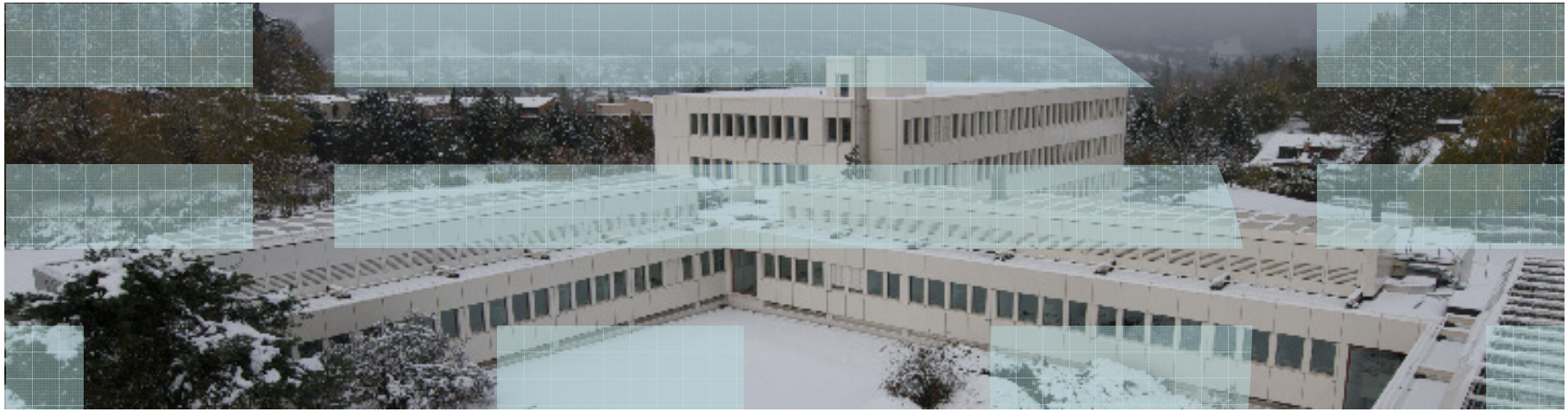
Robert Haas, Marko Vukolic (IBM)

7 April 2010

Access Control in KMIPv1.1



Summary of Changes

- Changes wrt. the last set of slides in red
- 2 additional role permissions related to creation/registration using templates
- Consolidation of Object Permissions
  archive + modify_attributes + rekey → *operate*
- A new Managed Object Attribute: Owner Role ID
- ***unwrap*** Object Permission added
- Other, minor comments from the reflector included

Some comments still unresolved (see Pending, unresolved comments slide)

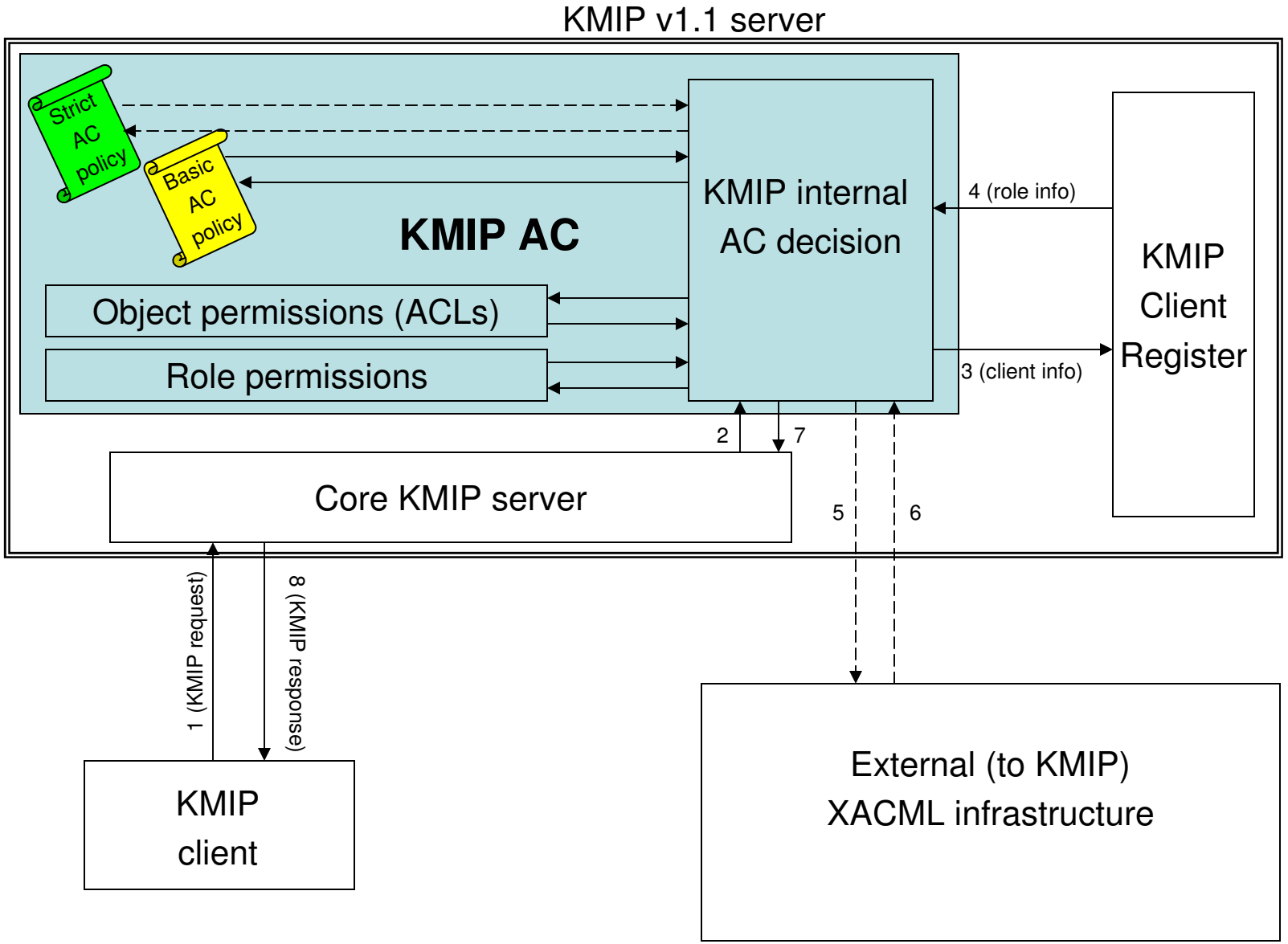
Proposal for AC in KMIP v1.1

- New KMIP ***roles***
  - New KMIP Objects
  - different from Managed Objects (e.g., different attributes)

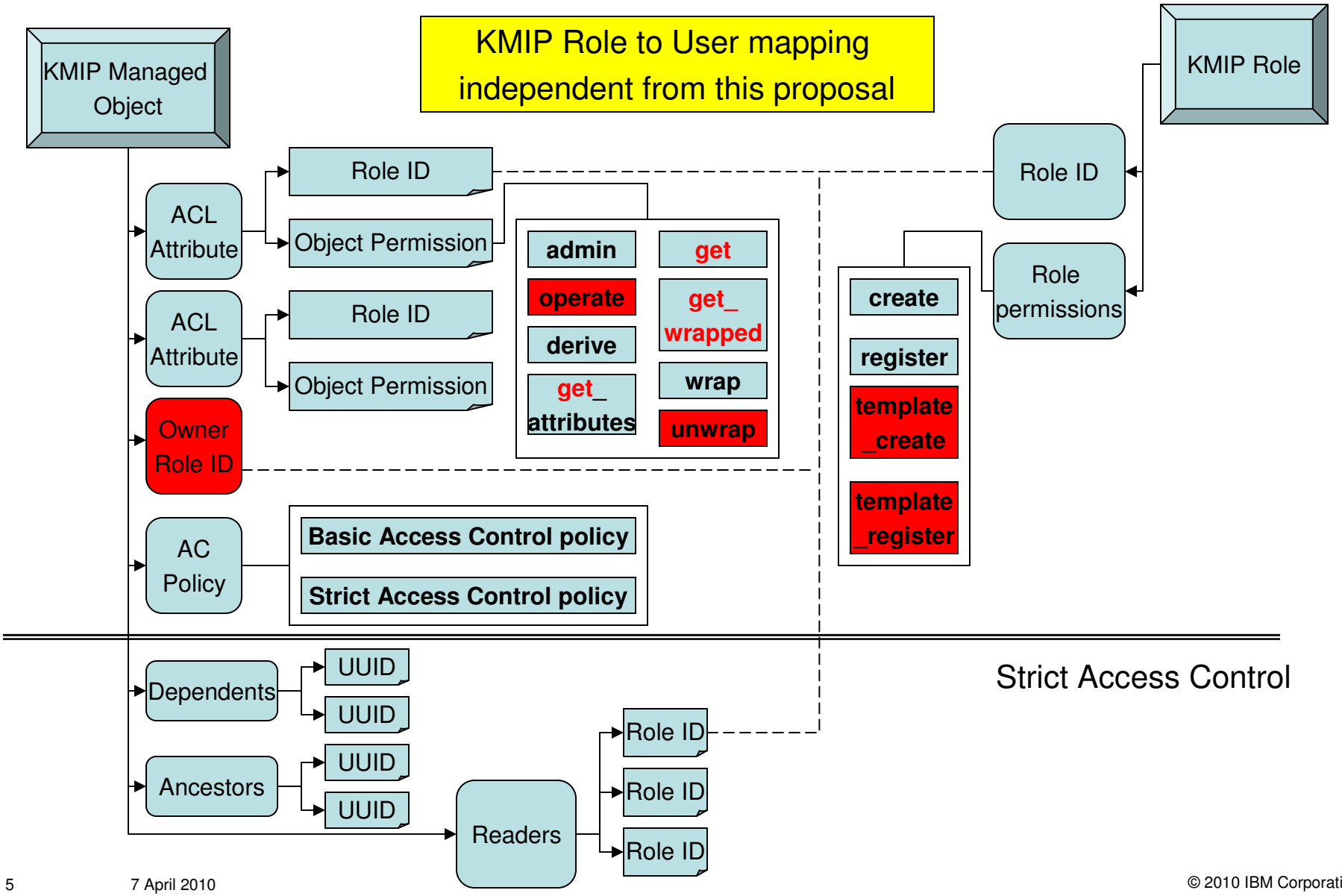
- New ACL attribute
  - A new attribute of a Managed Object
  - Per object (role, permission) pairs

- 2 Access Control policies
  - Basic AC policy
    - considers only ACL attribute of a Managed Object during policy enforcement
  - Strict AC policy (OPTIONAL)
    - considers ACL attributes of the potentially cryptographically related Managed Objects during policy enforcement

KMIP AC



KMIP AC related Objects and Attributes



KMIP roles

- 2 special roles
  - One special role *"any"*
    - carried over from KMIP v1
    - any KMIP client
  - Additional object-specific role "**owner"**
    - Related to the new *Owner Role ID* Attribute (detailed in slide 11)
      - By default maps to *creator* from KMIPv1 (detailed in slide 13)
    - creator NOT taken over "as is" from v1
      - (cf. problems with scenarios where creator rights would have to be revoked)

- Extendible set of roles
  - Additional roles ***(custom roles)*** can be defined in a given system

- Mapping between ***roles*** and users independent from this proposal
  - To be dealt with in Client Registration action item
  - The rest of the proposal assumes each client has a designated **primary role**

KMIP role

- KMIP Role Object has following attributes
  - Role ID (text string, e.g., “any”, “owner”)
  - Role permissions
  - Role description

- Role permissions permit KMIP operations that do not refer to existing objects
  - *Create*, *Create Key Pair* and *Register*

- KMIP Operations on Roles TBD

Proposed Role Permissions

Role permission	KMIP Operation(s) permitted
create	Create, Create Key Pair
register	Register
template_create	Create, Create Key Pair (attributes can only be propagated from a Template, cannot override using client-specified attributes)
template_register	Register (attributes can only be propagated from a Template, cannot override using client-specified attributes)

- ***create*** (resp., ***register***) permission implies ***template_create*** (resp., ***register_template***)
- Example: to create a key using template T, a role R would be required to have:
  template_create role permission AND
  get_attributes object permission on T (see also slide 12)
- Other operations to be handled via Object permissions within ACLs

ACL

- ACL is a new KMIP v1.1 multivalued attribute of a Managed Object
- Each instance of the ACL attribute consists of:
  - Role ID
  - Object Permission

ACL

Managed Object Attribute	Encoding	REQUIRED
ACL	Structure	
Role ID	Text String	YES
Object Permission	Enumeration	YES

SHALL always have a value	Yes
Initially set by	Server
Modifiable by server	Yes
Modifiable by client	Yes
Deletable by client	Yes
Multiple instances permitted	Yes
When implicitly set	Create, Create Key Pair, Register, Derive Key, Re-key, Certify, Re-Certify
Applies to Object Types	All Objects

Object Permission	
Name	Value
admin	00000001
operate	00000002
derive	00000003
get_attributes	00000004
get	00000005
get_wrapped	00000006
wrap	00000007
unwrap	00000008
Extensions	8XXXXXXX

Owner Role ID

Managed Object Attribute	Encoding	REQUIRED
Owner Role ID	Text String	YES

SHALL always have a value	Yes
Initially set by	Server or Client
Modifiable by server	Yes
Modifiable by client	Yes*
Deletable by client	No
Multiple instances permitted	No
When implicitly set	Create, Create Key Pair, Register, Derive Key, Re-key, Certify, Re-Certify
Applies to Object Types	All Objects

*Not if the Strict Access Control policy applies to the object

Proposed Object Permissions

Object Permission	KMIP Operation(s) permitted	Applies to Object Type(s)
admin	All KMIP ops (must have for Destroy and Add/Modify/Delete ACL Attr)	All
operate	Re-Key, Add/Modify/Delete Attribute*, Activate, Revoke, Archive, Recover, Certify, Re-Certify (* except the ACL Attribute (***admin*** permission is needed))	All applicable types
derive	Derive	Symmetric Keys, Secret Data
get_attributes	Locate, Check, Get Attributes, Get Attribute List + referencing a Template in Template/Attr structure	All
get	Get object in cleartext + Obtain Lease, Get Usage Allocation	All
get_wrapped	Get of the wrapped object + Obtain Lease, Get Usage Allocation	All
wrap	Referring to wrapping keys in Key Wrapping Data (in Get)	Keys
unwrap	Referring to unwrapping keys in Key Wrapping Data (in Register)	Keys
N/A	Validate	Certificates
N/A	Query, Cancel, Poll	N/A

Backward compatibility with KMIPv1

- If Operation Policy Name is “default” then
  - The Owner Role ID SHALL contain a primary role of a creator (in KMIP v1 speak)
  - ACL SHALL be set as follows:

    - Secret Objects (i.e., Symm. Keys, Private Keys, Split Keys, Secret Data, and Opaque Objects)

    ACL = [(owner, admin)]

    - Certificates

    ACL = [(owner, admin), (any, get_attributes), (any, get), (any, get_wrapped)]

    - Public Keys

    ACL = [(owner, admin), (any, get_attributes), (any, get), (any, get_wrapped), (any, wrap)]]

    - Private template

    ACL = [(owner, admin)]

    - Public template

    ACL = [(any, get); (any, get_attributes)]

- If ACLs are set differently then Operation Policy Name SHALL NOT be “default”

Access Control Policy

A new Managed Object Attribute

Managed Object Attribute	Encoding
Access Control Policy	Enumeration

SHALL always have a value	Yes
Initially set by	Server
Modifiable by server	No
Modifiable by client	No
Deletable by client	No
Multiple instances permitted	No
When implicitly set	Create, Create Key Pair, Register, Derive Key, Re-key
Applies to Object Types	All Objects

Access Control Policy	
Name	Value
Basic Access Control Policy	00000001
Strict Access Control Policy	00000002
Extensions	8XXXXXXX

Basic AC policy

Basic AC policy, roughly, governs by the following rule (details later in this presentation)

- A client in role (ID) **r** is permitted to perform an operation op on managed object **obj** if for the applicable permission **perm**:

  BAUTH(**r**,**perm**,**obj**) == TRUE

BAUTH(***r*,*perm*,*obj***) is defined as:

((*owner*, ***perm***) is in **obj**.ACL) AND (**r** = **obj.**Owner Role ID)

OR

((any, **perm**) is in **obj**.ACL)

OR

((**r**, **perm**) is in **obj**.ACL)

Here we assume **(r,admin)** implies **(r,perm)** for any **perm**

Strict AC Policy (why?)

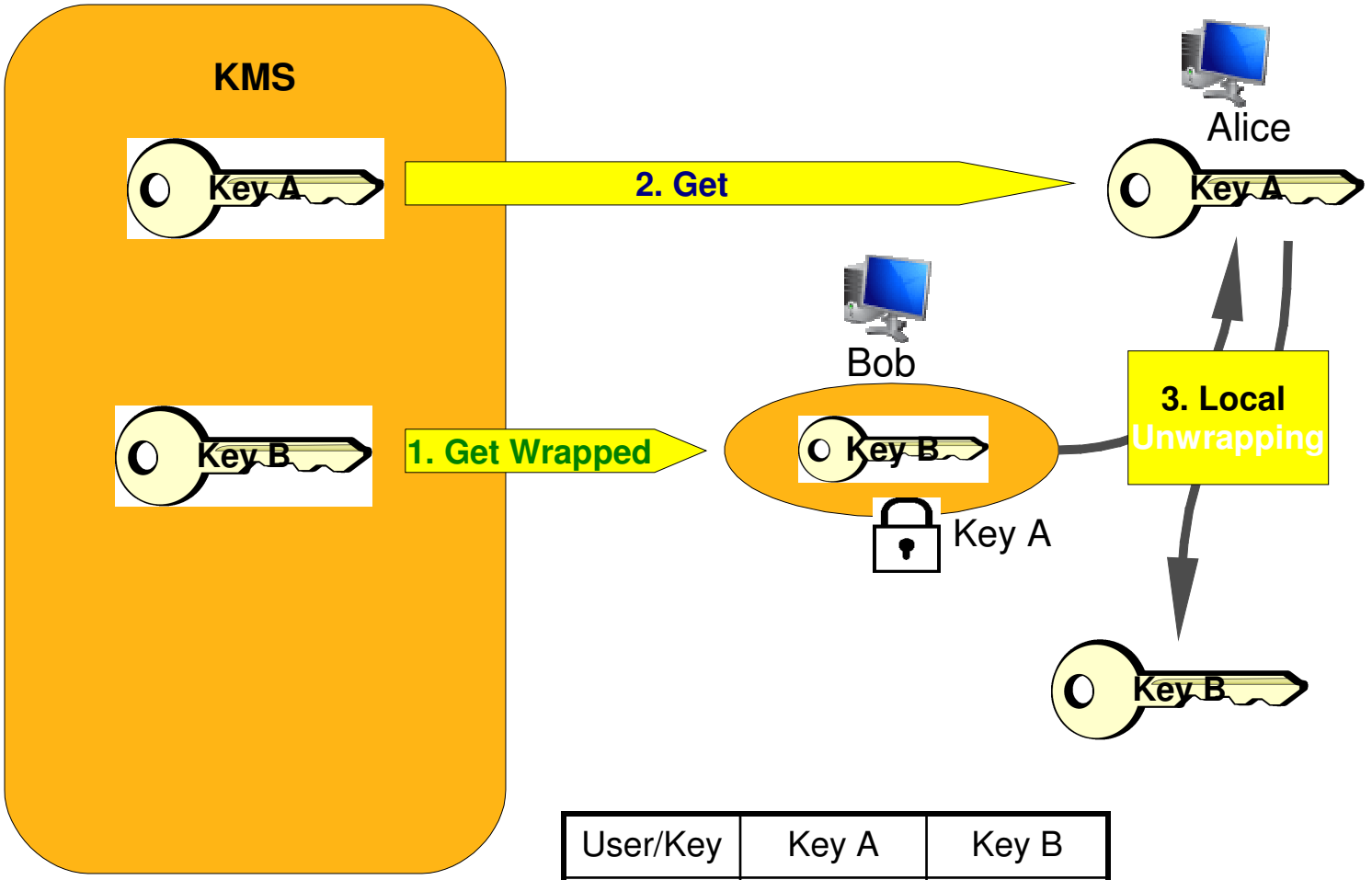
- To protect KMIP against API attacks
- API attacks possible with any AC policy that:
  - allows wrapping and derivation,
  - does not care about crypto dependencies among objects introduced by these.

Example of the API attack (Get Wrapped)

(Note: presented on the previous KMIP f2f, this and next slide can be skipped)

- Use case
  - Tape drive encryption
- Setup
  - Key B is used to encrypt data written to/read from a tape drive
  - Key B wrapped with key A is stored on a tape drive (e.g., by user Bob)
  - Initially no user (except administrator) has a permissions to read the key material of Key A
  - Administrator gives the permission to user Alice to read the key material on key A, not knowing that it was used to wrap key B
  - Administrator does not intend to allow Alice the permission to read key B
- Attack
  - Alice reads the key material of Key A from KMIP server, gets the wrapped Key B, unwraps it and obtains the key material of Key B

Example of the API attack (Get Wrapped)



User/Key	Key A	Key B
Alice	read	-
Bob	wrap	export

Strict AC Policy

- Prevents API attacks in KMIP
- SAC enforcement can be made optional to support legacy application
- An optional KMIPv1.1 profile related to SAC support can be defined

Attributes needed for SAC

- Dependents
- Ancestors
- Readers

This attributes are explained in the following slides

Dependents

- A multivalued attribute, contains uuids of objects wrapped using / derived from this key
- A key is always dependent on itself

Managed Object Attribute	Encoding
Dependents	Text String

SHALL always have a value	Yes*
Initially set by	Server
Modifiable by server	Yes
Modifiable by client	No
Deletable by client	No
Multiple instances permitted	Yes
When implicitly set**	Create, Create Key Pair, Register, Derive Key, Re-key
Applies to Object Types	Keys

*Only if the value of Access Control Policy attr is “Strict Access Control Policy”

** Since a key is always dependent on itself

Ancestors

- A multivalued attribute, contains uuids of objects this object is dependent of
- A key is always its own ancestor

Managed Object Attribute	Encoding
Ancestors	Text String

SHALL always have a value	Yes*
Initially set by	Server
Modifiable by server	Yes
Modifiable by client	No
Deletable by client	No
Multiple instances permitted	Yes
When implicitly set**	Create, Create Key Pair, Register, Derive Key, Re-key
Applies to Object Types	Keys

*Only if the value of Access Control Policy attr is “Strict Access Control Policy”

** Since a key is always its own ancestor

Readers

- contains a set of Role IDs (except *creator*)
- Denotes roles who have (or might have) obtained the cleartext of the given object **obj** because:
  - they have either executed a Get operation on **obj**
  - executed a Get operation for another object **obj'** and **obj** is in **obj'**.Dependents

Managed Object Attribute	Encoding	Description
Readers	Text String	Role ID

SHALL always have a value	No
Initially set by	Server
Modifiable by server	Yes
Modifiable by client	No
Deletable by client	No
Multiple instances permitted	Yes
When implicitly set	Create, Create Key Pair, Register, Derive Key, Re-key
Applies to Object Types	Symmetric Keys, Private Keys, Split keys, Secret Data, Opaque Objects

Authorization of operations (for both BAC and SAC)

Create, Create Key Pair, Register

- For both BAC and SAC:
  - A role **r** is permitted to perform Create or Create Key Pair (resp., Register) if:
    - Role **r** has **create (**resp., **register)** role permission
  - Additionally, for Register of a wrapped object:
    - **wrapkey** denotes the UUID of the Encryption Key in Key Wrapping Data
    - **signkey** denotes the UUID of the MAC/Signature Key in Key Wrapping Data
    - Condition:
      (BAUTH(**r**,**unwrap**,**signkey**) == TRUE) AND (BAUTH(**r**,**unwrap,wrapkey)** == TRUE) AND
      wrapkey. Cryptographic_Usage_Mask has Unwrap Key flag set
      signkey. Cryptographic_Usage_Mask has MAC Verify or Verify flag set

- In addition, for Register with SAC:
  + there is no object already present at the server whose Digest attribute (SHA-256) is equal to the one of the registered obj

Re-Key

- Condition (for both BAC and SAC)::
  BAUTH(**r**,**operate**,**obj**) == TRUE

Derive Key

- Only if a registered object is used for derivation (e.g., not in case of HASH key derivation)
- Condition (for both BAC and SAC):
  BAUTH(**r**,**derive**,**obj**) == TRUE
  AND
  obj. Cryptographic_Usage_Mask has Derive Key flag set
- Effects:
  - BAC
    - AC Policy Attr of the new object **new** is set to “Basic Access Control Policy”
  - SAC
    - AC Policy Attr of the new object **new** is set to “Strict Access Control Policy”
    - For every object **o** in **obj**.Ancestors: add **new.**UUID to **o.**Dependents
    - Set **new.**Ancestors to **obj.**Ancestors + **new.**UUID
    - **new.**Readers ← **obj**.Readers

Certify, Re-Certify

- Condition (for both BAC and SAC)::
  BAUTH(**r**,**operate**,**obj**) == TRUE

- NB: The above condition is in addition and orthogonal to AC mechanisms outside KMIP,
  - E.g., proof of possession in PKCS#10 request

Locate

- Condition (for both BAC and SAC):
  - BAUTH(**r**,**get_attributes**,**o**) must be true for every object **o** whose UUID is returned in the reply from Locate

Check, Get Attributes, Get Attribute List

- Condition (for both BAC and SAC):
  - BAUTH(**r**,**get**_**attributes**,**obj**) == TRUE

Get (in cleartext)

- Condition:
  - BAC
    - BAUTH(**r**,**get**,**obj**) == TRUE
  - SAC
    - for every object **o** in **obj.**Dependents: BAUTH(**r**,**get**,**o**) == TRUE
- Effect:
  - BAC
    - None
  - SAC
    - for every object **o** in **obj.**Dependents: Add role **r** to **o.**Readers

Get (wrapped)

- **wrapkey** denotes the UUID of the Encryption Key in Key Wrapping Data
- **signkey** denotes the UUID of the MAC/Signature Key in Key Wrapping Data

- Condition:
  - BAC:

    (BAUTH(**r**,**get_wrapped**,**obj**) == TRUE) AND (BAUTH(**r**,**wrap,wrapkey)** == TRUE) AND
    (BAUTH(**r**,**wrap,signkey)** == TRUE) AND

    wrapkey. Cryptographic_Usage_Mask has Wrap Key flag set
    signkey. Cryptographic_Usage_Mask has MAC Generate or Sign flag set

  - SAC (in conjunction to BAC condition):

    wrapkey. Cryptographic_Usage_Mask does not have Sign, Verify, Encrypt, Decrypt, Derive Key flags set AND
    For every role **r** in **wrapkey.**Readers, for every object **o** in **obj**.Dependents: BAUTH(**r, get, o**) AND
    wrapkey. Access_Control_Policy = “Strict Access Control Policy” AND
    wrapkey.UUID not in **obj**.Dependents

- Effects:
  - BAC: None
  - SAC:
    - for every object **o** in **obj.**Dependents: **o**.Readers ← **o**.Readers ∪ **wrapkey**.Readers
    - Add **obj**.UUID to **wrapkey**.Dependents
    - Add **wrapkey**.UUID to **obj.**Ancestors

Add, Modify, Delete Attribute, Activate, Revoke

- Condition
  - BAC: BAUTH(**r**,**operate**,**obj**) == TRUE

    Exception: when ACL Attribute is changed:

    BAUTH(**r**,**admin**,**obj**) == TRUE
  - SAC (in conjunction with BAC condition, when ACL Attribute is changed):

    Let ACL' be the new ACL attribute:

    For every role **r** ≠ *owner* such that **(r,get)** in ACL'**:**

    for every object **o** in **obj.**Dependents: BAUTH(**r**, **get**, **o**) == TRUE

    AND

    If (*owner*,**get)** in ACL'**:**

    for every **o** in **obj.**Dependents: BAUTH**(obj**.Owner Role ID, **get**, **o**) == TRUE

    (i.e., if a **get** permission is given on some object, **get** must be permitted on all dependents)
- Effects
  - None

Obtain Lease, Get Usage Allocation

- Condition (for both BAC and SAC):
  BAUTH(**r**,**get**,**obj**) == TRUE OR BAUTH(**r**,**get_wrapped**,**obj**) == TRUE

Destroy

- Condition (for both BAC and SAC):
  BAUTH(**r**,**admin**,**obj**) == TRUE

Archive, Recover

- Condition (for both BAC and SAC):
  BAUTH(**r**,**operate**,**obj**) == TRUE

Validate, Query, Cancel, Poll

- Always allowed
- NB: Query, Cancel and Poll do not apply to any specific Managed Object

Pending, unresolved comments

1) Comment from Marcus:

> Export
> ------
> Do we want to restrict which keys can be used to wrap another key – for example by adding a list of UUIDs to
> either export of wrap permissions.
>
> The list on export would specify the keys that are permitted to wrap this key. The list on export would specify the keys
> that this key could be used to wrap.

Marko: Would you really want to segment export/wrap permissions or would a simple introduction of additional Link Types like 'WrappableBy' and 'CanWrap' be sufficient?