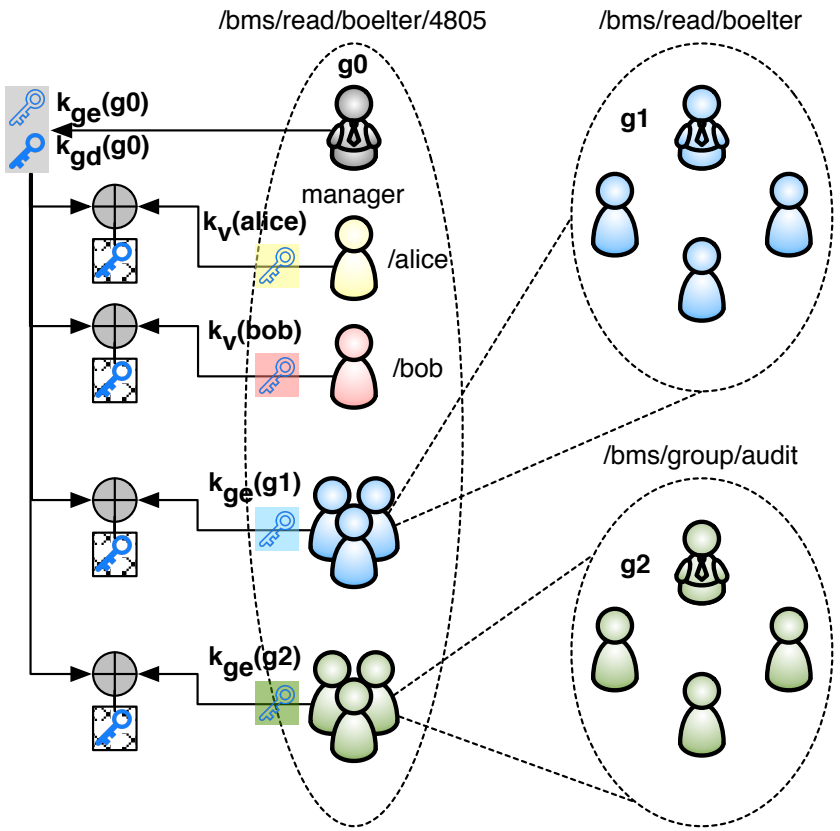
Group-based Encryption Protocol

Scenario

- One or more data producers
  - produced contents are encrypted
  - data is produced in a time sequence
- User group
  - group members have the same read access to data
  - a group member could be an individual user or another group
  - each group has a manager who can decide the membership
- Read access to data is granted through groups
  - data producer has a primary read access group
    - multiple producers may share the same primary read access group
  - manager of the primary read access group can
    - grant the access to another user or a secondary group by adding the user or group as a group member
  - a secondary group consists of individual users

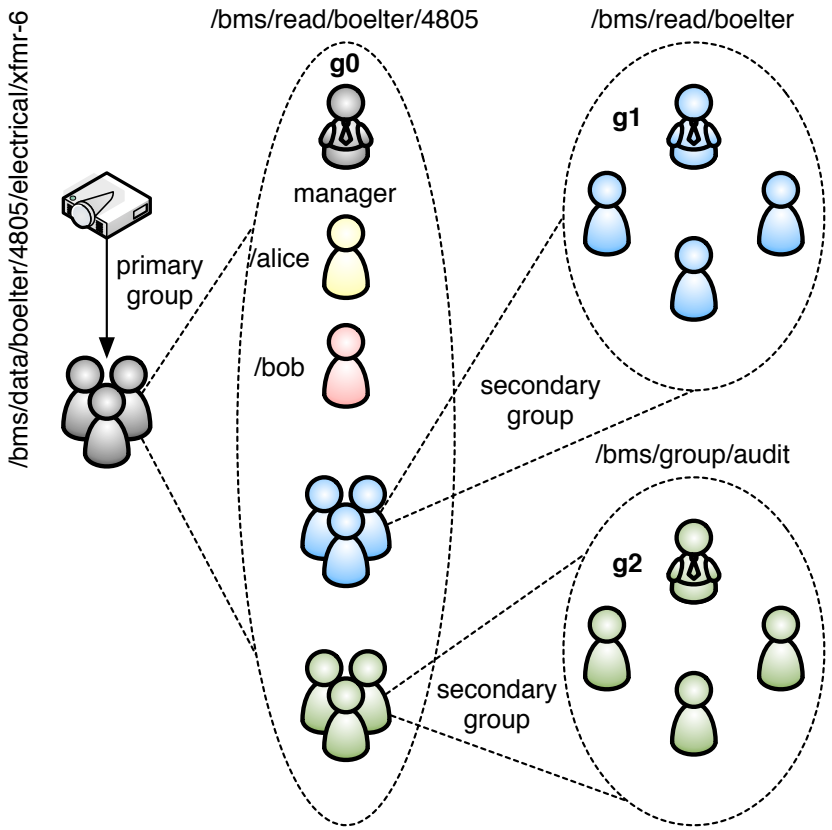
Group Keys

- Each group (either primary or secondary) has two pairs of public/private keys
  - group authority key: ($\mathbf{k_{gv}}$, $\mathbf{k_{gs}}$)
    - only used for verification/signing
    - private key owner: group manager
  - group encrypt/decrypt key: ($\mathbf{k_{ge}}$, $\mathbf{k_{gd}}$)
    - only used for encryption/decryption
    - private key owner: every group member
- Group decrypt key $\mathbf{k_{gd}}$
  - generated by group manager
  - encrypted with members' public key
    - if member is a group, encrypted with the member group's $\mathbf{k_{ge}}$
  - (optionally) signed with $\mathbf{k_{gs}}$



Primary/Secondary Groups

- Each producer
  - must have a primary read access group
  - may have one ore more secondary groups
- Secondary groups are managed as members of the primary group
  - the primary group's decrypt key $\mathbf{k_{gd}}$ is encrypted with secondary group's $\mathbf{k'_{ge}}$
  - members of a primary group also have the access to the primary group's $\mathbf{k_{gd}}$
- Producer only needs to encrypt its data encryption key $\mathbf{k_e}$ with its primary group's $\mathbf{k_{ge}}$



Primary/Secondary Groups (cont'd)

- Primary group's privilege
  - determined by the group name
  - group name is related to producer name
    - group name: /bms/read/boelter/4805
    - producer name: /bms/data/boelter/4805/electrical/xfmr-6

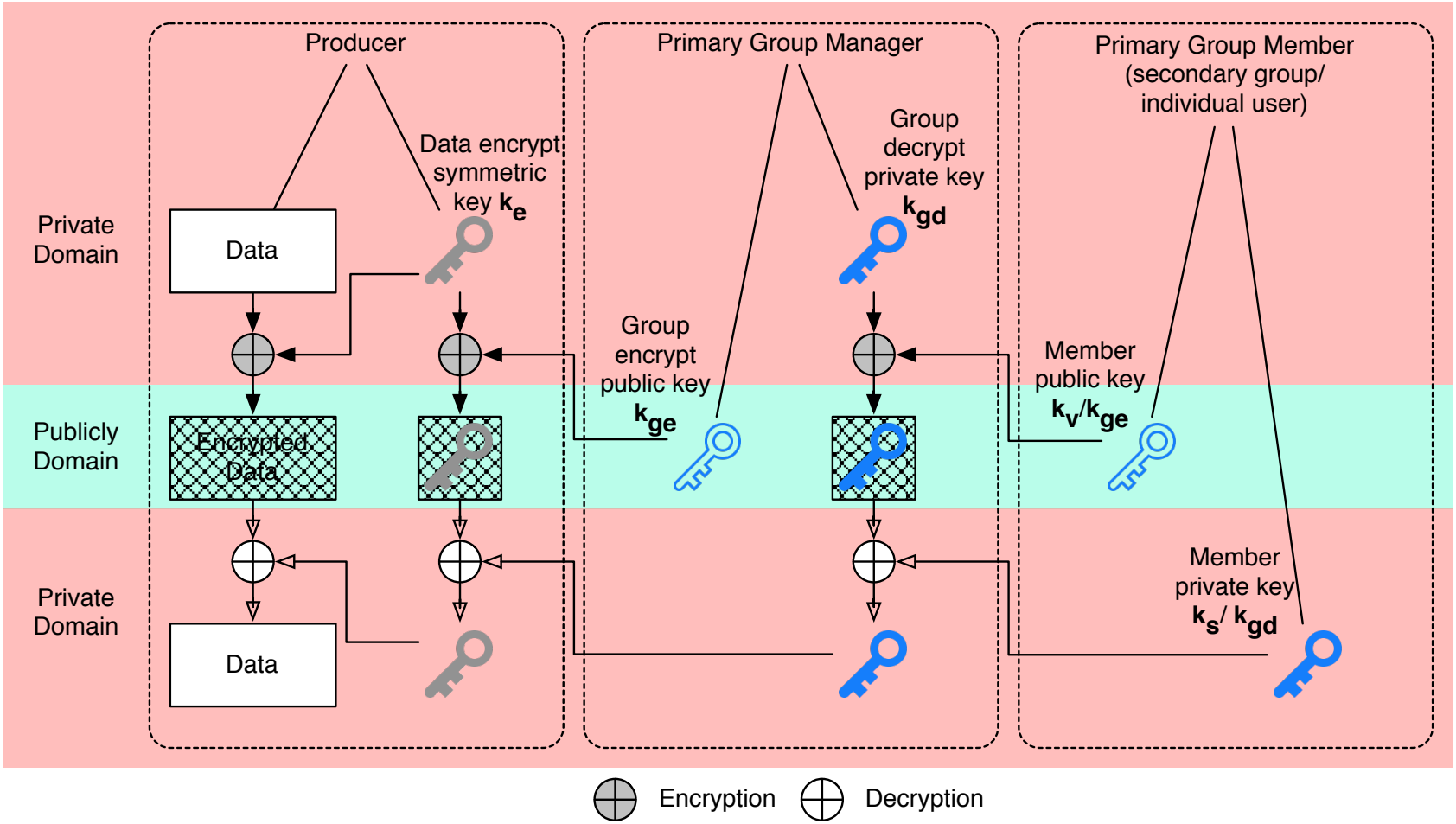
- Secondary group's privilege:
  - combination of primary groups of which the group holds a membership
  - group name is irrelevant to producer name
    - if a group /bms/group/audit is the member of both /bms/read/boelter and /bms/read/melnitz, the member of group /bms/group/audit has the access to data under both /bms/data/boelter and /bms/data/melnitz

- Ideally
  - the membership of primary groups are defined by secondary groups and are relatively stable
    - audit group are always authorized to read data from each building
  - the membership of secondary groups are defined by individual users and may change from time to time
    - a individual user may be occasionally added into/removed from the audit group

General Process

- Data publishing
  - generate content
  - encrypt content using a symmetric content encryption key $\mathbf{k_e}$
  - publish encrypted content
    - signed with the producer's private key
  - encrypt $\mathbf{k_e}$ using the primary group encryption public key $\mathbf{k_{ge}}$
  - publish encrypted $\mathbf{k_e}$
    - signed with the producer's private key
- Data consuming
  - fetch the encrypted content
  - fetch the encrypted content encrypt key $\mathbf{k_e}$ (through EncryptKeyLocator)
  - determine the corresponding primary group's encrypt key $\mathbf{k_{gd}}$
  - if a consumer is authorized (member of the primary group or secondary group), the consumer should have the primary group decrypt key $\mathbf{k_{gd}}$
  - decrypt content encrypt key $\mathbf{k_e}$
  - decrypt content
- Centralized encryption key management is avoided

General Process



Group Key Rollover

- Adding a new member does not require a new group encrypt/decrypt key
- A new group encrypt/decrypt key must be generated when a member is removed from the group
- A group manager may also periodically generate a new group encrypt/decrypt key
- Primary group key rollover
  - each decrypt key has a timestamp and represents the access to data produced during a certain period
  - access to a particular decrypt key must be explicitly granted
    - access to a decrypt key with a later timestamp does not imply the access to previous decrypt keys
- Secondary group key rollover
  - each decrypt key has a version number
  - a member has the access to all the previous versions of decrypt key
    - implicitly done through key chaining
    - a key of version N is encrypted with a key of version N+1

Encrypted Data Format

- Encode encryption related information in content
  - minimize packet format changes

```
Content               ::= CONTENT-TYPE TLV-LENGTH
                          EncryptedContent
EncryptedContent      ::= ENCRYPTED-CONTENT-TYPE TLV-LENGTH
                          KeyLocator
                          EncryptionAlgorithm
                          InitialVector?
                          EncryptedPayLoad
EncryptionAlgorithm   ::= ENCRYPTION-ALOGRITHM-TYPE TLV-LENGTH
                          nonNegativeInteger // algorithm id
InitialVector         ::= INITIAL-VECTOR-TYPE TLV-LENGTH
                          BYTE+
EncryptedPayLoad      ::= ENCRYPTED-PAYLOAD-TYPE TLV-LENGTH
                          BYTE+
```

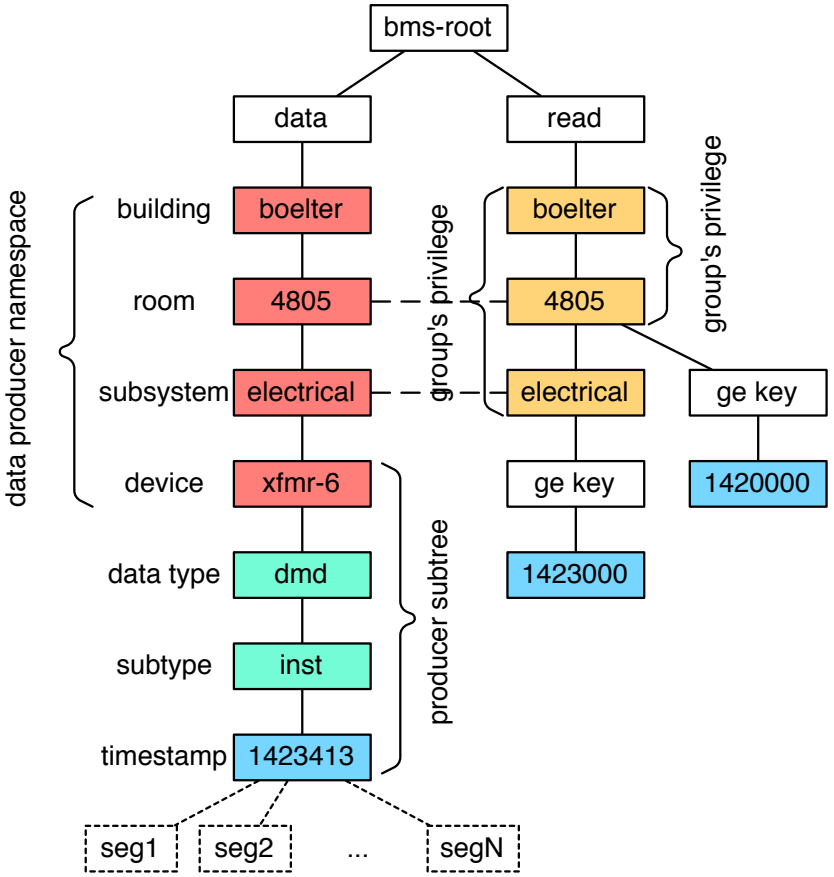
Encrypt Private Keys

- If we need to use a public key k^1_{pub} to encrypt a private key k^2_{priv}
- The content payload consists of two EncryptedContent TLV blocks
  - block 1: a symmetric key k_s encrypted using k^1_{pub}
    - ks length should be less then k^1_{pub}
  - block 2: private key k^2_{priv} encrypted using k_s
    - the EncryptionKeyLocator will be ignored

Naming Tree

- Two branches under the data root
  - Data branch
    - consists of producer's namespaces
    - producer may have sub tree under its own namespace
    - the basic data unit is at the timestamp level
      - data unit may consists of multiple segments
  - Read branch
    - consists of primary read access groups
    - node in read branch $\mathbf{n^r}$ has a corresponding node in data tree $\mathbf{n^d}$

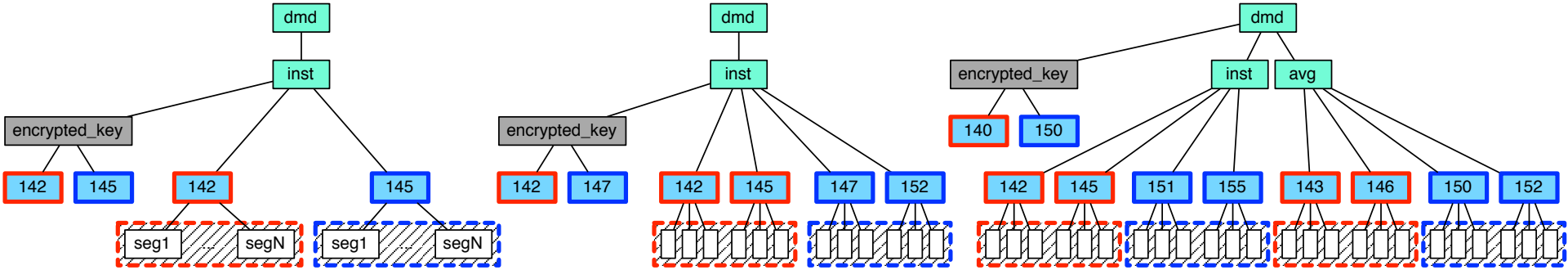
- How to determine a producer's primary read access group?
  - given a $\mathbf{n^d}$, $\mathbf{n^r}$ that shares the longest "prefix" with $\mathbf{n^d}$
  - for producer (in the example)
    - /bms/**data**/boelter/4805/electrical/xfmr-6
  - the primary group should be
    - /bms/**read**/boelter/4805/electrical
  - rather than
    - /bms/**read**/boelter/4805



Content Encrypt Key $\mathbf{k_e}$

- Name
  - /<data-root>/**data**/<data_node_name>/**encrypted_key**/[timestamp]
  - /bms/**data**/boelter/4805/electrical/xfmr-6/dmd/inst/**encrypted_key**/1423413

- Data & $\mathbf{k_e}$
  - $\mathbf{k_e}$ name is placed in data's EncryptionKeyLocator
  - ideal case: one $\mathbf{k_e}$ for one data unit
    - segments of the same data unit are encrypted using the same $\mathbf{k_e}$
    - timestamp of $\mathbf{k_e}$ should be the same as the one of data unit
  - sequential case: one $\mathbf{k_e}$ for data produced during a certain period
    - beginning of the period: timestamp of $\mathbf{k_e}$
    - end of the period: timestamp of next $\mathbf{k_e}$
  - hierarchical case: one $\mathbf{k_e}$ for a group of data under the same data node during a certain period



(a) Ideal Case (b) Sequential Case (c) Hierarchical Case

Primary Group Encrypt Key ($\mathbf{k_{ge}}$, $\mathbf{k_{gd}}$)

- Group name
  - /<data-root>/**read**/<data_name_space>
  - /bms/**read**/boelter/4805

- Each group encrypt/decrypt key has a timestamp
  - indicate the beginning of the period when the key takes effect
  - also implicitly indicate the end of the effective period of the previous key

- Group encrypt key $\mathbf{k_{ge}}$ (public key)
  - name: /<group_name>/**encryption_key**/[timestamp]
  - content: key bits of $\mathbf{k_{ge}}$
  - signed by group authority key $\mathbf{k_{gs}}$

- Group decrypt key $\mathbf{k_{gd}}$ (private key)
  - published as a copy encrypted using each group member's encryption key
  - name: /<group_encrypt_key_name>/[member_public_key_name]
  - content: EncryptedContent (EncryptionKeyLocator: member's public key name)
  - signed by group signing key $\mathbf{k_{gs}}$ (optional)

- $\mathbf{k_e}$ & $\mathbf{k_{ge}}$
  - a producer's content encrypt key $\mathbf{k_e}$ is encrypted with the encryption key $\mathbf{k_{ge}}$ of the producer's primary group
  - the effective period of $\mathbf{k_e}$ must fall into the effective period of $\mathbf{k_{ge}}$.
  - content of $\mathbf{k_e}$: EncryptedContent (EncryptionKeyLocator: primary group's encrypt key name)

Secondary Group Encrypt Key

- Group name
  - no restriction, recommend /<data-root>/**group**/<any_group_tag>
  - /bms/**group**/audit
- Each group encrypt/decrypt key has a version
  - indicates the state of group membership
  - once a member is removed, generate a new version of key
  - a member with the access to the key of version N should also have the access to the key of version N-1
- Group encrypt key (public key)
  - name: /<group_name>/**encryption_key**/[version]
  - content: public key bits
  - signed by group authority key
- Group decrypt key (private key)
  - name: /<group_name>/**encryption_key**/[version]/[member_public_key_name]
  - content: encrypted private key (EncryptionKeyLocator: member's public key name)
  - signed by group authority key (optional)
- Key chaining
  - /<group_name>/**encryption_key**/[old_version]/[new_version]
  - when a user is admitted into a group, the user can collect all the previous decrypt keys

Group Authority Key ($\mathbf{k_{gv}}$, $\mathbf{k_{gs}}$)

- Owned by group manager only
- Usage 1: data signing
  - sign group encryption key $\mathbf{k_{ge}}$ (public key)
  - may also sign the encrypted copies of group decryption key $\mathbf{k_{gd}}$ (private key)
- Usage 2: privilege delegation
  - signing the authority key of a primary group for a sub-namespace
    - /bms/read/boelter can create a sub primary group /bms/read/boelter/4805
    - sub primary group has less privilege
      - members of /bms/**read**/boelter/4805 cannot access data under /bms/**data**/boelter/4809 which is accessible to members of /bms/**read**/boelter
  - the parent primary group still retain the access of its child group through "reverse-adding"
    - child group should add parent group as a member (encrypt child group's decrypt key with parent group's encrypt key)
    - if child group fails to do so, parent group can revoke the certificate of child's authority key
  - optimization: child group may "reverse-add" all its ancestors

Primary Group Delegation Example

- One (say A_1) owns the root of auth sub-tree, e.g., /bms/read
  - A_1 has the private key of authority key of group /bms/read
    - /bms/read/KEY/%01%ff/%01
  - A_1 creates the group encryption public key with timestamp 142000
    - /bms/read/encryption_key/142000

- Add group member
  - one (say U_1) requests the membership of group /bms/read
  - A_1 verifies the eligibility of U_1 (external verification)
  - A_1 publishes an encrypted group decryption key
    - /bms/read/encryption_key/142000/[U_1 public key name]

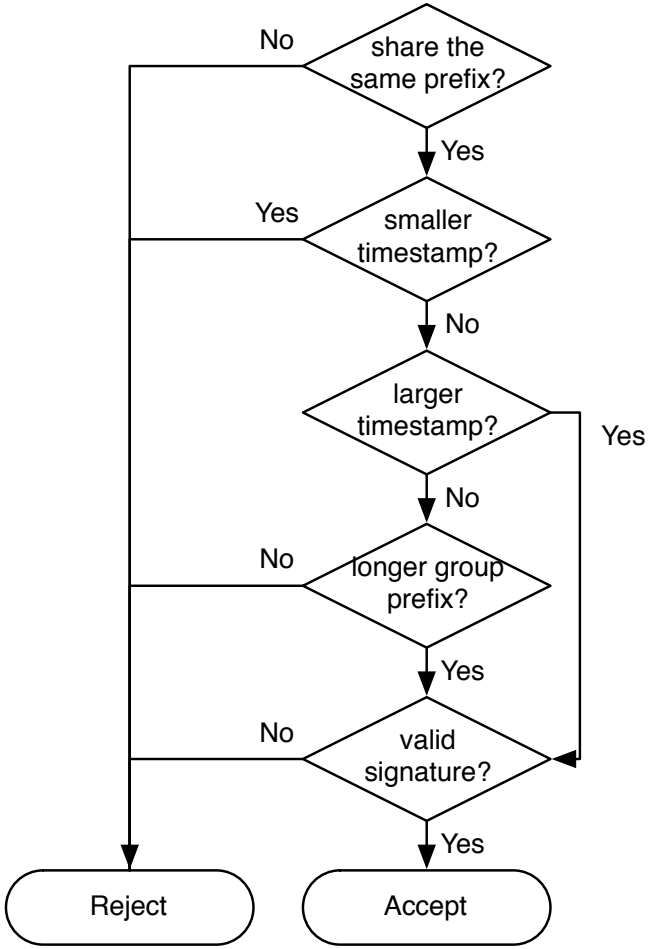
- Create a sub group
  - one (say A_2) requests a sub group /bms/read/boelter
    - A2 create a key /bms/read/boelter/KEY/%c0%9d
  - A_1 verifies the eligibility of A_2 (external verification)
  - A_1 signs the group authority key for /bms/read/boelter
  - A_2 creates its own group encryption public key with timestamp 1423000
    - /bms/read/boelter/encryption_key/142300
  - A_2 adds its parent group (/bms/read) as its group member (**reverse-adding**)
    - publishing its group decryption key encrypted using /bms/read encryption key
    - /bms/read/boelter/encryption_key/142300/[/bms/read/encryption_key/142000]
  - so member of /bms/read have all the access that member of /bms/read/boelter has
  - If A_2 failed to do so, A_1 can revoke A_2's group authority public key

Decrypt Key Change

- When?
  - a member is removed from a group,
    - the group manager should generate a new group encryption/decryption key pair
    - the new key pair should have
      - a larger timestamp (for primary group)
      - a larger version (+1) (for secondary group)
    - the new key pair is encrypted using the public key of remaining members
      - removed member loses the access
    - for secondary group, the old key is also encrypted with the new key
- Who is affected?
  - anyone who use the corresponding encrypt key
    - groups to which the decrypt key owner belongs to
    - data producers if its primary group's encrypt key is changed
- How to detect? discussed later
- What to do?
  - affected data producer must update its content encryption key
  - it is up to affected group manager to update the its own encryption/decryption key pair

How to detect encrypt key change?

- Approach 1: proactively notification
  - each group should know its covered producers
  - send an interest with its latest group encryption key encoded
  - producer verifies the encryption key
    - verification logic →
  - producer reply to the interest with its current group encryption key name



How to detect encrypt key change?

- Approach 2: proactively probe
  - producer subscribe following changes on its corresponding group
    - primary group encryption key change
    - potential primary group changes
      - new primary group added
      - current primary group removed
  - Apply the same verification logic as Approach 1

Producer <-> Primary Group (active mode)

- Assume
  - each primary group has a management process running all the time
- A producer sends interests to retrieve its primary group's encryption key
  - primary group resolution: find the group which has the longest prefix of the producer
- A primary group publishes its delegation info
  - /<primary_group_name>/DelegationInfo/[version]
  - a list of delegate name spaces (sub primary groups)
  - a producer starts from fetching the delegation info of root primary group, then recursively find its corresponding primary group
- A primary group also publishes its encryption key
  - both delegation info and key are placed in a repo
- A producer still keeps outstanding interests to retrieve delegation updates
  - always retrieve the latest version
  - interest may contain an exclude filter

Producer <-> Primary Group (passive mode)

- Assume
  - each primary group has a management process running all the time
  - data producers cannot express interests

- Primary group encryption public key is sent through an interest to a producer
  - primary group management process maintains a managed producer list (configured)
  - each producer register a prefix to receive group public key
    - /<producer_name>/PrimaryGroupKey
  - an interest name is
    - /<producer_name>/PrimaryGroupKey/[primary_group_encrypt_key_cert]
  - interest does not need to be signed
    - producer should be able to verify the certificate of primary group encrypt key

- When the primary group manager generates a new encrypt key, the management process distributes the key to all the managed producers

Primary Group <-> Secondary Group

- A secondary group key is sent to primary group through interests
  - primary group registers a prefix:
    - /<primary_group_name>/SecondaryGroupKey
  - an interest name is
    - /<primary_group_name>/SecondaryGroupKey/[secondary_group_encrypt_key_cert]
  - interest does not need to be signed
    - primary group should be able to validate the secondary group's key
    - mapping from secondary to primary group is defined in a trust schema

- A secondary group does not require an online process
  - secondary group is managed by user
    - primary group requires an online process which is managed automatically
  - secondary group manager sends its group encryption key (in terms of interest) to its related primary group management processes
  - secondary group manager publish its group decryption key (encrypted using each member's public key) in a repo

- A primary group process, when receiving an updates of a member's encryption key, create a new group encryption key
  - notify related producer (either through interests or simply publish the encryption key)
  - publish its decryption key (encrypted using each secondary group's encrypt key) in a repo