH^{+}00].

It is natural to merge the smaller group onto the larger one, i.e., to place a smaller tree directly on top of the larger one. If the two trees are of the same size, we can use an unambiguous ordering to decide which group joins which. (For example, compare the identifiers of the respective sponsors.) Consequently, the lowest-numbered leaf of the smaller tree becomes the right child of a new intermediate node. The left child of the new intermediate node is the root of the larger tree. Since the respective trees are known a priori (before the key management starts), all nodes can construct the new key tree before receiving or computing any cryptographic information.

In the first round of the merge protocol, the two sponsors (topmost members of each group) exchange their respective key trees containing all blinded keys. The highest-numbered member of the larger tree becomes the sponsor of the second round in the merge protocol. Using the blinded session randoms of the other group, this sponsor computes every (key, blinded key) pair upto the intermediate node just below the root node. It then broadcasts the key tree with the blinded keys and blinded session randoms to the other members. All members now have the complete set of blinded keys, which allows them to compute the new group key. In any case, the merge protocol runs in two communication rounds.

5. ROBUSTNESS

5.1. PROTOCOL UNIFICATION

Although described separately in the preceding sections, the four STR operations: join, leave, merge and partition, actually represent different expression of a single protocol. We justify this claim with an informal argument below.

Obviously, join and leave are special cases of merge and partition, respectively. It is less clear that merge and partition can be collapsed into a single protocol, because in either case, the key tree changes and the remaining group members lack some number (sometimes none) of blinded keys or blinded session randoms which prevents them from computing the new root key. When a partition occurs, the remaining members reconstruct the tree where some blinded keys are missing. In case of a merge, a shorter tree $\mathcal{A}$ is merged into a taller tree $\mathcal{B}$. Any member in $\mathcal{B}$ now can compute the group key since it knows blinded session random of any member in $\mathcal{A}$. The deepest member in $\mathcal{A}$ also can compute the group key since it knows the blinded session random of any other member in $\mathcal{A}$ and blinded group key of $\mathcal{B}$. Using the broadcast message any member now can compute the new group key.

We established that both partition and merge initially result in a new key tree with a number of missing blinded keys. In case of merge, the missing blinded keys can be distributed in two rounds. This is because a sponsor in both of $\mathcal{A}$ and $\mathcal{B}$ broadcasts its own subtree including all blinded keys. Any member in a given subtree can compute the new root key after receiving both broadcasts. The case of partition is very similar except that the missing blinded keys and the new group key can be distributed in one round.

This apparent similarity between partition and merge allows us to lump the protocols stemming from all membership events into a single, unified protocol. The following figure shows the pseudocode.

```
receive msg (msg type = membership event)
construct new tree
while there are missing blinded keys
  if (I can compute any missing keys and I am the sponsor)
    compute missing blinded keys
    broadcast new blinded keys
  endif
  receive msg (msg type = broadcast)
  update current tree
endwhile
```

The incentive for this is threefold. First, unification allows us to simplify the implementation and minimize its size. Second, the overall security and correctness are easier to demonstrate with a single protocol. Third, we can now claim that (with a slight modification) the STR protocol is self-stabilizing and fault-tolerant as discussed below.

5.2. CASCADED EVENTS

Since network disruptions are random and unpredictable, it is natural to consider the possibility of so-called *cascaded membership events*. In fact, cascaded events and their impact on group protocols are often considered in group communication literature, but, alas, frequently neglected in the security literature. Furthermore, the probability of a cascaded event is much higher on a wide area network. A cascaded event occurs when one membership change occurs while another is being handled. For example, a partition can occur while a prior partition is processed, resulting in a cascade of size two.

We claim that the STR partition protocol is self-stabilizing, i.e., robust against cascaded network events. In general, self-stabilization is a very desirable feature since lack thereof requires extensive and complicated protocol "coating" to either 1) shield the protocol from cascaded events, or 2) harden it sufficiently to make the protocol robust with respect to cascaded events (essentially, by making it re-entrant). The latter is often very complicated and inefficient as seen from [AKNR+01].

The pseudocode for the self-stabilizing protocol is shown as below.

```
receive msg (msg type = membership event)
construct new tree
while there are missing blinded keys
  if (I can compute any missing keys and I am the sponsor)
    compute missing blinded keys
    broadcast new blinded keys
  endif
  receive msg
  if (msg type = broadcast) update current tree
  else (msg type = membership event) construct new tree
endwhile
```

Based on view synchrony discussed in Section 2, we provide an informal proof that the above protocol terminates on any finite number of consecutive cascaded events. Due to view synchrony, every member has the same membership view. We can further assume that the ordering of members in the group communication system is same as that of the key tree. By Remark 1, at least a member, say M_i can compute the group key if all of the blinded session randoms are known. All members can then compute the group key using the broadcast message of the member M_i by Remark 2.

Hence, it is enough to show that at least one member knows every other member's session random, eventually. In the above pseudocode, the sponsor is the node below the lowest node whose blinded session random is missing. Now, if a sponsor M_s cannot compute the group key since some of the blinded keys are missing, it broadcasts the key tree which includes every blinded session random and blinded keys M_s knows. Then the sponsor of the next round will be the one who owns the missing blinded session random. Note that every member will have strictly more blinded session randoms and blinded keys as number of round increases. Hence, as cascaded events stabilize in the group communication system, the STR protocol also terminates.

6. DISCUSSION

6.1. SECURITY

The STR protocol suite and the structure of its group key form a special case of the TGDH key agreement recently presented in [KPT00]. (The latter defines a more general tree-based Diffie-Hellman key agreement.) As such, STR benefits from the provable security of TGDH protocols. Briefly, in [KPT00] it is shown that group key secrecy is reducible to the Decision Diffie-Hellman (DDH) problem [MvOV97].

However, the basic property of group key secrecy is not sufficient for the security of the entire protocol suite. Recall the desired security properties defined in Section 3. We will show that STR offers not only group key secrecy but also weak forward and backward secrecy properties. Furthermore, we show that STR can provide key independence by modifying the protocol slightly.

We now present an informal argument for weak forward and backward secrecy.

The group key secrecy property implies that the group key cannot be derived from the blinded keys alone. At least one secret key K is needed to compute all secret keys from K up to the root key. Hence, we need to show that the joining member M cannot obtain any keys of the previous key tree. First, M picks its secret share r, blinds it and broadcasts α^r as part of its join request. Once M receives all blinded keys on its co-path, it can compute all secret keys on its key path. Clearly, all these keys will contain M's contribution (r); hence, they are independent of previous secret keys on that path. Therefore, M cannot derive any previous keys.

Similarly, we argue that STR provides weak forward secrecy. When a member M leaves the group, the rightmost member of the subtree rooted at the sibling node changes its secret share. Then, M's leaf node is deleted and its parent node is replaced with its sibling node. This operation causes M's contribution to be removed from each key on M's former key path. Hence, M only knows all blinded keys, and the group key secrecy property prevents M from deriving the new group key.

As presented in Section 4, the STR protocols do not provide key independence. This means that an active attacker who somehow acquires a group key used before an additive event (join or merge) can use the knowledge of that key to compute a newer key used after such an event. The same does not hold for subtractive events (leave and partition) since a sponsor always changes its session random following each such event.

The join and merge protocols can be modified slightly to provide key independence as explained in the join and merge protocol: Upon each join or merge event, a sponsor (both sponsors, in case of a merge) changes its session random and recomputes its blinded key before proceeding with the rest of the protocol.

This simple change results in key independence since each membership change is followed by at least one session random change. (Of course, we assume that individual members are honest and do not leak their session randoms to the adversary. This behavior can be regarded as equivalent to revealing the group key.)

6.2. COMPLEXITY ANALYSIS

This section compares the computation and communication of STR protocol to other recent group key agreement methods, Cliques GDH.2 [STW00], Tree-Based Diffie-Hellman (TGDH) [KPT00], and Burmester/Desmedt (BD) [BD94]. These protocols provide contributory group key agreement based on different extensions of the two-party Diffie-Hellman key exchange. Moreover, they all support dynamic membership operations.

We consider the following costs:

- Number of rounds: this affects serial communication delay. Total number of messages: as the number of messages grows, the probability of message loss or corruption is increased, and so is the delay.
- Number of unicasts and broadcasts: a broadcast is much more expensive operation than a unicast, since it requires many acknowledgments within the group communication system.
- Number of serial exponentiation: this is the main factor in the computation overhead.
- Robustness: Lack of robustness requires additional measures to make the secure group communication system robust against cascaded (nested) faults and membership events.

Table 1 shows a comparison of the current approaches for group key management. The bold text refers to a parameter that severely slows down the protocol in a WAN deployment, for which STR is best suited.

In Cliques GDH.2 protocol, the number of new members k is considered, since the merge cost depends on number of new members. The cost for TGDH is the average value when the key tree is fully balanced. The partition or leave cost for STR is computed on average, since it depends on the depth of the lowest-numbered leaving member node. For security reasons [STW00], BD always has to restart anew upon every membership event.

As seen from the table, STR is minimal in communication on every membership event. We showed in Section 5 that robustness in the STR protocol is not only easier to implement than in other protocols, but it also achieves higher robustness to network partitions. Cliques GDH.2 is quite expensive protocol in wide area network, since: 1) it is hard or very expensive to provide robustness against cascaded events [AKNR+01] and 2) communication cost for merge increases linearly as the number of new members does. In TGDH, the partition protocol is expensive (relatively slow) which may cause more cascaded faults and long delays to agree on a key. The cost of BD is mostly acceptable but large number of simultaneous broadcast messages can be problematic over a wide area network.

Table 1 Protocol Comparison

		Rounds	Messages	Ucast	Bcast	Exp.	Robust
Cliques	J	2	2	1	1	$2n$	**Hard**
	L/P	1	1	0	1	n	
	M	$\mathbf{k+3}$	$\mathbf{n+2k+1}$	$\mathbf{n+2k-1}$	2	$n+2k$	
TGDH	J/M	2	3	0	3	$2\log n$	Easy
	L	1	1	0	1	$\log n$	
	P	$\mathbf{O(\log n)}$	$\mathbf{O(\log n)}$	0	$\mathbf{O(\log n)}$	$O(\log n)$	
BD		2	$\mathbf{2n}$	0	$\mathbf{2n}$	3	Easy
STR	J	1	2	1	1	2	Easy
	L/P	1	1	0	1	$\frac{3n}{2}+2$	
	M	2	3	2	1	$3k$	

J: Join, L: Leave, P: Partition, M: Merge, Ucast: Unicast, Bcast: Broadcast, Exp: Exponentiation

References

[AAH+00] Y. Amir, G. Ateniese, D. Hasse, Y. Kim, C. Nita-Rotaru, T. Schlossnagle, J. Schultz, J. Stanton, and G. Tsudik. Secure group communication in asynchronous networks with failures: Integration and experiments. In *ICDCS 2000*, April 2000.

[AKNR+01] Y. Amir, Y. Kim, C. Nita-Rotaru, J. Schultz, J. Stanton, and G. Tsudik. Exploring robustness in group key agreement. In *ICDCS 2001*, 2001.

[BD94] M. Burmester and Y. Desmedt. A Secure and Efficient Conference Key Distribution System. In *EUROCRYPT94*, 1994.

[FLS97] A. Fekete, N. Lynch, and A. Shvartsman. Specifying and using a partionable group communication service. In *ACM PODC '97*, Santa Barbara, CA, August 1997.

[KPT00] Y. Kim, A. Perrig, and G. Tsudik. Simple and fault-tolerant key agreement for dynamic collaborative groups. In *ACM CCS 2000*, November 2000.

[MAMSA94] L. Moser, Y. Amir, P. Melliar-Smith, and D. Agarwal. Extended virtual synchrony. In *ICDCS '94*, June 1994.

[MvOV97] A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1997.

[SSDW88] D. Steer, L. Strawczynski, W. Diffie, and M. Wiener. A secure audio teleconference system. In *CRYPTO '88*, 1988.

[STW00] M. Steiner, G. Tsudik, and M. Waidner. Cliques: A new approach to group key agreement. *IEEE TPDS*, August 2000.