Secure group software distribution for AMI

Marco Tiloca

(SICS Swedish ICT)

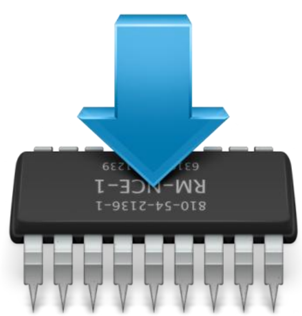
Exabier Bilbao Hernandez

(ZIV)

Secure group software distribution for AMI

- **Need for software upgrade**
  - Fix vulnerabilities
  - Upon specific events
  - On a regular basis

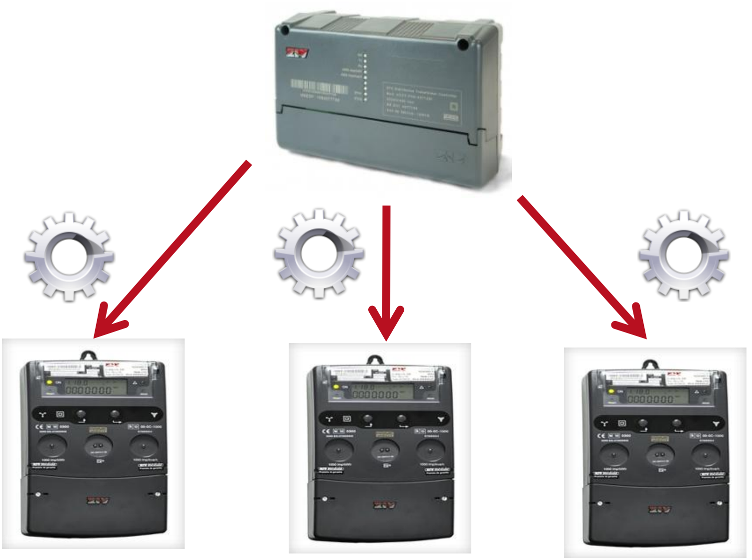
- **Typical software upgrades**
  - Vulnerability patches
  - Extension modules
  - Updated firmware version



Secure group software distribution for AMI

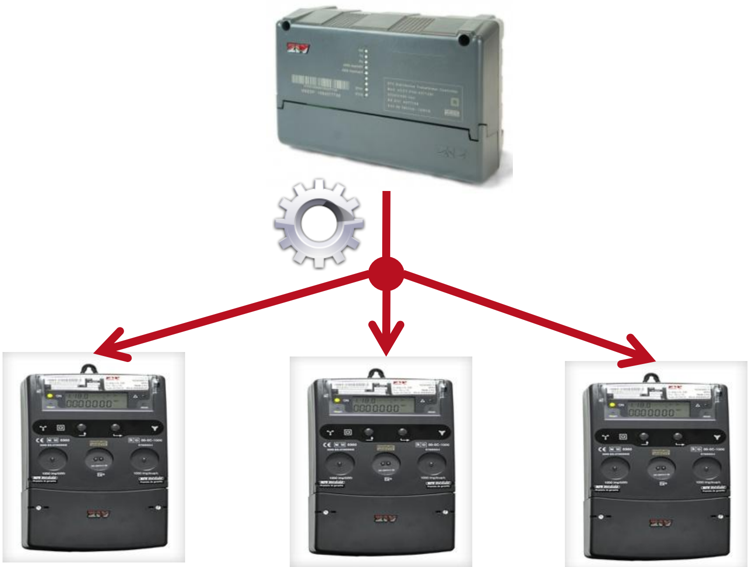
• Point-to-point distribution

- Update one meter at a time
- Long time to update all meters
- Inefficient performance



• Group multicast distribution

- Meters organized as a group
- Single update to the group
- Much shorter update time
- Efficient performance



Secure group software distribution for AMI

- **Need for secure distribution**
  - Integrity and authentication
  - Confidentiality of updates

- **Requirements**
  - Ensure that updates are the intended ones
  - Updates distributed by the intended sources
  - Prevent reverse engineering of updates

- **How to ensure secure software distribution?**

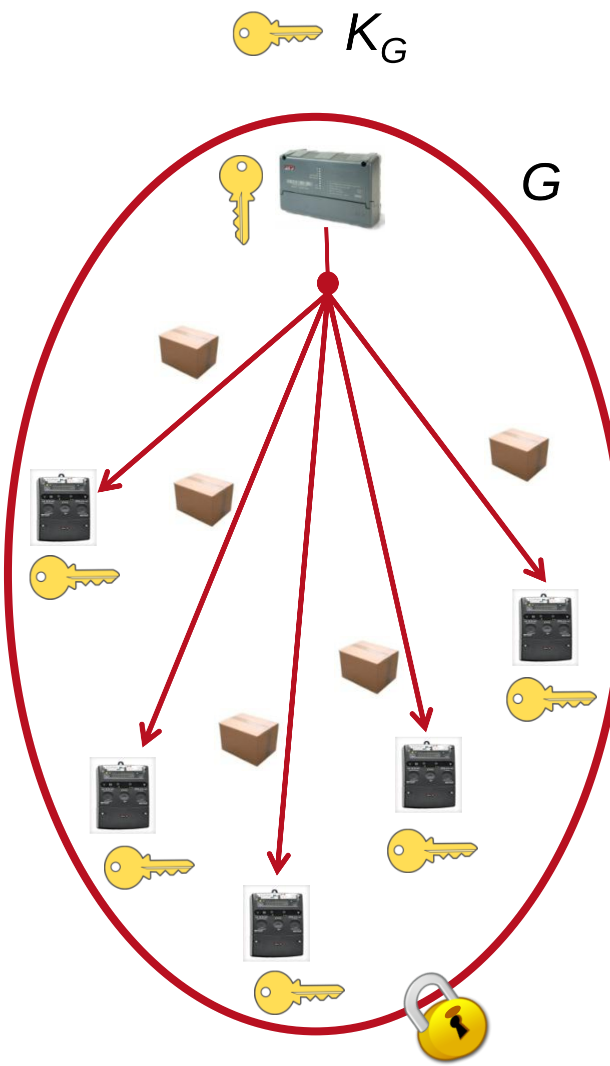
Secure group software distribution for AMI

- **Secure point-to-point distribution**
  - One pairwise key per meter
  - Pairwise keys shared with data concentrator
  - Easy to address addition/removal of meters
  - <u>Inefficient performance and poor scalability</u>

- **Secure multicast distribution**
  - One group key for all the meters in the group
  - Group key shared with the data concentrator
  - Complex to address addition/removal of meters
  - <u>Efficient performance and high scalability</u>

Secure group software distribution for AMI

- **Group membership**
  - A meter explicitly joins the group as a member
  - A meter can explicitly ask to leave the group
  - A meter can be forced to leave, if compromised



- **Secure message exchange**
  - Meters receive multicast software updates
  - Group key K_G shared with the data concentrator
  - K_G is used to secure/unsecure software updates

Secure group software distribution for AMI

- **Group key renewal (rekeying)**
  - On a periodical fashion
  - Upon joining of a new meter (backward security)
  - Upon leaving of a meter (forward security)

- **Centralized key manager**
  - In charge of the rekeying process
  - Likely to be the data concentrator
  - Possible cooperation with the Back-end

- **What specific key management scheme?**
  - Securely revoke the current group key
  - Securely distribute a new one

Secure group software distribution for AMI

• Basic approach (join rekeying)

- Single-step rekeying
- New key sent once to the group
- Protected by the current group key

• Basic approach (leave rekeying)

- Point-to-point rekeing
- New key sent separately to each meter
- Protected by respective pairwise key

• Performance

- Very easy to implement and execute
- The leave rekeying is very inefficient
- In fact, it is a total group re-initialization!

Secure group software distribution for AMI

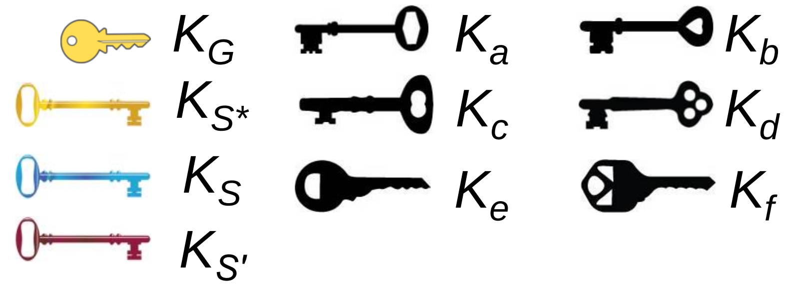
- **GREP: a novel rekeying scheme (*)**
  - Highly efficient and short in time
  - Highly scalable with the group size
  - Join and leave rekeying procedures

- **Two concepts combined together**
  - Logical subgrouping of group members
  - History of join events in the group
- **Administrative key material**
  - Limited number of encryption keys
  - Additional tokens reflecting the join history

(*) M. Tiloca and G. Dini, *GREP: a Group REkeying Protocol based on member's join history*, The twenty-first IEEE Symposium on Computers and Communications (ISCC 2016), pp 326-333, Messina (Italy), 2016

Secure group software distribution for AMI

Key material (system view)



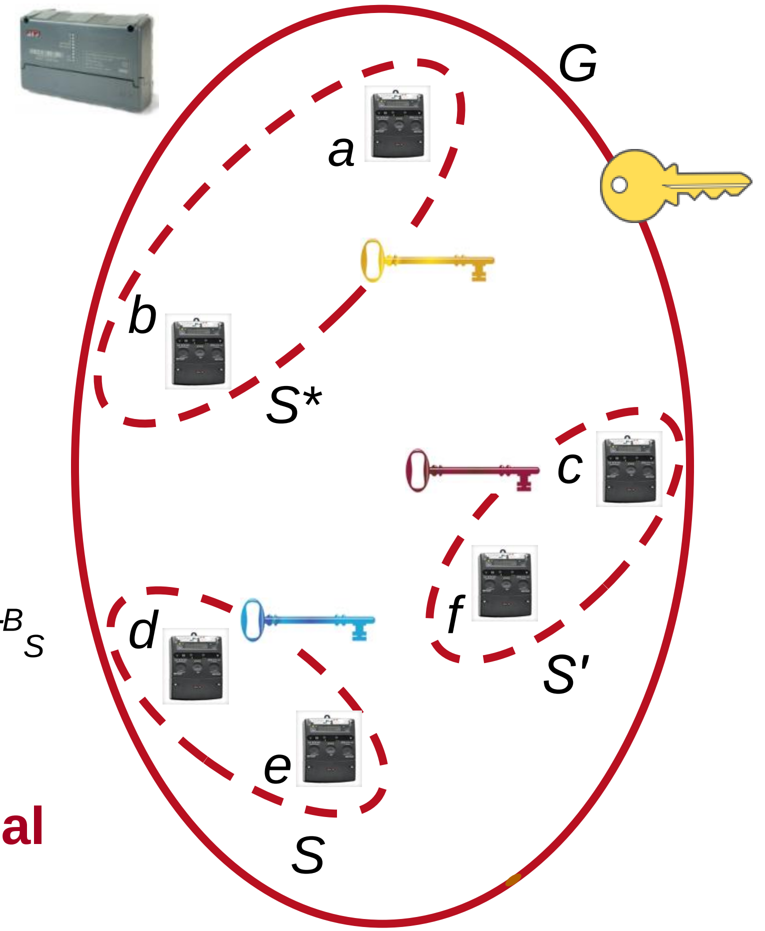
- **Keys and tokens**

- **The group G is associated to K_G**

- **Each meter in a subgroup S is:**
  - Associated to a node key K_u
  - Associated to a backward node token $t^B{}_u$
  - Associated to a forward node token $t^F{}_u$

- **Each subgroup S is:**
  - Associated to a subgroup key K_S
  - Associated to a backward subgroup token $st^B{}_S$
  - Associated to a forward subgroup token $st^F{}_S$

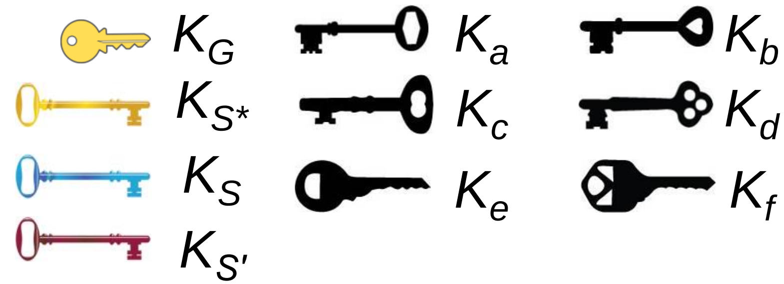
- **The key manager stores all the key material**

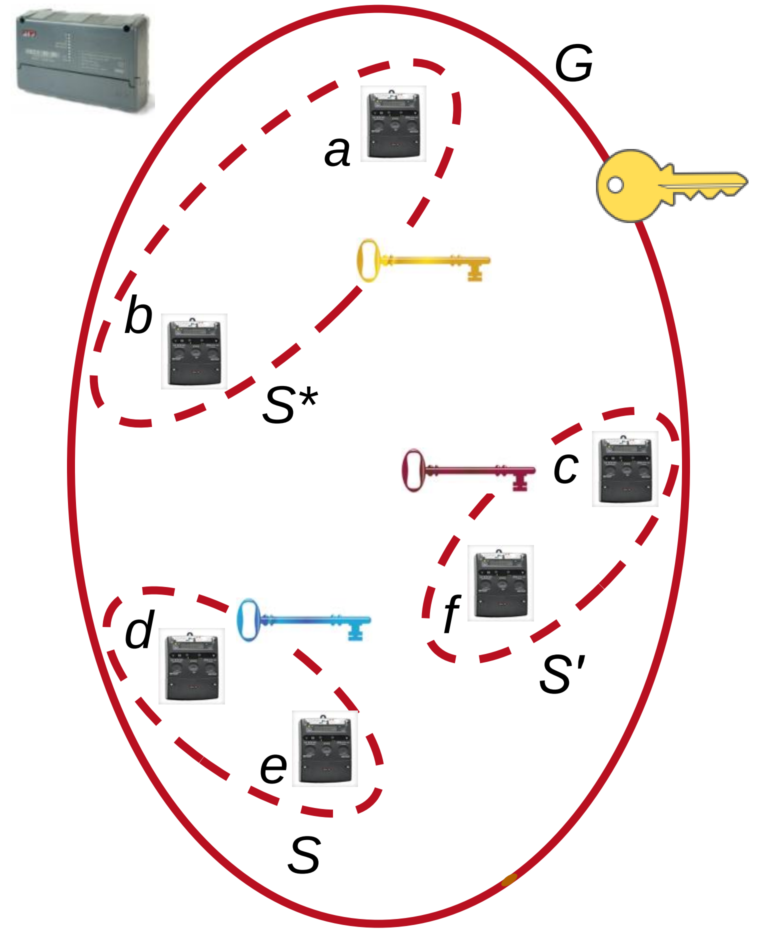


Secure group software distribution for AMI

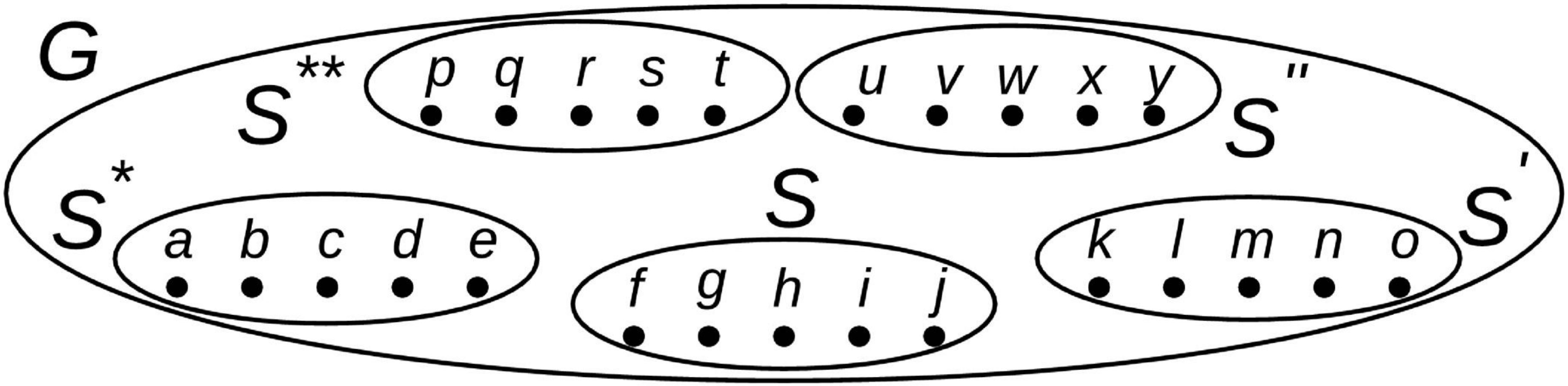
Key material (meter view)

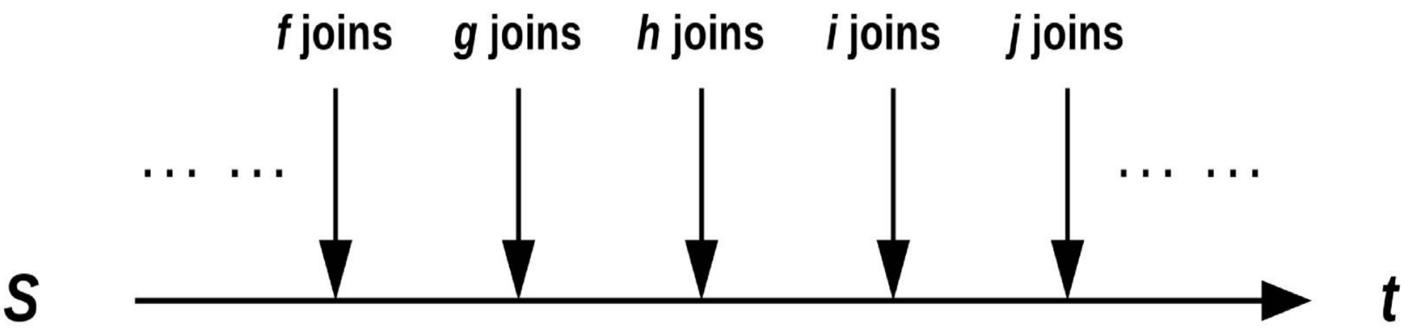
- **1 group key K_G**
  - Shared with all other meters in *G*
- **1 subgroup key K_S**
  - Shared with all other meters in subgroup *S*
- **1 node key K_u**
  - Shared with the key manager only
- **Multiple <u>backward</u> node tokens**
  - One for each meter that joined *S* <u>before</u> *u*
- **Multiple <u>forward</u> node tokens**
  - One for each meter that joined *S* <u>after</u> *u*
- **Multiple <u>backward</u> subgroup tokens**
  - One for each subgroup created <u>before</u> *S*
- **Multiple <u>forward</u> subgroup tokens**
  - One for each subgroup created <u>after</u> *S*



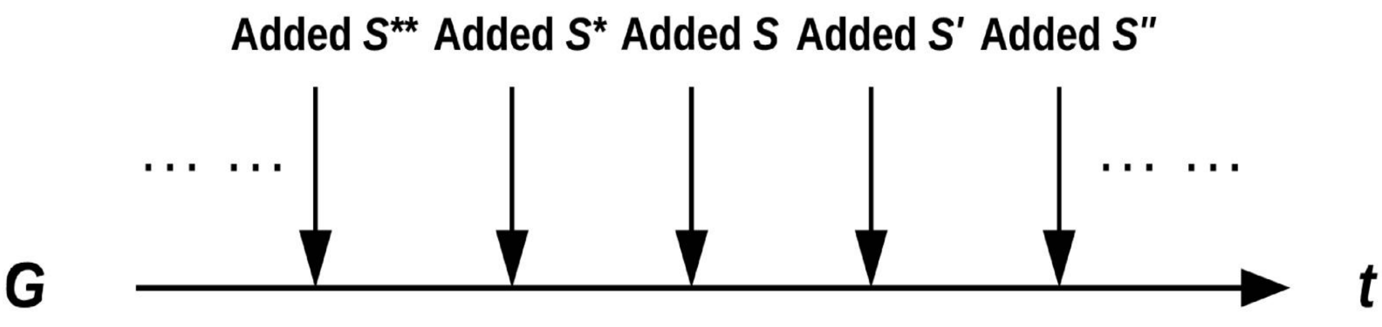


Secure group software distribution for AMI





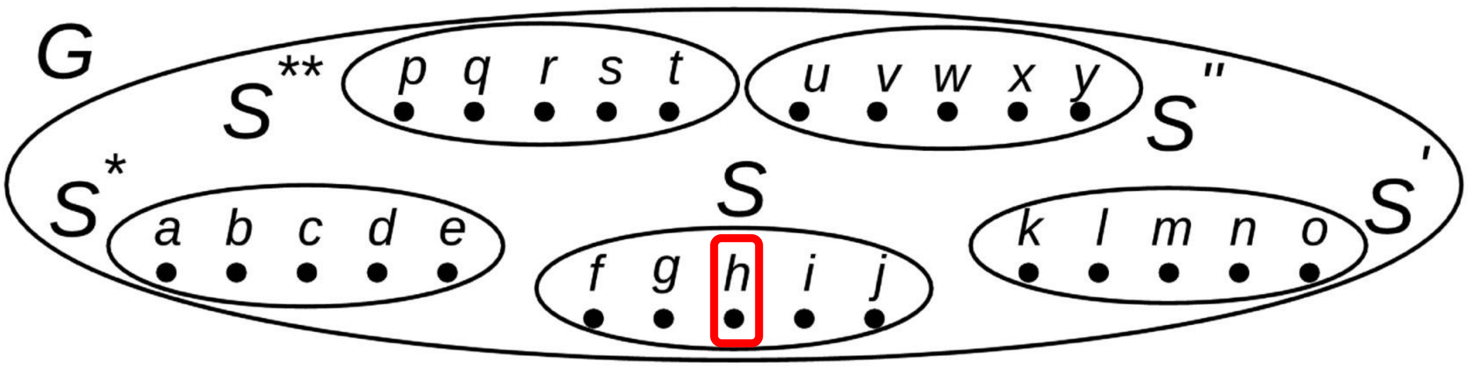
Join history of subgroup *S*



Addition history of group *G*

Secure group software distribution for AMI

Subgroup	Meter	Backward node tokens	Forward node tokens	Backward subgroup tokens	Forward subgroup tokens
S	f	—	$t^F_g, t^F_h, t^F_i, t^F_j$	$st^B_{S^{**}}, st^B_{S^*}$	$st^F_{S'}, st^F_{S''}$
	g	t^B_f	t^F_h, t^F_i, t^F_j	$st^B_{S^{**}}, st^B_{S^*}$	$st^F_{S'}, st^F_{S''}$
	h	t^B_f, t^B_g	t^F_i, t^F_j	$st^B_{S^{**}}, st^B_{S^*}$	$st^F_{S'}, st^F_{S''}$
	i	t^B_f, t^B_g, t^B_h	t^F_j	$st^B_{S^{**}}, st^B_{S^*}$	$st^F_{S'}, st^F_{S''}$
	j	$t^B_f, t^B_g, t^B_h, t^B_i$	—	$st^B_{S^{**}}, st^B_{S^*}$	$st^F_{S'}, st^F_{S''}$

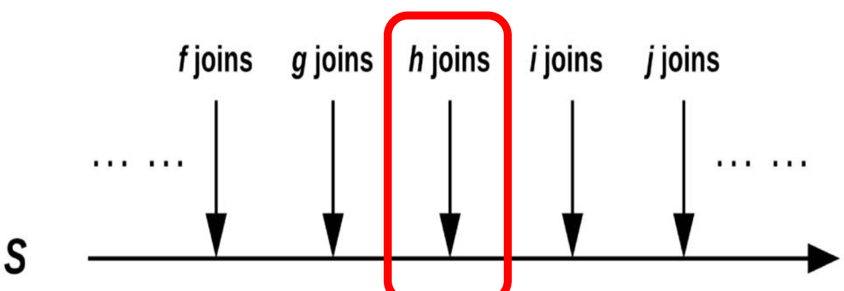


Keys on *h*

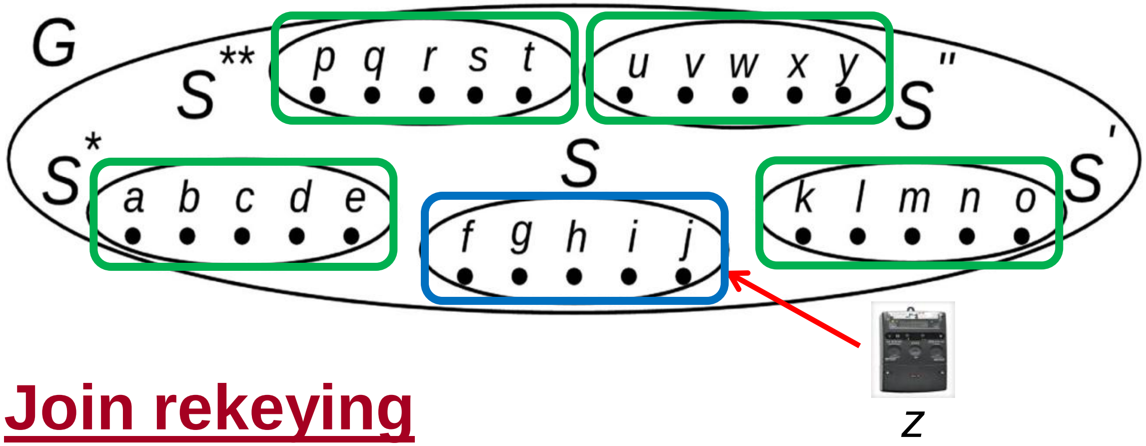
K_G

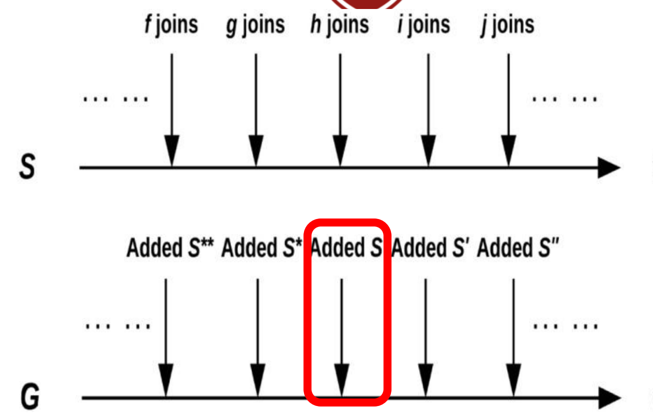
K_S

K_h



Secure group software distribution for AMI

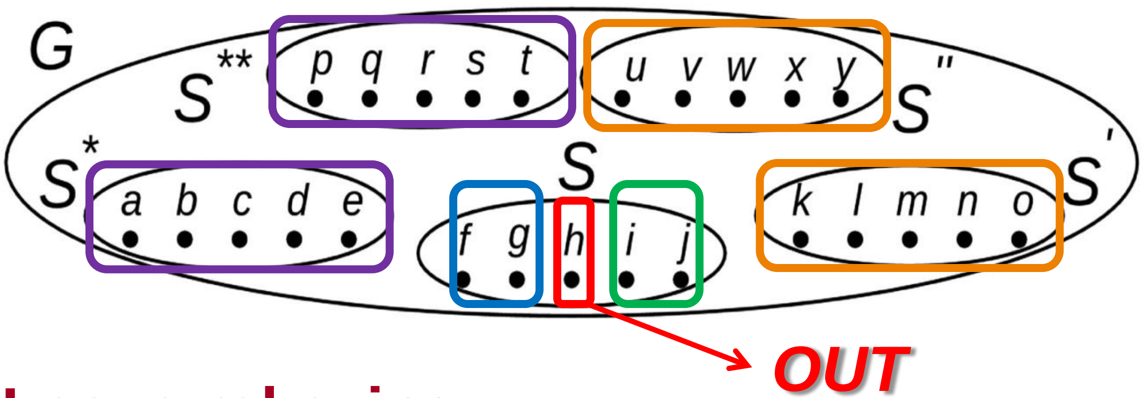


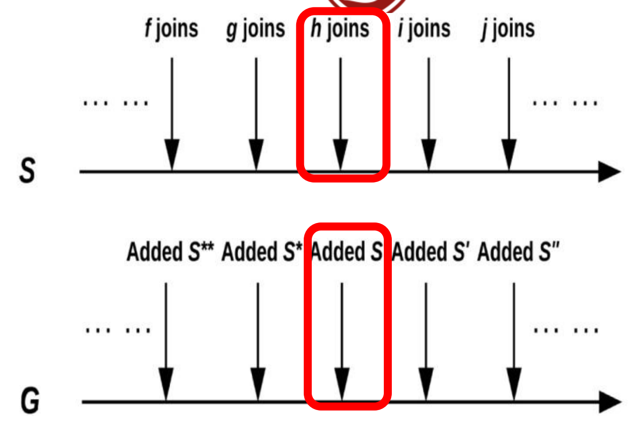


Join rekeying

- **The key manager generates:**
  - nid_z , K_z , K_R , $t^B{}_Z$, t_M
  - $K^+{}_G$ = KDF(K_G || K_R) , $K^+{}_S$ = KDF(K_S || K_R) , $t^F{}_z$ = KDF(t_M || K_R)
- **The key manager transmits two messages:**
  - JM1 KM → S : < nid_z , E({t_M, K_R} , K_S) > // To rekey meters in subgroup S
  - JM2 KM → G : < E(K_R , K_G) > // To rekey meters <u>not</u> in subgroup S
- **The key manager provides meter *z* with:**
  - nid_z , $K^+{}_G$, $K^+{}_S$, K_z
  - Backward node tokens of nodes *f*, *g*, *h*, *i*, *j*
  - Backward subgroup tokens of S^{**} and S^*
  - Forward subgroup tokens of S' and S''

Secure group software distribution for AMI





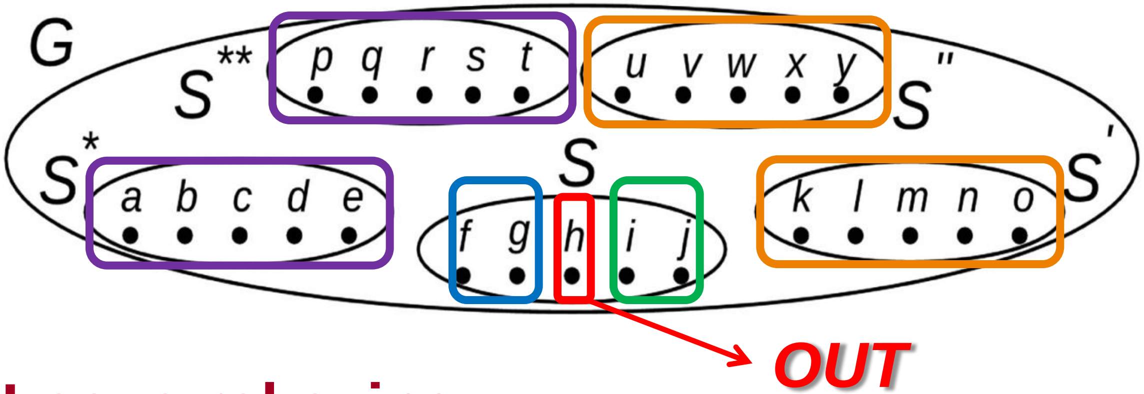
Leave rekeying

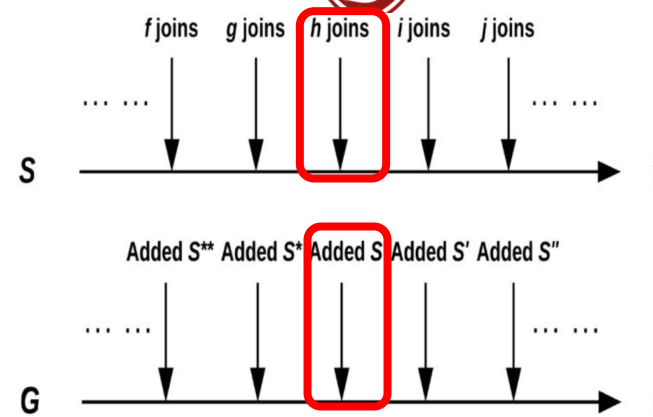
- **The key manager generates:**
  - K_R , $K^+_G = \mathrm{KDF}(K_G \,||\, K_R)$, $K^+_S = \mathrm{KDF}(K_S \,||\, K_R)$
  - $K_F = \mathrm{KDF}(t^F_h)$, $K_B = \mathrm{KDF}(t^B_h)$, $K^S_F = \mathrm{KDF}(st^F_S)$, $K^S_B = \mathrm{KDF}(st^B_S)$

- **The key manager transmits two messages:**
  - LM1 KM $\rightarrow S : < nid_h$, E(K_R , K_F), E(K_R , K_B) > // To rekey meters in subgroup S
  - LM2 KM $\rightarrow G : < sid_S$, E(K_R , K^S_F), E(K_R , K^S_B) > // To rekey meters <u>not</u> in subgroup S

Use key material that the leaving node does not know!

Secure group software distribution for AMI





Leave rekeying

- **Meters in subgroup *S* derive:**
  - Derive $K^{+}_{G} = \text{KDF}(K_G \mid\mid K_R)$, $K^{+}_{S} = \text{KDF}(K_S \mid\mid K_R)$
  - Delete the tokens associated to *h*
  - Update stored node tokens and subgroup tokens t as $t \leftarrow \text{H}(t \mid\mid K_R)$

- **Meters <u>not</u> in subgroup *S* derive:**
  - Derive $K^{+}_{G} = \text{KDF}(K_G \mid\mid K_R)$
  - Update stored subgroup tokens t as $t \leftarrow \text{H}(t \mid\mid K_R)$

Use key material that the leaving node does not know!

Performance evaluation

- **Analytical results and trends**
  - Storage overhead (stored information items)
  - Computing overhead (performed cryptographic operations)
  - Communication overhead (received information items)
- **Reference scenario**
  - n = 1024 group members
  - p subgroups of m members each, i.e. $n = p \times m$
  - Keys, tokens and IDs have the same size
  - Key derivation and hash functions have the same complexity
- **Focus on the worst-case condition**

SEGRID

Performance evaluation

- **Storage overhead**
  - Stored items: (*2* x √ *n*) for each group member

Storage overhead $\mathcal{O}(\sqrt{n})$

- **Join rekeying overhead**
  - A meter *u* joins subgroup *S*. Then, …
  - the worst case regards any meter *h* already in *S*
  - Computing: 1 decryption and 3 hash functions
  - Communication: 4 items received

Computing overhead	Constant
Communication overhead	Constant

- **Leave rekeying overhead**
  - A meter *u* leaves subgroup *S*. Then, …
  - the worst case regards any other meter *h* in *S*
  - Computing: 1 decryption and 64 hash functions
  - Communication: 6 items received

Computing overhead	Constant
Communication overhead	Constant

GREP is secure, fast, and highly scalable!

SEGRID

Conclusions

GREP

- **Novel group key management protocol**
  - Support for secure group software distribution
  - Logical sub-grouping of the group members
  - Members' join history and subgroups' addition history
- **Main benefits**
  - Highly scalable with the group size *n*
  - Affordable on resource-constrained nodes
  - Does not require a total group re-initialization
- **Impact on performance**
  - Memory overhead grows as $\sqrt{(n)}$
  - Join overhead is limited and constant
  - Leave overhead is limited and constant
  - The rekeying process requires a shorter time

SWEDISH ICT SICS

Secure group software distribution for AMI

Demo!

- **Secure software distribution**
  - First C++ implementation by SICS
  - Ported, up and running, on ZIV facilities
  - One data concentrator, multiple meters

- **Group maintenance operations**
  - Performed via multicast
  - Supported by group key management
  - Basic rekeying scheme vs. GREP

- **Two test cases**
  - Open/close disconnectors on smart meters
  - Firmware upgrade on smart meters

Thanks for your attention!

Marco Tiloca
marco@sics.se
http://www.sics.se/~marco