Distributed Collaborative Key Agreement Protocols for Dynamic Peer Groups

Patrick P. C. Lee, John C. S. Lui and David K. Y. Yau

IEEE ICNP 2002
Outline

- Identify the motivations of group key agreement and its requirements.
- Introduce Tree-Based Group Diffie-Hellman (TGDH), which uses a key tree to arrange all the keys.
- Propose three interval-based rekeying algorithms: Rebuild, Batch and Queue-batch.
- Illustrate experimental results.
Motivations

- Many group-oriented and distributed applications require security services.
- Example: a closed and confidential business meeting in a p2p network.
- We therefore need a secure distributed group key agreement scheme so that the group can encrypt their communication data with a common secret group key.
Requirements of Group Key Agreement

- **Distributed**: there is no centralized key server, which has the following limitations:
  - A single point of failure; and
  - Not suitable for peer groups and ad hoc networks.

- **Collaborative**: all group members contribute their own part to generate a group key.

- **Dynamic**: the protocol remains efficient even when the occurrences of join/leave events are very frequent.
Our Work

- We worked on the Tree-Based Group Diffie-Hellman protocol by Kim et al. in ACM CCS’00.

- We designed three interval-based rekeying algorithms that have the distributed, collaborative and dynamic features.

- We performed quantitative and simulation-based analysis to illustrate the performance merits of the interval-based algorithms.
Tree-Based Group Diffie-Hellman Protocol (TGDH)

- A key tree is formed. Each node $v$ represents a secret (private) key $K_v$ and a blinded (public) key $B K_v$.

- $B K_v = \alpha^{K_v} \mod p$, where $\alpha$ and $p$ are public parameters.

- Every member holds the secret keys along the key path, and all the blinded keys in the key tree.

- $K_0$ is the group key.
Tree-Based Group Diffie-Hellman Protocol (TGDH)

- A key tree is formed. Each node $v$ represents a secret (private) key $K_v$ and a blinded (public) key $BK_v$.

- $BK_v = \alpha^{K_v} \mod p$, where $\alpha$ and $p$ are public parameters.

- Every member holds the secret keys along the key path, and all the blinded keys in the key tree.

- $K_0$ is the group key.
TGDH: Relationships between nodes
The secret key of a non-leaf node $v$ can be generated by:
The secret key of a non-leaf node $v$ can be generated by:

\[ 2v+1 \quad \text{and} \quad 2v+2 \]
The secret key of a non-leaf node $v$ can be generated by:

$$K_v = (BK_{2v+1})^{K_{2v+2}} \mod p = (\alpha^{K_{2v+1}})^{K_{2v+2}} \mod p$$
The secret key of a non-leaf node $v$ can be generated by:

$$K_v = (BK_{2v+1})^{K_{2v+2}} \mod p = (\alpha^{K_{2v+1}})^{K_{2v+2}} \mod p$$

$$K_v = (BK_{2v+2})^{K_{2v+1}} \mod p = (\alpha^{K_{2v+2}})^{K_{2v+1}} \mod p$$
TGDH: Relationships between nodes

- The secret key of a non-leaf node \( v \) can be generated by:

\[
K_v = (BK_{2v+1})^{K_{2v+2}} \mod p = (\alpha^{K_{2v+1}})^{K_{2v+2}} \mod p
\]

\[
K_v = (BK_{2v+2})^{K_{2v+1}} \mod p = (\alpha^{K_{2v+2}})^{K_{2v+1}} \mod p
\]

\[
K_v = \alpha^{K_{2v+1}K_{2v+2}} \mod p
\]
A key tree is formed. Each node $v$ represents a secret (private) key $K_v$ and a blinded (public) key $BK_v$.

$BK_v = \alpha^{K_v} \mod p$, where $\alpha$ and $p$ are public parameters.

Every member holds the secret keys along the key path, and all the blinded keys in the key tree.

$K_0$ is the group key.
Tree-Based Group Diffie-Hellman Protocol (TGDH)

- A key tree is formed. Each node $v$ represents a secret (private) key $K_v$ and a blinded (public) key $BK_v$.

- $BK_v = \alpha^{K_v} \mod p$, where $\alpha$ and $p$ are public parameters.

- Every member holds the secret keys along the key path, and all the blinded keys in the key tree.

- $K_0$ is the group key.
TGDH: Handle membership events

- **Rekeying** *(renewing the keys of the nodes)* is performed for every single join/leave event to ensure backward and forward confidentiality.
Rekeying (renewing the keys of the nodes) is performed for every single join/leave event to ensure backward and forward confidentiality.
TGDH: Handle membership events

- **Rekeying** *(renewing the keys of the nodes)* is performed for every single join/leave event to ensure backward and forward confidentiality.

- A special member called **sponsor** is elected to be responsible for broadcasting updated blinded keys.
TGDH: Single Leave Case
TGDH: Single Leave Case

M₅ leaves
TGDH: Single Leave Case

M₅ leaves
TGDH: Single Leave Case

M₅ leaves
TGDH: Single Leave Case

M₅ leaves
TGDH: Single Leave Case

M_5 leaves
M_4 becomes the **sponsor**. It rekeys the secret keys K_2 and K_0 and broadcasts the blinded key BK_2.

M_1, M_2 and M_3 compute K_0 given BK_2.

M_6 and M_7 compute K_2 and then K_0 given BK_5.
TGDH: Single Join Case

Diagram:

0

1

2

3 4

5 6

7 8

M_1 M_2

M_3 M_4

M_6 M_7
TGDH: Single Join Case

M₈ joins
TGDH: Single Join Case

M₈ joins
TGDH: Single Join Case

0

1

3

7

M_1

8

M_2

4

M_3

2

5

5

M_4

6

13

M_6

14

M_7

M_8 joins
TGDH: Single Join Case

M₈ joins

Diagram:

- Node 0 is the root node.
- Node 1 has two children:
  - Node 3: M₁
  - Node 4: M₂
- Node 2 has two children:
  - Node 5: M₃
  - Node 6: M₄(S)
- Node 11 is highlighted with a dashed blue circle.
- Node 13 and Node 14 are child nodes of Node 6.
TGDH: Single Join Case

M₈ joins

Diagram showing a tree structure with nodes labeled 0, 1, 2, 3, 4, 5, 6, 7, 8, 11, 12, 13, 14, and edges labeled M₁, M₂, M₃, M₄(S), M₅, M₆, M₇.
TGDH: Single Join Case

M₈ joins
TGDH: Single Join Case

- $M_8$ broadcasts its individual blinded key $BK_{12}$ on joining.
- $M_4$ becomes the sponsor again. It rekeys $K_5$, $K_2$ and $K_0$ given $BK_{12}$ and broadcasts the blinded key $BK_5$ and $BK_2$.
- Now everyone can compute the new group key.
Interval-Based Rekeying Algorithms

- We can reduce one rekeying operation if we can simply replace $M_5$ by $M_8$ in node 12.

- Interval-based rekeying is proposed such that rekeying is performed on a batch of join and leave requests at regular rekey intervals.

- Interval-based rekeying improves system performance.

- We propose three interval-based rekeying algorithms, namely Rebuild, Batch and Queue-batch.
Rebuild Algorithm

- **Intuition:** Minimize the final tree height so that the number of rekeying operations of every member is reduced.

- **Basic Idea:** Reconstruct the whole key tree to form a complete tree.

- **We can explore under which workload Rebuild is good.**
Rebuild Algorithm

- **Intuition:** Minimize the final tree height so that the number of rekeying operations of every member is reduced.

- **Basic Idea:** Reconstruct the whole key tree to form a complete tree.

- **We can explore under which workload Rebuild is good.**

M₂, M₅, M₇ leave M₈ joins
Rebuild Algorithm

- **Intuition:** Minimize the final tree height so that the number of rekeying operations of every member is reduced.

- **Basic Idea:** Reconstruct the whole key tree to form a complete tree.

- **We can explore under which workload Rebuild is good.**

![Diagram of key tree](image)
Rebuild Algorithm

- **Intuition:** Minimize the final tree height so that the number of rekeying operations of every member is reduced.

- **Basic Idea:** Reconstruct the whole key tree to form a complete tree.

- **We can explore under which workload Rebuild is good.**
Batch Algorithm

- Based on the centralized batch rekeying approach by Li et al. in WWW10 2001.

- **Basic Idea**: add the joins to suitable nodes:
  - Replace the leave nodes with the join nodes.
  - Attach the join nodes to the shallowest positions.
  - Keep the key tree balanced.

- Elect the sponsors who help broadcast new blinded keys.

- Given J joins and L leaves, we illustrate two cases:
  - $L > J > 0$
  - $J > L > 0$
Batch – Example 1: $L > J > 0$
Batch – Example 1: L > J > 0

M_2, M_5, M_7 leave
M_8 joins
Batch - Example 1: L > J > 0

M2, M5, M7 leave
M8 joins
Batch - Example 1: L > J > 0

M_2, M_5, M_7 leave
M_8 joins
Batch - Example 1: $L > J > 0$

$M_2, M_5, M_7$ leave
$M_8$ joins
Batch - Example 1: L > J > 0

M₂, M₅, M₇ leave
M₈ joins
Batch - Example 1: $L > J > 0$

- $M_2, M_5, M_7$ leave
- $M_8$ joins
Batch - Example 1: \( L > J > 0 \)

\[ \text{M}_2, \text{M}_5, \text{M}_7 \text{ leave} \]
\[ \text{M}_8 \text{ joins} \]
Batch - Example 1: L > J > 0

M_2, M_5, M_7 leave
M_8 joins
Batch - Example 1: L > J > 0

- $M_8$ broadcasts its join request, including its blinded key.
- $M_1$ rekeys secret keys $K_1$ and $K_0$. $M_4$ rekeys $K_5$, $K_2$ and $K_0$.
- $M_1$ broadcasts $BK_1$. $M_4$ broadcasts $BK_5$ and $BK_2$. 

$M_2$, $M_5$, $M_7$ leave $M_8$ joins
Batch - Example 2: J > L > 0
Batch - Example 2: J > L > 0

M₈, M₉, M₁₀ join
M₂, M₇ leave
Batch - Example 2: $J > L > O$

$M_8, M_9, M_{10}$ join
$M_2, M_7$ leave
Batch - Example 2: J > L > 0

M₈, M₉, M₁₀ join
M₂, M₇ leave
Batch - Example 2: J > L > 0

M₈, M₉, M₁₀ join
M₂, M₇ leave
Batch - Example 2: J > L > 0

M₈, M₉, M₁₀ join
M₂, M₇ leave
Batch – Example 2: \( J > L > 0 \)

- \( M_8 \) and \( M_9 \) form a subtree \( T_1' \). \( M_{10} \) itself forms a subtree \( T_2' \).
- \( M_8 \) and \( M_9 \) compute \( K_6 \), and one of them broadcasts \( BK_6 \).
- \( M_1 \) rekeys \( K_3 \) and \( K_1 \). \( M_6 \) rekeys \( K_2 \).
- \( M_1 \) broadcasts \( BK_3 \) and \( BK_1 \). \( M_6 \) broadcasts \( BK_2 \).
Queue-batch Algorithm

- **Intuition:**
  - The previous approaches perform rekeying at the start of every rekey interval, leading to a heavy processing workload at the update instance.
  - Reduce the load by pre-processing the join events during the idle rekey interval.
Queue-batch Algorithm

- Two stages: Queue-subtree and Queue-merge.

  **Queue-subtree**: Within the idle rekey interval, form a subtree $T'$ with all joining members, just like individual rekeying for a single join event.

  **Queue-merge**: At the beginning of the next rekey interval, prune all departed leaf nodes if any and add the subtree $T'$ to the highest leave position (or attach $T'$ to the shallowest position).

- Elect the sponsors who can help broadcast the new blinded keys.
Queue-batch – Example of Queue-merge Phase
Queue-batch -
Example of Queue-merge Phase

M₈, M₉, M₁₀ join
M₂, M₇ leave
Queue-batch – Example of Queue-merge Phase

\[ 0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \]

- \( M_8, M_9, M_{10} \) join
- \( M_2, M_7 \) leave
Queue-batch -
Example of Queue-merge Phase

M_8, M_9, M_{10} join
M_2, M_7 leave
Queue-batch - Example of Queue-merge Phase

M₈, M₉, M₁₀ join
M₂, M₇ leave
Example of Queue-merge Phase

- M₈, M₉, M₁₀ join
- M₂, M₇ leave

Diagram:
- Node 0
  - Node 1
    - Node 3
      - M₁(S)
    - Node 4
      - M₃
  - Node 2
    - Node 5
      - Node 11
        - Node 23
          - M₄
        - Node 24
          - M₅
      - Node 12
        - Node 27
          - M₆
        - Node 28
          - M₈
    - Node 6
      - M₉
      - T'
Queue-batch - Example of Queue-merge Phase

M₈, M₉, M₁₀ join
M₂, M₇ leave
Queue-batch – Example of Queue-merge Phase

- $T'$ is attached to node 6.
- $M_{10}$, the sponsor, will broadcast $BK_6$.
- $M_1$ rekeys $K_1$. $M_6$ rekeys $K_2$.
- $M_1$ broadcasts $BK_1$. $M_6$ broadcasts $BK_2$. 

$M_8$, $M_9$, $M_{10}$ join
$M_2$, $M_7$ leave
Performance Evaluations

- Study the performance of the interval-based algorithms.

**Performance Metrics:**

- **Number of renewed nodes:** a *renewed* node refers to a non-leaf node whose keys are renewed. This metric provides a measure of the communication cost.

- **Number of exponentiation operations:** this metric provides a measure of the computation load.

**Settings:**

- There is only one group.
- The population size is fixed at 1024 users.
- Originally, 512 members are in the group.
- Every potential member joins the group with probability $p_j$, and every existing member leaves the group with probability $p_l$. 
Experiment 1: Evaluation using Mathematical Models

- Start with a well-balanced tree with 512 members.
- Obtain the metrics under different numbers of joins and leaves.
- Queue-batch offers the best performance, and a significant computation/communication reduction when the group is very dynamic.
- Details on mathematical models are referred to the paper/technical report.
Experiment 2: Average Analysis

- Average number of exponentiations at different join probabilities:

\[
p_J = 0.2\]

\[
p_J = 0.5\]

\[
p_J = 0.7\]

![Graphs showing average number of exponentiations for different join probabilities](image)
Experiment 2: Average Analysis

- Average number of renewed nodes at different join probabilities:

  \[ p_J = 0.2 \]

  \[ p_J = 0.5 \]

  \[ p_J = 0.7 \]
Experiment 3: Instantaneous Analysis

- Instantaneous number of exponentiations at different join probabilities for Batch and Queue-batch:

\[
\begin{align*}
&\text{Batch} \\
&\text{p}_J = 0.25 \\
&\text{p}_L = 0.25
\end{align*}
\]

\[
\begin{align*}
&\text{Queue-batch} \\
&\text{p}_J = 0.25 \\
&\text{p}_L = 0.75
\end{align*}
\]

\[
\begin{align*}
&\text{Batch} \\
&\text{p}_J = 0.75 \\
&\text{p}_L = 0.25
\end{align*}
\]

\[
\begin{align*}
&\text{Queue-batch} \\
&\text{p}_J = 0.75 \\
&\text{p}_L = 0.75
\end{align*}
\]
Experiment 3: Instantaneous Analysis

- Instantaneous number of renewed nodes at different join probabilities for Batch and Queue-batch:

  - $p_J = 0.25$
  - $p_L = 0.25$
  - $p_J = 0.75$
  - $p_L = 0.25$
  - $p_J = 0.75$
  - $p_L = 0.75$
Experimental Results

- **Queue-batch** offers the **best performance** in terms of computation and communication costs among the three interval-based algorithms.

- The superior performance of Queue-batch is more obvious when the occurrences of joins/leaves are highly frequent.
Future Work

- Authentication
- Sponsors’ coordination
- Fault tolerance
- System implementation