Tree-based Group Key Agreement

Yongdae Kim, Adrian Perrig, Gene Tsudik, in ACM Transactions on Information and System Security, 2004

Naishil Shah
CMPE 253 – Network Security
nanshah@ucsc.edu

Order of Content

- Introduction
- Assumptions and Requirements
- Cryptographic Properties
- Notations
- TGDH Protocols
- Practical Aspects (Issues and solutions)
- Performance Analysis
- Related Work
- Conclusion

Introduction

- Focus on Group Key Management in Dynamic Peer Groups (DPGs).
- 3 approaches in Group Key Distribution:
  1. Centralized Group Key Distribution
  2. Decentralized Group Key Distribution
  3. Contributory Key Management (Focus)
     a) Unify 2 trends – *Key trees and Diffie-Hellman key exchange.*
     b) *Tree-Based Group Diffie-Hellman (TGDH) is born.*

Order of Content

- Introduction
- ***Assumptions and Requirements***
- Cryptographic Properties
- Notations
- TGDH Protocols
- Practical Aspects (Issues and solutions)
- Performance Analysis
- Related Work
- Conclusion

Assumptions & Requirements

1. Semantics and Support
   a) Two commonly used group communication semantics – Extended Virtual Synchrony (EVS) and View Synchrony (VS)
   b) Both guarantee that:
      i. Members see the same set of messages between 2 sequential events
      ii. The sender's requested message order (e.g. FIFO) is preserved.
   c) What makes them different?
   d) What is preferred in the context of this paper?
      i. VS service is required to be provided by the underlying group communication.
      ii. Reason – Membership events are unpredictable, can overlap in time and cause instability if significant amount of state is kept.
      iii. Only used for sake of fault-tolerance and robustness.

Assumptions & Requirements….

2. Group Membership Events
   a. Categorized in various types of events:
      i. Single or Multiple
      ii. Additive or Subtractive
      iii. Voluntary or Involuntary
   b. Single and Multiple events include – ***join*** or ***leave***
   c. Additive or Subtractive events include – ***group merge*** or ***group partition***
   d. Examples – Network Failure, Explicit Partition, Network Fault heal, Explicit merge
   e. Most important security feature – Key Freshness

Order of Content

- Introduction
- Assumptions and Requirements
- ***Cryptographic Properties***
- Notations
- TGDH Protocols
- Practical Aspects (Issues and solutions)
- Performance Analysis
- Related Work
- Conclusion

Cryptographic Properties

- To define the 4 important security properties of group key management, we proceed with the following assumption:

Assume group key is changed m times and sequence of successive group keys is

$$K = \{K_0, \ldots., K_m\}$$

1. *Group Key Secrecy*
2. *Forward Secrecy*
3. *Backward Secrecy*
4. *Key Independence*

- We do not assume key authentication here. All communication channels are public but authentic.

Order of Content

- Introduction
- Assumptions and Requirements
- Cryptographic Properties
- ***Notations***
- TGDH Protocols
- Practical Aspects (Issues and solutions)
- Performance Analysis
- Related Work
- Conclusion

Notations

We use the following notation:

N	number of protocol parties (group members)
$\mathcal{C}$	set of current group members
$\mathcal{L}$	set of leaving members
$\mathcal{J}$	set of newly joining members
M_i	i-th group member; $i \in \{1, \ldots, N\}$
h	height of a tree
$\langle l, v \rangle$	v-th node at level l in a tree
T_i	M_i's view of the key tree
$\widehat{T}_i$	M_i's modified tree after membership operation
$T_{\langle i,j \rangle}$	A subtree rooted at node $\langle i, j \rangle$
BK_i^*	set of M_i's blinded keys
p, q	prime integers
α	exponentiation base

- Use of binary trees
- Key-path
- Co-path
- Key of node present at <l,v> - $K_{<l,v>}$
- Blind key - $BK_{<l,v>} = f(K_{<l,v>})$
  - Where $f(k) = a^K \bmod p$

Notations…

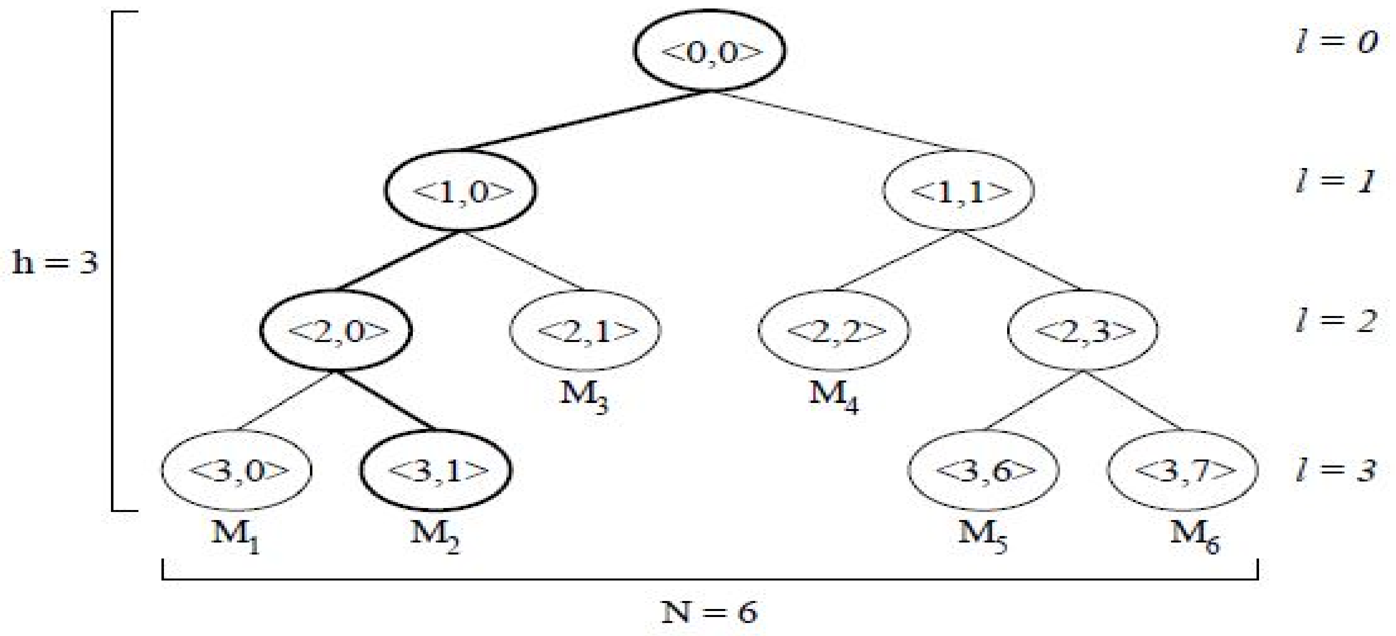

Fig. 1. Notation of a key tree

For example, in Figure 1, M_2 can compute $K_{\langle 2,0\rangle}$, $K_{\langle 1,0\rangle}$ and $K_{\langle 0,0\rangle}$ using $BK_{\langle 3,0\rangle}$, $BK_{\langle 2,1\rangle}$, $BK_{\langle 1,1\rangle}$, and $K_{\langle 3,1\rangle}$. The final group key $K_{\langle 0,0\rangle}$ is:

$$K_{\langle 0,0\rangle} = \alpha^{\left(\alpha^{r_3\left(\alpha^{r_1 r_2}\right)}\right)\left(\alpha^{r_4\left(\alpha^{r_5 r_6}\right)}\right)}.$$

Order of Content

- Introduction
- Assumptions and Requirements
- Cryptographic Properties
- Notations
- ***TGDH Protocols***
- Practical Aspects (Issues and solutions)
- Performance Analysis
- Related Work
- Conclusion

TGDH Protocols

- Four basic protocol form the suite
- Common framework:
  1. Equal share
  2. Secret shares
  3. Membership changes taken into account
  4. RSA for message signing
- Minimum Requirement :
  A group key can be computed from any member's secret share (i.e. any leaf value) and all the blind keys on the co-path to the root.
- Knowledge of all Bkeys

TGDH Protocols….

- *SPONSOR :*
  1. Role
  2. Additive Change
  3. Subtractive Change
- Assumption – Every member can unambiguously determine both the sponsors and the insertion location in the key tree (additive event)

Join Protocol

Step 1: The new member broadcasts request for join.

$$M_{n+1} \xrightarrow{\quad BK_{\langle 0,0\rangle}=\alpha^{r_{n+1}} \quad} \mathcal{C} = \{M_1, \ldots, M_n\}$$

Step 2: Every member

- update key tree by adding new member node and new intermediate node,
- removes all keys and bkeys from the leaf node related to the sponsor to the root node.

The sponsor M_s additionally

- generates new share and computes all $[key, bkey]$ pairs on the key-path,
- broadcasts updated tree $\widehat{T}_s$ including only bkeys.

$$\mathcal{C} \cup \{M_{n+1}\} = \{M_1, \ldots, M_{n+1}\} \xleftarrow{\quad \widehat{T}_s(BK_s^*) \quad} M_s$$

Step 3: Every member computes the group key using $\widehat{T}_s$.

Fig. 2. Join Protocol

Join Protocol....

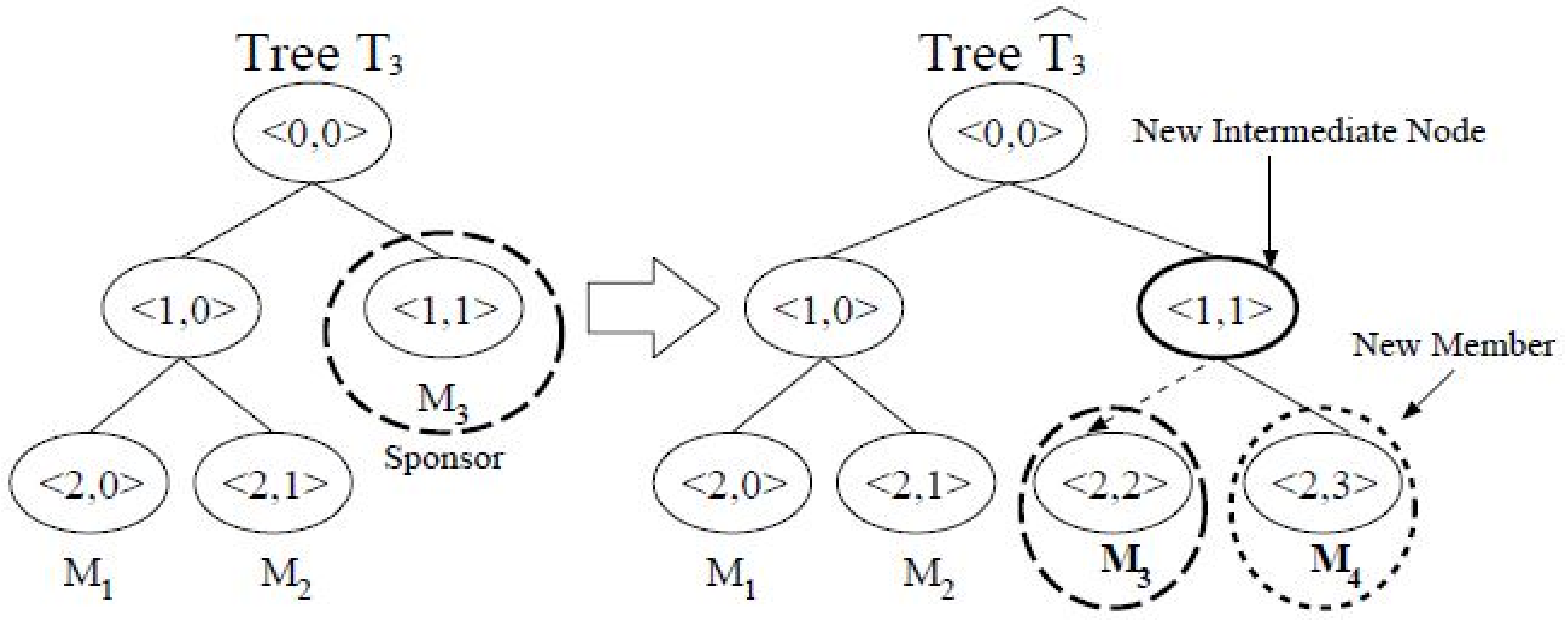

Fig. 3. Tree update: join

Leave Protocol

Step 1: Every member

- updates key tree by by removing the leaving member node and relevant parent node,
- removes all keys and bkeys from the leaf node related to the sponsor to the root node.

Sponsor M_i additionally

- generates new share and computes all $[key, bkey]$ pairs on the key-path,
- broadcasts updated tree $\widehat{T}_s$ including only bkeys.

$$M_s \xrightarrow{\widehat{T}_s(BK_s^*)} \{M_1..M_n\} - \{M_d\}$$

Step 2: Every member computes the group key using $\widehat{T}_s$.

Fig. 4. Leave Protocol

Leave Protocol…

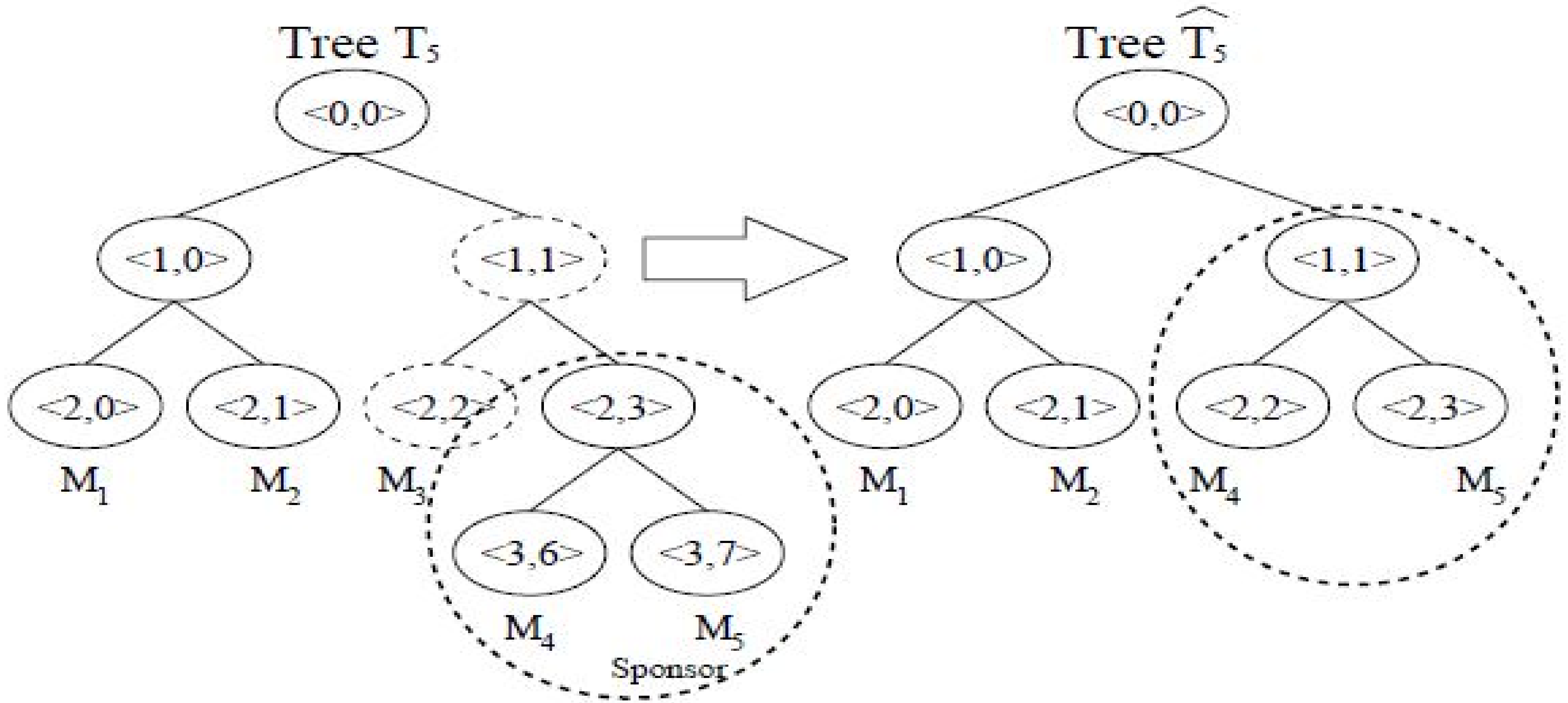

Fig. 5. Tree updating in leave operation

Partition Protocol

Step 1: Every member

- updates key tree by by removing all the leaving member nodes and their parent node,
- removes all keys and bkeys from the leaf node related to the sponsor to the root node.

– Each sponsor M_{s_t}

- If M_{s_t} is the shallowest rightmost sponsor, generates new share,
- computes all $[key, bkey]$ pairs on the key-path until it can proceed,
- broadcasts updated tree $\widehat{T}_{s_t}$ including only bkeys.

$$M_{s_t} \xrightarrow{\widehat{T}_{s_t}(BK^*_{s_t})} \mathcal{C} - \mathcal{L}$$

Step 2 to h (Until a sponsor M_{s_j} computes the group key)

– Each sponsor M_{s_t}

- computes all $[key, bkey]$ pairs on the key-path until it can proceed,
- broadcasts updated tree $\widehat{T}_{s_t}$ including only bkeys.

$$M_{s_t} \xrightarrow{\widehat{T}_{s_t}(BK^*_{s_t})} \mathcal{C} - \mathcal{L}$$

Step $h+1$: Every member computes the group key using $\widehat{T}_{s_t}$

Fig. 6. Partition Protocol

Partition Protocol….

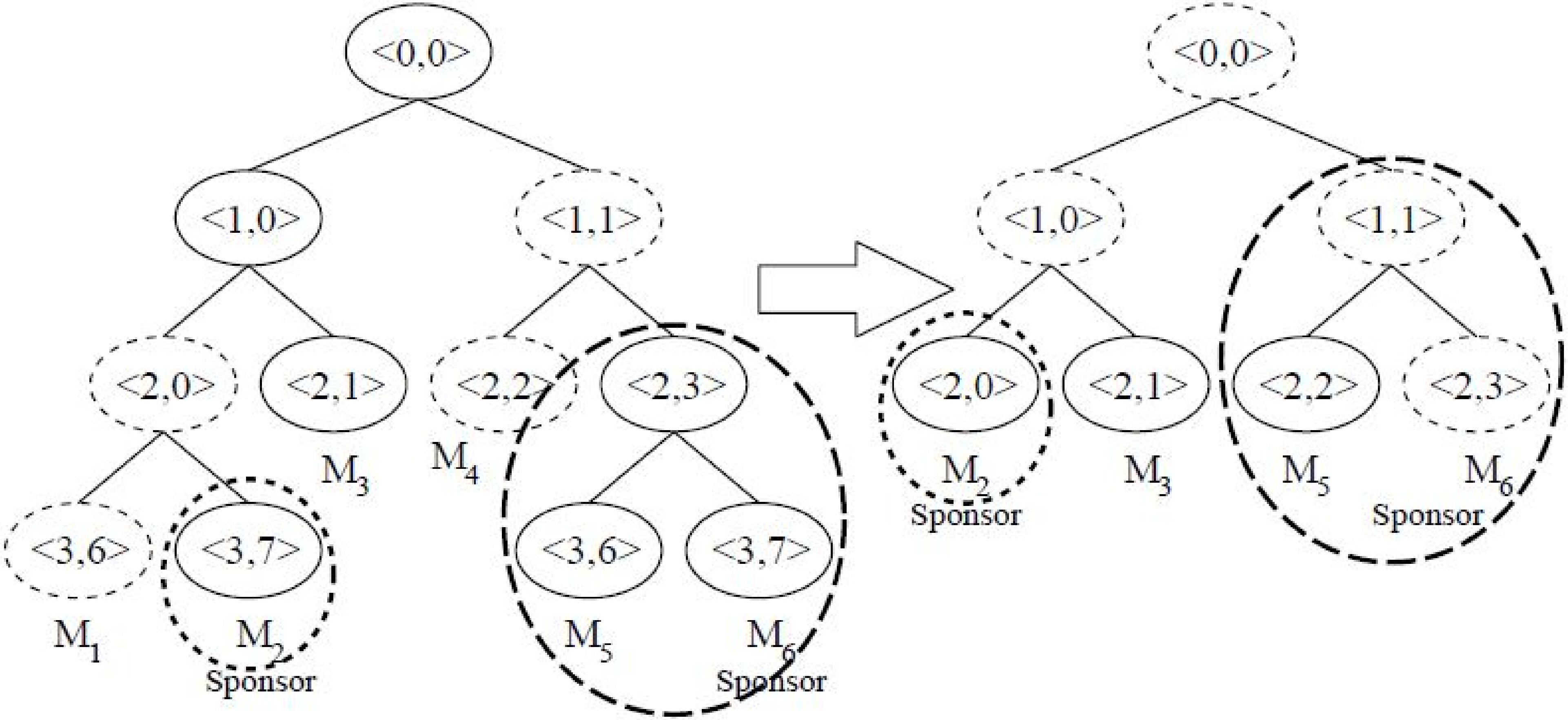

Fig. 7. Tree updating in partition operation

Merge Protocol

Step 1: Each M_{s_i} in each tree T_{s_i}:

- generate new share and compute all $[key, bkey]$ pairs on the key-path of T_{s_i},
- broadcast updated tree $\widehat{T}_{s_i}$ including only bkeys.

$$M_{s_i} \xrightarrow{\widehat{T}_{s_i}(BK^*_{s_i})} \cup_{i=1}^{k} \mathcal{C}_i$$

Step 2: Every member:

- update key tree by adding new trees and new intermediate nodes,
- remove all keys and bkeys from the leaf node related to the sponsor to the root node,

Each sponsor M_{s_t} additionally:

- compute all possible $[key, bkey]$ pairs on the key-path,
- broadcast updated tree $\widehat{T}_s$.

$$M_{s_t} \xrightarrow{\widehat{T}_{s_t}(BK^*_{s_t})} \cup_{i=1}^{k} \mathcal{C}_i$$

Step 3 to h (Until a sponsor M_{s_j} computes the group key): Each sponsor M_{s_t}:

- compute all possible $[key, bkey]$ pairs on the key-path,
- broadcast updated tree $\widehat{T}_{s_t}$.

$$M_{s_t} \xrightarrow{\widehat{T}_{s_t}(BK^*_{s_t})} \cup_{i=1}^{k} \mathcal{C}_i$$

Step $h + 1$: Every member computes the group key using $\widehat{T}_{s_t}$

Fig. 8. Merge Protocol

Merge Protocol….

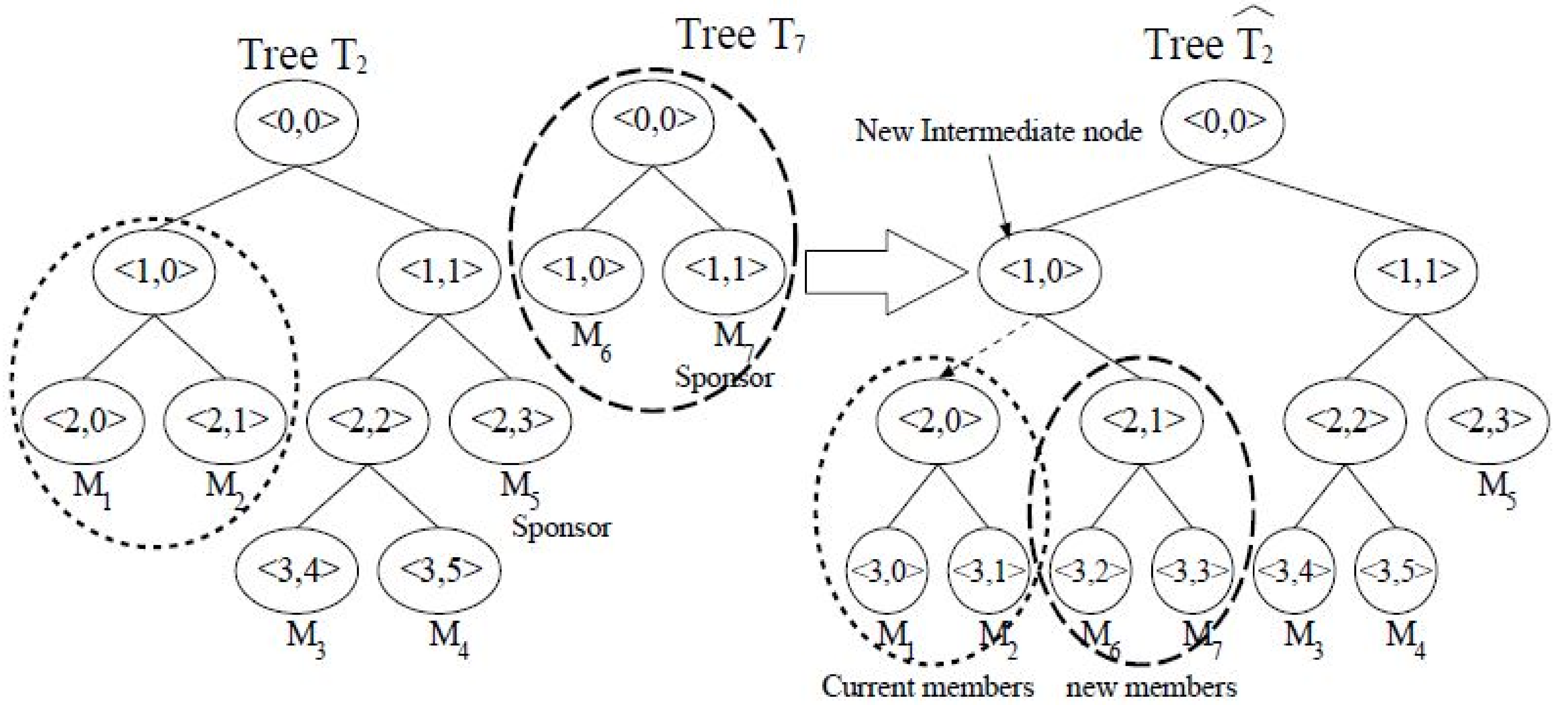

Fig. 9. Tree update in merge

Tree Management

- Tree management has the following goals:
  1. Balanced key tree
  2. Minimum number of modular exponentials
  3. Minimum number of protocol rounds

Policy of Additive and Subtractive Events

- Selection of insertion node
- Tree balancing scheme for subtractive events

```
merge_trees (T_h, T_l) {
  T = T_h
  i = 1, j = 2^i-1;

  While (1) {
    If (height (T_l) >= Max {height (T_<i, j>) | 0 <= j < 2^i}) {
      // If the height of the smaller tree is
      // greater than that of all subtrees
      result = T_h // Nowhere to join, join to root
      Break
    } EndIf

    If (T_l is joinable to node <i, j> of tree T_h){
      result = T_<i, j> // Join to node T_<i, j>
    } EndIf
    Else{
      j--
      If(j < 0){
        i++, j = 2^i-1
      } EndIf
    } EndElse
  } EndWhile

  // Merge two trees
  T_<i+1, 2j> = T_<i, j>
    // Old T_<i, j> becomes the left child of new T_<i,j>
  T_<i+1, 2j+1> = T_l
    // T_l becomes the right child of new T_<i, j>

  Return T
}
```

Sponsor Selection Summary

- Additive Events
- Subtractive Events
- Partition Events
- This cover only initial protocol round.
- Roles –
  1. Refresh its key share
  2. Compute all *[key, bkey]* pairs as far as the key path as possible
  3. Broadcast updated key tree to all *current* group members

Order of Content

- Introduction
- Assumptions and Requirements
- Cryptographic Properties
- Notations
- TGDH Protocols
- ***Practical Aspects (Issues and solutions)***
- Performance Analysis
- Related Work
- Conclusion

Implementation Architecture

- TREE_API
- Contains the following function calls
  1. tree_new_user
  2. tree_merge_req
  3. tree_cascade
- Protocol Unification Benefits:
  1. Simplify implementation and minimize the size.
  2. Overall security and correctness is easier to demonstrate.
  3. With slight modification, TGDH is self-stabilizing and fault tolerant.

```
receive msg (msg type = membership event)
construct new tree
while there are missing bkeys
  if ((I can compute any missing keys and I am the sponsor) ||
      (sponsor computed a key))
    while(1)
      compute missing (key, bkey) pairs
      if (I cannot compute)
        break
      endif
    if (others need my information)
      broadcast new bkeys
    endif
  endif
  receive msg
  if (msg type = broadcast)
    update current tree
  endif
endwhile
```

Fig. 10. Unified protocol pseudocode

Cascaded Events

```
receive msg (msg type = membership event)
construct new tree
while there are missing bkeys
  if ((I can compute any missing keys and I am the sponsor) ||
      (sponsor computed a key))
    while(1)
      compute missing (key, bkey) pairs
      if (I cannot compute)
        break
      endif
    if (others need my information)
      broadcast new bkeys
    endif
  endif
  receive msg
  if (msg type = broadcast)
    update current tree
  else (msg type = membership event)
    construct new tree
  endif
endwhile
```

Fig. 11. Self-stabilizing protocol pseudocode

Self Clustering

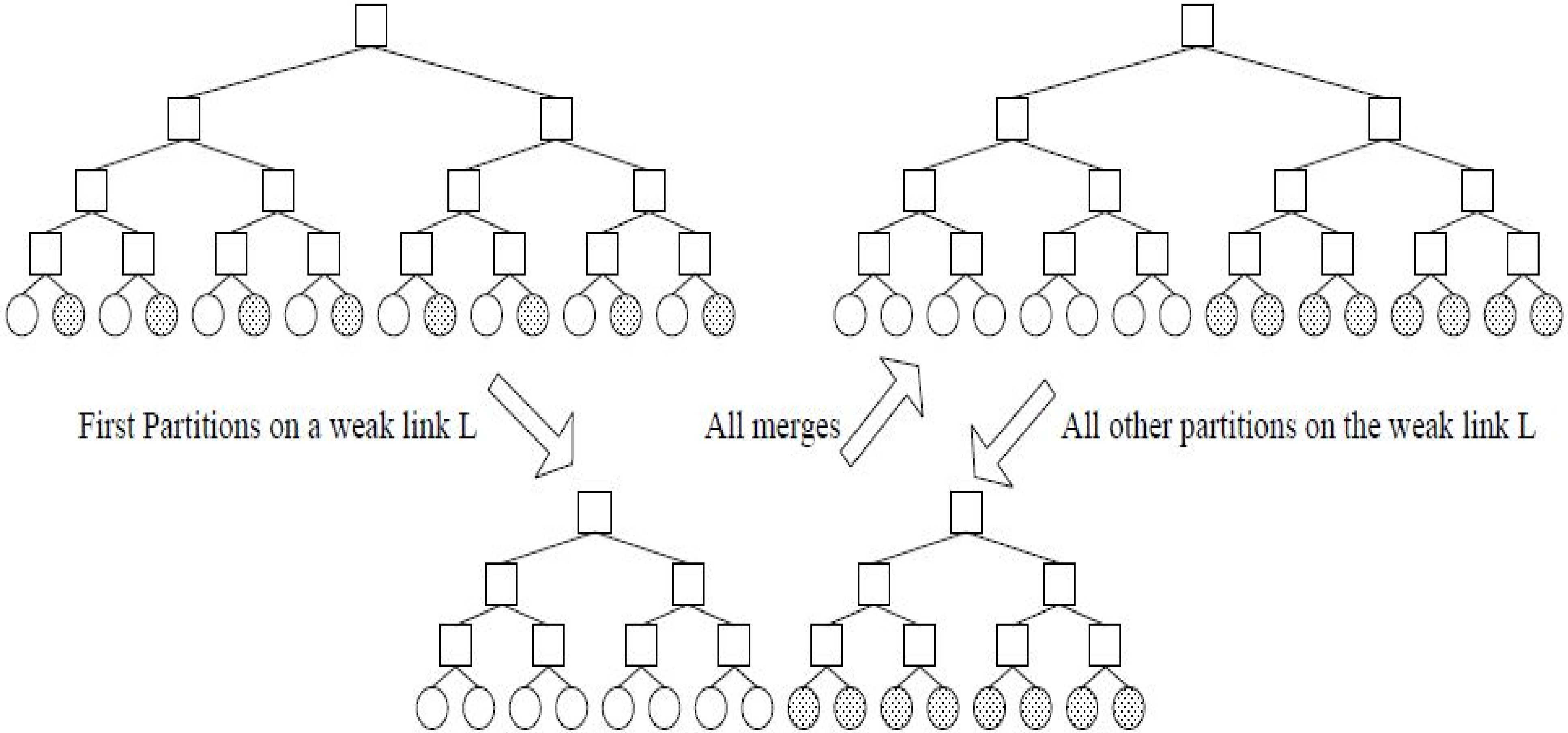

Fig. 13. An Extreme Example of Self-Clustering

Order of Content

- Introduction
- Assumptions and Requirements
- Cryptographic Properties
- Notations
- TGDH Protocols
- Practical Aspects (Issues and solutions)
- ***Performance Analysis***
- Related Work
- Conclusion

Performance Analysis

- Complexity Analysis

Table 1. Communication and Computation Costs

		Communication		Computation		
		Rounds	Messages	Exponentiations	Signatures	Verifications
GDH	Join	4	$n+3$	$n+3$	4	$n+3$
	Leave	1	1	$n-1$	1	1
	Merge	$m+3$	$n+2m+1$	$n+2m+1$	$m+3$	$n+2m+1$
	Partition	1	1	$n-p$	1	1
TGDH	Join	2	3	$3h-3$	2	3
	Leave	1	1	$3h-3$	1	1
	merge	$\lceil \log_2 k \rceil + 1$	$2k$	$3h-3$	$\lceil \log_2 k \rceil + 1$	$\lceil \log_2 k \rceil$
	Partition	$min(\lceil \log_2 p \rceil + 1, h)$	$min(2p, \lceil \frac{n}{2} \rceil)$	$3h-3$	$min(\lceil \log_2 p \rceil + 1, h)$	$min(2p, \lceil \frac{n}{2} \rceil)$
STR	Join	2	3	4	2	3
	Leave	1	1	$\frac{3n}{2}+2$	1	1
	Merge	2	$k+1$	$3m+1$	2	3
	Partition	1	1	$\frac{3n}{2}+2$	1	1
BD	Join	2	$2n+2$	3	2	$n+3$
	Leave	2	$2n-2$	3	2	$n+1$
	Merge	2	$2n+2m$	3	2	$n+m+2$
	Partition	2	$2n-2p$	3	2	$n-p+2$

Performance Analysis….

- Join and Leave Cost Comparison

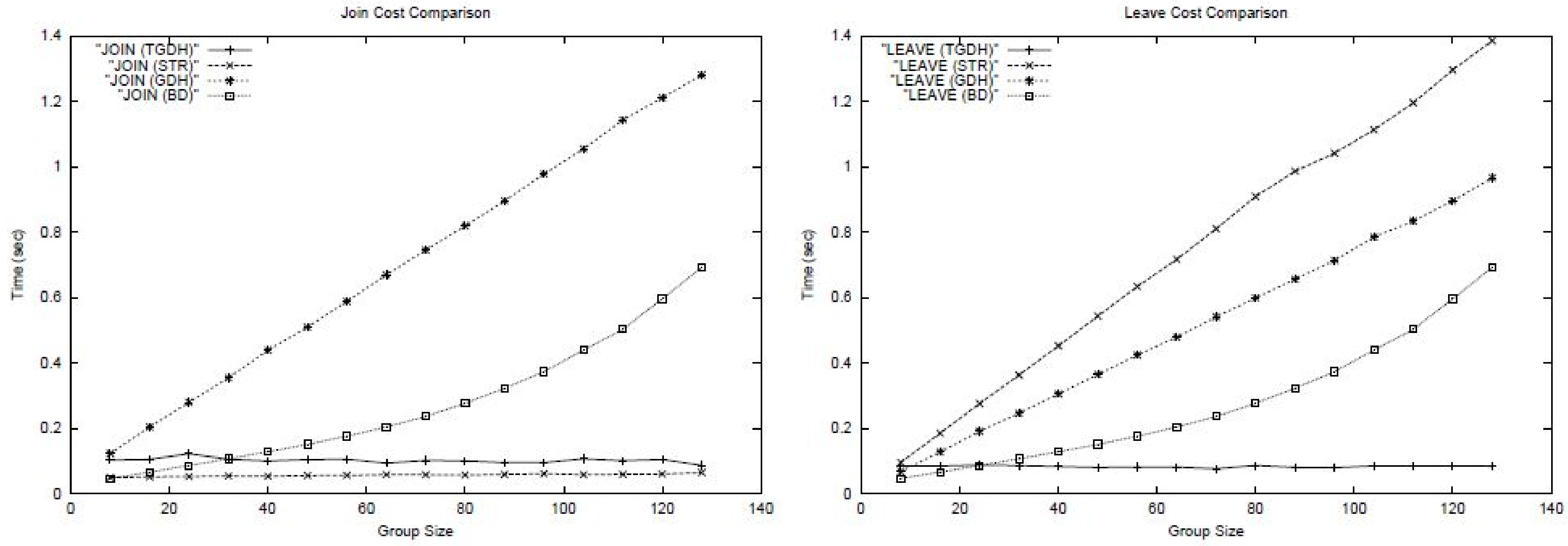

Fig. 14. Join and Leave Cost Comparison: (x, y) =(number of remaining group members after JOIN/LEAVE, computational overhead in seconds)

Performance Analysis….

- Partition Cost Comparison

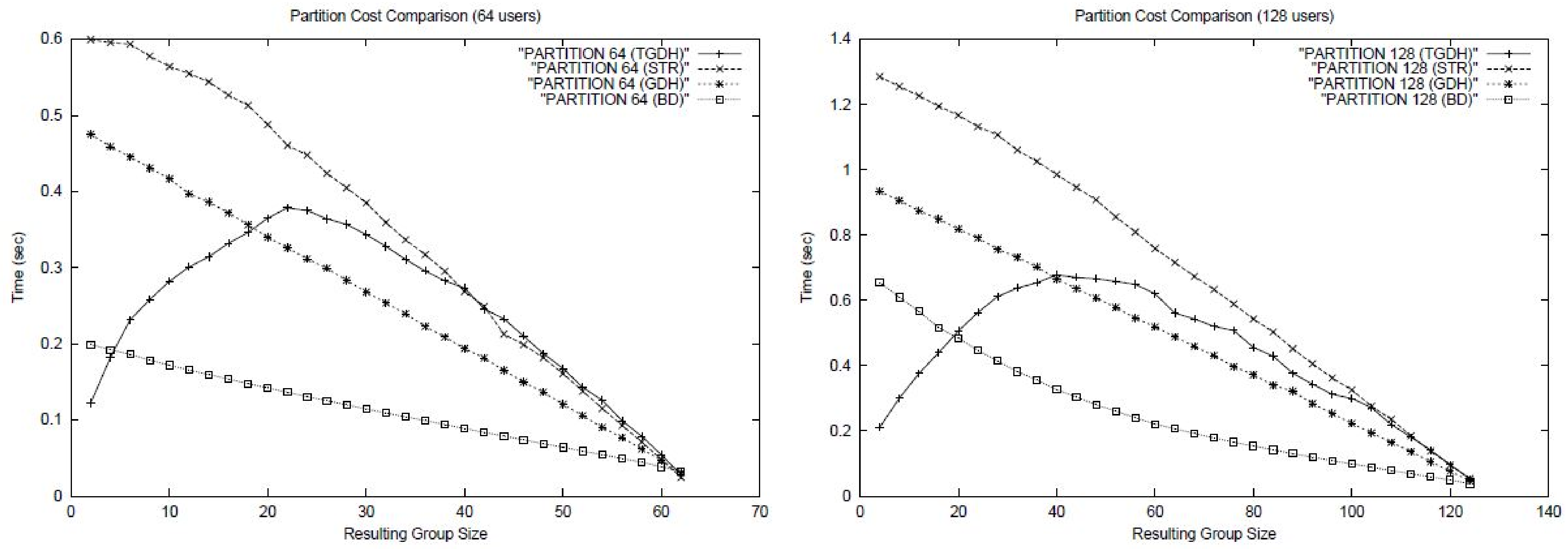

Fig. 15. Partition Cost Comparison: (x, y) =(number of remaining group members after the partition, computational overhead for an existing member if the original group shrinks to a group of x members), the original numbers of group members are 16, 32, 64, 128 respectively.

Performance Analysis….

- Merge Cost Comparison

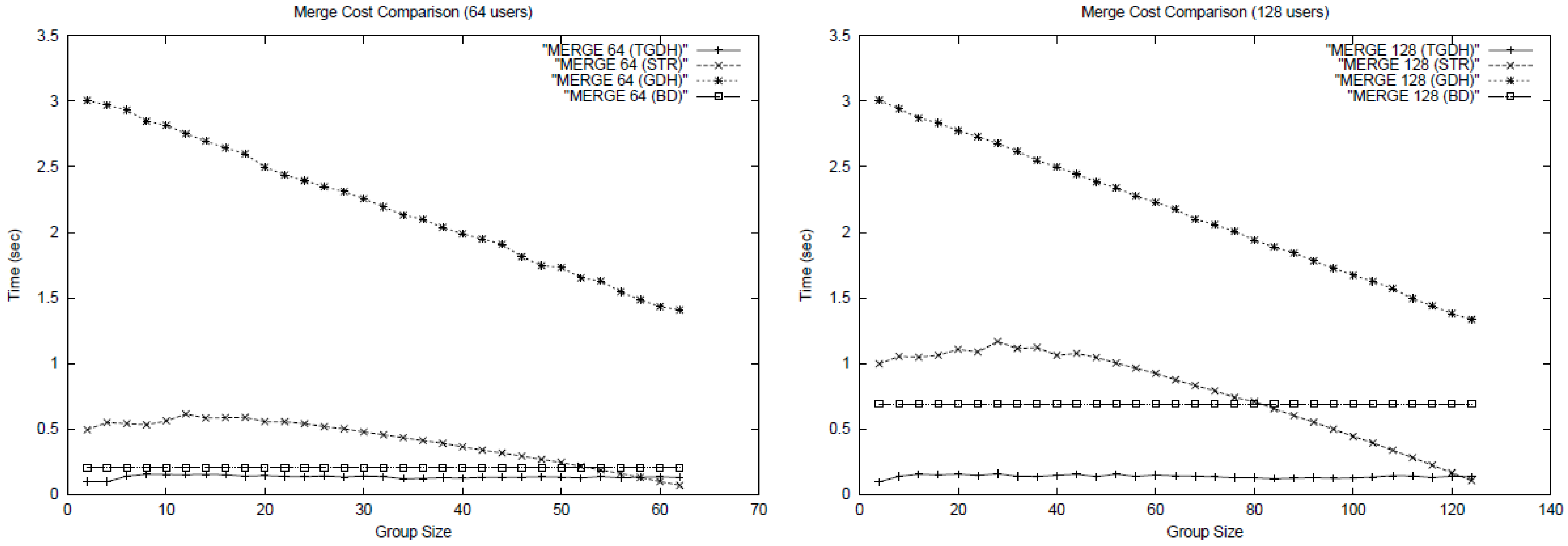

Fig. 16. Merge Cost Comparison: (x, y) =(number of current group members, computational overhead for a member located in the group of x members), after the membership event the number of group members becomes 16, 32, 64, and 128 respectively.

Order of Content

- Introduction
- Assumptions and Requirements
- Cryptographic Properties
- Notations
- TGDH Protocols
- Practical Aspects (Issues and solutions)
- Performance Analysis
- ***Related Work***
- Conclusion

Related Work

- Group Key Agreement Protocols
  1. Built upon 2 party Diffie Hellman – Ingemarsson et al (ING protocol). Disadvantage:
     a) Members have to start in sync
     b) n-1 rounds required to compute group key
     c) N sequential modular exponentials are required
  2. Proposed by Steer
     a) Well suited for adding new members – 2 rounds and 4 modular exponentiations
     b) Exclusion was relatively difficult
  3. Perrig extended OFT (One-way Functional Tree)
     a) Served as a foundation for TGDH

Related Work

- Decentralized Group Key Distribution Protocols
  1. Involve dynamically selecting a group member who generates and distributes keys to other group members.
  2. First protocol proposed was by Waldvogel. Unfortunately the scheme was insecure.
  3. Dondeti modified OFT to provide dynamic server election. This did not handle merge and partition events
  4. Many more....

Order of Content

- Introduction
- Assumptions and Requirements
- Cryptographic Properties
- Notations
- TGDH Protocols
- Practical Aspects (Issues and solutions)
- Performance Analysis
- Related Work
- ***Conclusion***

Conclusion

- Novel decentralized group key management approach – TGDH
- Unify 2 trends – *Key trees and Diffie-Hellman key exchange.*
- Robust to cascaded key management operations.
- Secure, simple and very efficient key management solution.

Thank you