A Survey on Decentralized Group Key Management Schemes

Presented by:
Anurodh Joshi
Paper by Ling Cheung et. al presented a Decentralized group key management scheme which uses Cipher Attribute based encryption (CP-ABE)

Critique
- The Group Controller (GC) is actively involved in management of the group (Join and Leave) and becomes a bottleneck for large groups. Are there schemes that reduce the load on GC?
- How do the other schemes handle member(s) joining and leaving the group (rekeying)?
- *In this paper, if the SGC leaves the group, all members lose their membership and have to rejoin. Do other schemes handle this problem in a more elegant way?*

Goal:
- To answer above questions
- Get a broad idea of Decentralized Group Key Management Scheme
- Provide a starting point for further exploration of Group Key Management
- Overview of Group Key management schemes (why Decentralized Scheme?)
- Taxonomy of Decentralized Group Key Management Scheme
  - Membership-driven Rekeying
  - Time-driven Rekeying
- Membership Driven Rekeying Vs Time Driven Rekeying
- Group Key Sharing
- Concluding Remarks
Group Key Management requirements (Challal et. Al [1])

- Security Requirements
  1) Forward Secrecy
  2) Backward Secrecy
  3) Collusion Freedom
  4) Key Independence
  5) Minimal Trust

- QoS Requirements
  1) Low Bandwidth
  2) No 1-affects-n
  3) Minimal delays
  4) Service Availability

- Key Server Requirements
  1) Low Storage
  2) Low Computation

- Group Members Requirements
  1) Low Storage
  2) Low Computation
Group Key Management Schemes

Taxonomy of Group Key Management Protocols

- Centralized Protocols
  - A single entity Group Controller (GC) is responsible for managing Data Encryption key (DEK) and the group
  - Approach such as Logical Key Hierarchy (LKH) reduces the overhead of rekeying
  - Has a Single point of failure -> GC
Group Key Management Schemes

- **Distributed Scheme**
  - Characterized by having no group controller; members generate group key in a contributory manner
  - Solves the problem of single point of failure
  - Reduces bottlenecks in the network in comparison to centralized schemes
  - Problem:
    1. Generating keys entails that each member be aware of the list of current members (Trust issues)
    2. Each member does not have the computational resources to generate keys
  - Overall, hard to manage the group
Group Key Management Schemes

An example of Decentralized Group Key Management Architecture

- Decentralized Scheme
  - A hierarchy of key managers share the labor of distributing the DEK and managing the group
  - Avoids bottlenecks as well as single point of failure*
  - Each subgroup (centralized node) is managed using a centralized scheme
  - Uses the advantages of both centralized and distributed scheme
Taxonomy

- **Membership-Driven Re-keying**
  - The DEK is changed each time a join or a leave occurs

- **Time-Driven Re-keying**
  - Periodic refresh

- Discuss the protocols based on:
  - Involvement of Group Controller and Sub-group controller in managing the group
  - Rekeying
  - Sharing of the DEK
  - Forward and Backward secrecy, Data transmission (security)
Membership-Driven Re-keying Protocols

- XU Yanyan et al. [2] (core tree based approach)
- Core tree
  - GC and SGC
  - DEK is shared among core members
  - SGC $K_{\text{IND}}$
- Subgroups
  - Subgroup Key
  - GMs $K_{\text{u}}$.
- **DEK is not shared with GMs.**
- Join:
  - The Subgroup Key is refreshed
  - For existing members, the new Subgroup key is encrypted with $K_{\text{u}}$
  - For new member, $K_{\text{u}}$ provided through secure unicast and Subgroup is encrypted with $K_{\text{u}}$
- Data Transmission:
  - Within a subgroup, use Subgroup Key to encrypt data.
  - Between subgroups, first DEK is used to transmit the message to the core tree. Other SGCs can encrypt it using Subgroup key and transmit it to GMs.
- Leave:
  - Local
  - Subgroup Key is refreshed
- SGC leave: the GC updates the DEK and sends it to all SGCs by encrypting it with $K_{\text{IND}}$. The subgroup is then merged to the adjacent core.
Xingfeng Guo et al. [3] propose a **simple** RSA key based (public, private keys) decentralized key management scheme.

The DEK is shared with everyone within the group.

The GC shares the DEK with SGC by encrypting the DEK with SGCs RSA public key.

**Join**
- Member contacts the RSA server and sends its public key to the RSA server.
- The SGC re-purchases the public keys updates its child nodes list.
- The GC will generate a new DEK and distribute it to SGC. SGC distributes it to GMs (using GMs public key).

**Data Transmission**: Data is encrypted using RSA public key of the GM within a subgroup.

**Leave**
- SGC updates child nodes list.
- DEK is updated and distributed.

**Periodic DEK refresh for added security**
Hydra:

- Subgroups managed by Hydra Servers
- **GC is only involved during group creation phase and not in key management**
  - The GC distributes the group policies, and certificates to the HSs.
- **The group key is shared among the whole group in this scheme.**
- Hydra key: is used to encrypt messages between HSs during communication
- Subgroup key: protects the group key within a subgroup. Each subgroup has a unique subgroup key.
- Key Distribution: The HSs agree on a common group key and distribute the group key to subgroup members
- **Backward and Forward Secrecy:**
  - Each new group key is completely independent of the previous ones
  - Since the group key is common, the group key has to be changed if membership changes
  - The HS of the subgroup whose membership changes generates a new key and sends it to other HSs. The HSs will then distribute this to their group members
- **Data Transmission**
  - Hydra Key is used to encrypt messages during communication between HSs
  - A subgroup key is used to encrypt group key in the subgroup. When a new group key is distributed among HSs, then each HS will encrypt the group key with subgroup key and distribute it to the subgroup members.
Time-Driven Re-keying Protocols
Peravyian et. al. [4]

- No central GC.
- Multiple subgroups managed by Group Managers
- A random secret key K (session based) is generated and distributed to group members in all groups by encrypting K with the public key of group members
- Working Key (DEK): Generated based on a random value ‘n’, and secret key, K. The encrypted message contains ‘n’

Backward and Forward Secrecy:

- Each new secret Key, and ‘n’ is completely independent of the previous ones
- If a new member joins, he is given the current group key.
- Even if a member leaves, the group key is kept intact till next session
- Collusion: Each session has independent keys

The group key is shared among the whole group in this scheme.
Kronos:

- Works with existing framework such as Intra domain group key management protocols (IGKMP)
- **Targeted for handling large groups spread across the internet for example pay per view multicast groups.**
- Decouples membership size change from rekeying

![Diagram of Kronos](attachment:image.png)
- The DEK is shared across the group.
- Join: the **Area Key Distributor (AKD)** establishes a private key for the new member and sends him the DEK by encrypting it with the private key.
- The joins and leaves are processed in batches during rekey events.
- Periodic key refresh: All AKDs have their clocks synchronized. Hence, at the refresh time, each AKD independently refreshes the group key by using the same encryption algorithm and the old group key i.e.

\[ \text{DEK}_{i+1} = E_K(\text{DEK}_i) \]

where \( i \geq 0 \), where \( E \) is the encryption algorithm \( K \) is the secret key distributed by the **Domain Key Distributor (DKD)**.

- Backward and Forward Secrecy:
  - The DKD can refresh \( K \), and DEK.
  - Periodic refresh

- Disadvantage: The new keys depend on the old keys. Hence, it may not be too hard to hack a new key if one knows the old key. Refreshing the \( K \) and \( R \) does help to some extent.
Membership Driven Rekeying Vs Time Driven Rekeying

- Depends on the need of the group as well as its scale
  - Time Driven Rekeying:
    • Decouples the rekeying from size and dynamics of the group -> scalable to large groups
    • Is applicable to multicasts groups such as pay per view, sports events and so on
    • For large groups spread across the internet this approach reduces the overhead of constant rekeying due to membership change
    • Processes membership requests (joins & leaves) in batches, so membership takes time to come into effect

- Membership Driven Rekeying
  • For groups where backward and forward secrecy is critical, for example -> intelligence data
  • Applicable for small to medium groups
If the group key shared across the group, i.e. if each member in every subgroup knows the DEK, it usually entails refreshing the DEK with change in membership.

Can be avoided by creating two sets of encryption keys:
- DEK -> shared between GC and SGC only
- Subgroup key is shared between SGC and Group Members
- Problem?
However, this means that to share messages between subgroups, the SGC will have to convert the messages and this can prove to be a bottleneck.
Decentralized approach makes it easy to manage group keys
- Avoids single point of failure
- Has a hierarchy which makes it easy to manage the group

Membership-driven rekeying is apt for small to medium size groups

Time-driven rekeying is suitable for very large groups as it decouples rekeying from size and dynamics of the group

* When the SGC leaves?
- Most papers do not focus on SGC leave (trusted few -> rare)
- Xu Yanyan et. Al -> If a member in the core tree (subgroup controller) leaves, the Group controller updates the DEK in the core tree. The GC then encrypts it with each core nodes' individual key $K_{\text{IND}}$ and distributes it. The subgroup whose SGC left is merged to an adjacent core.
- Hydra: If a Hydra Server leaves the group, then the subgroup members in that subgroup will join another HS by using the Expand Searching Ring Protocol (ERSP) to find the closest HS


Questions?