Registration-Based Encryption in the Plain Model

Jesko Dujmovic
CISPA and Saarland University

Giulio Malavolta
CIFRA and Bocconi University

Wei Qi
CIFRA and Bocconi University

Identity Based Encryption [Shamir86, BF01]

Identity Based Encryption [Shamir86, BF01]

Central Authority

Identity Based Encryption [Shamir86, BF01]

Master Secret Key
(MSK)

Central Authority

Identity Based Encryption [Shamir86, BF01]

Master Secret Key
(MSK)

Central Authority

Master Public Key
(MPK)

Identity Based Encryption [Shamir86, BF01]

Master Secret Key
(MSK)

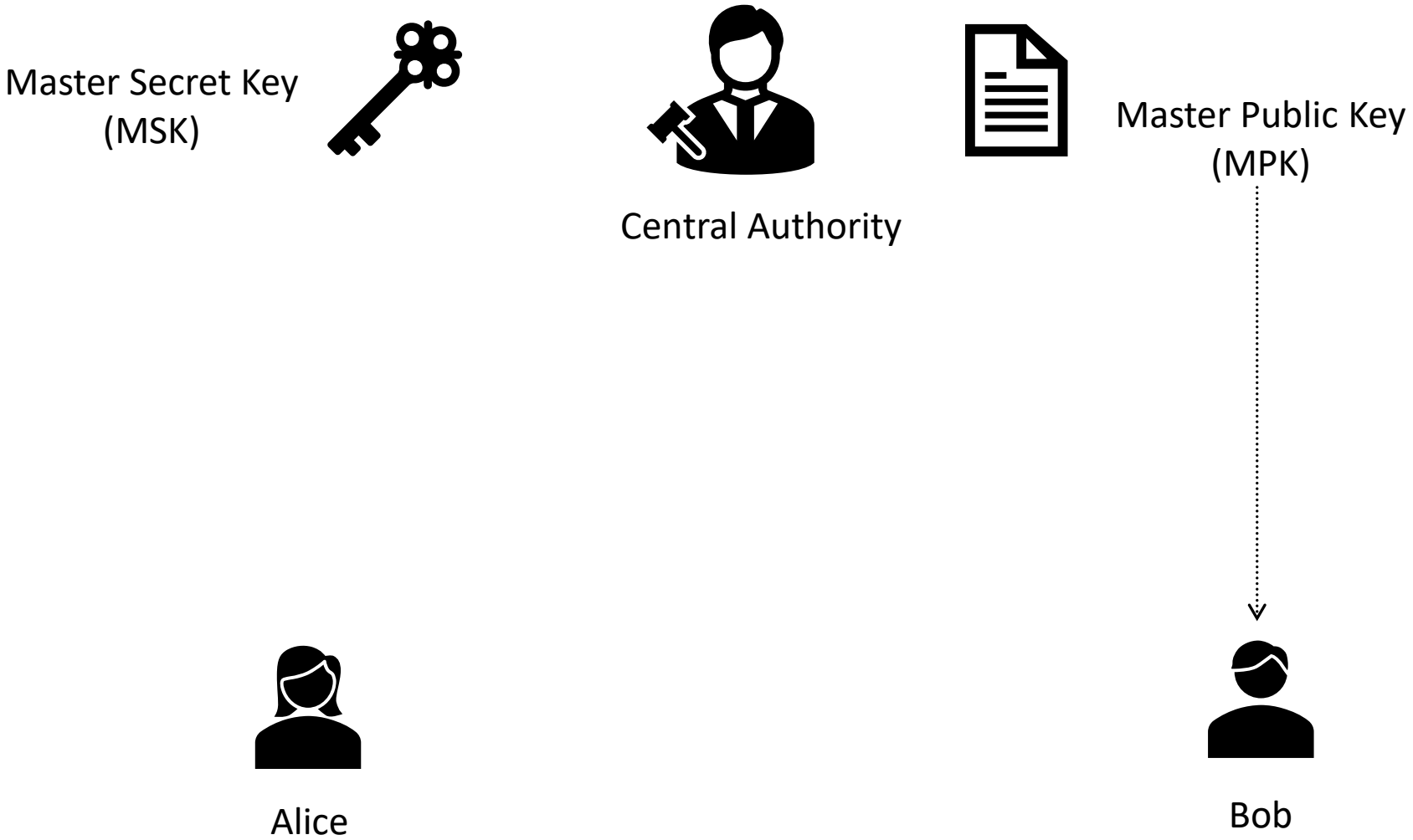
Central Authority

Master Public Key
(MPK)

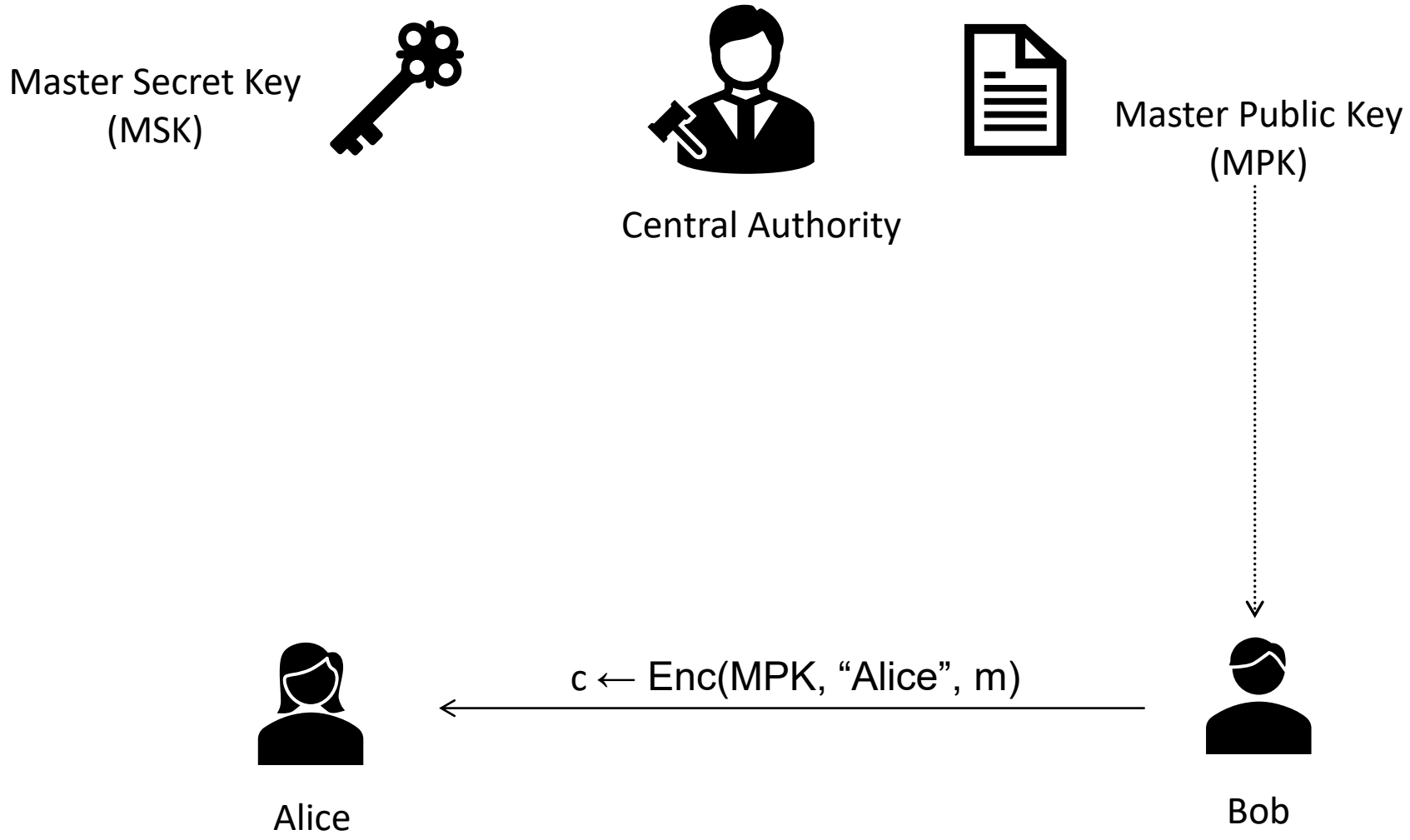
Alice

Bob

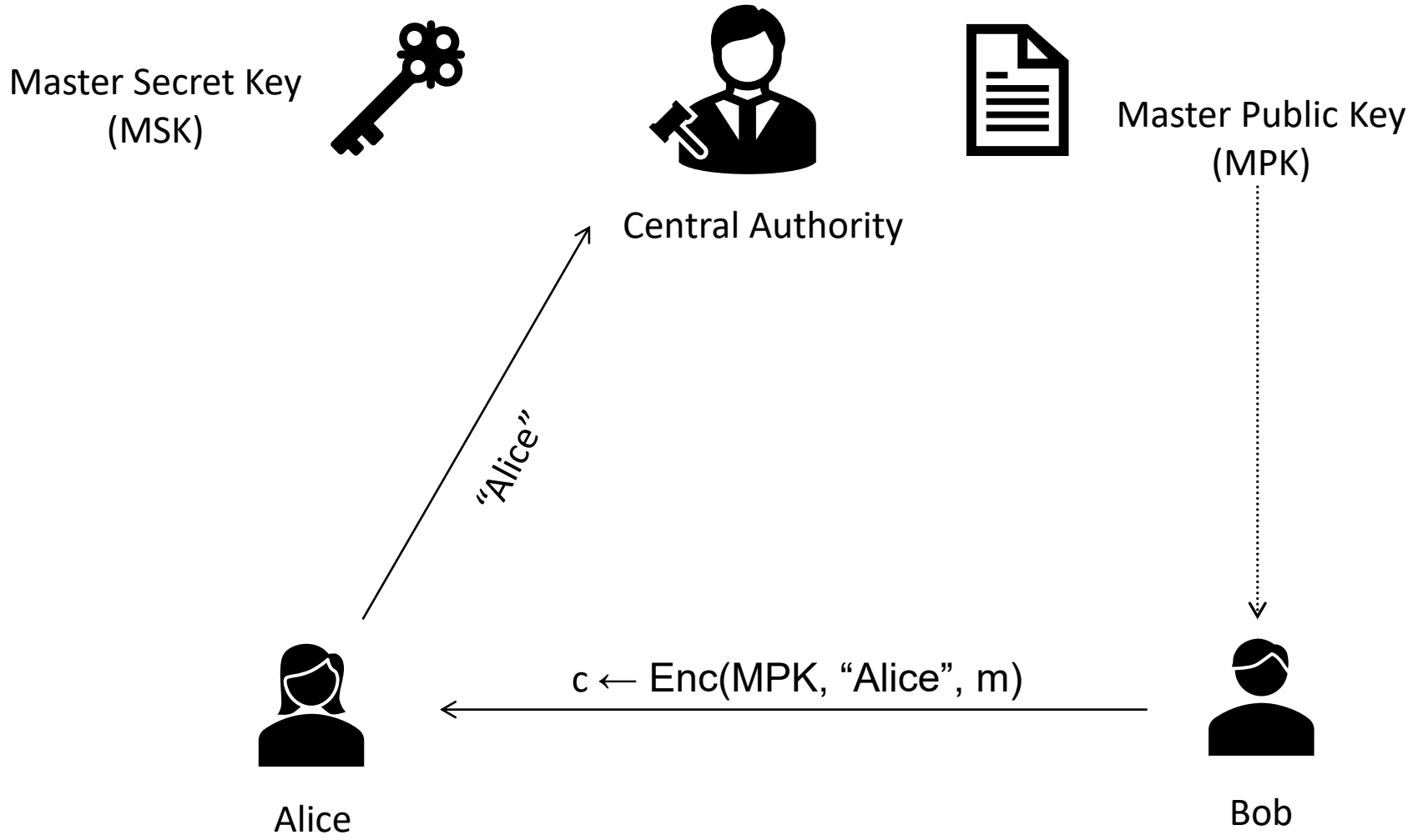
Identity Based Encryption [Shamir86, BF01]



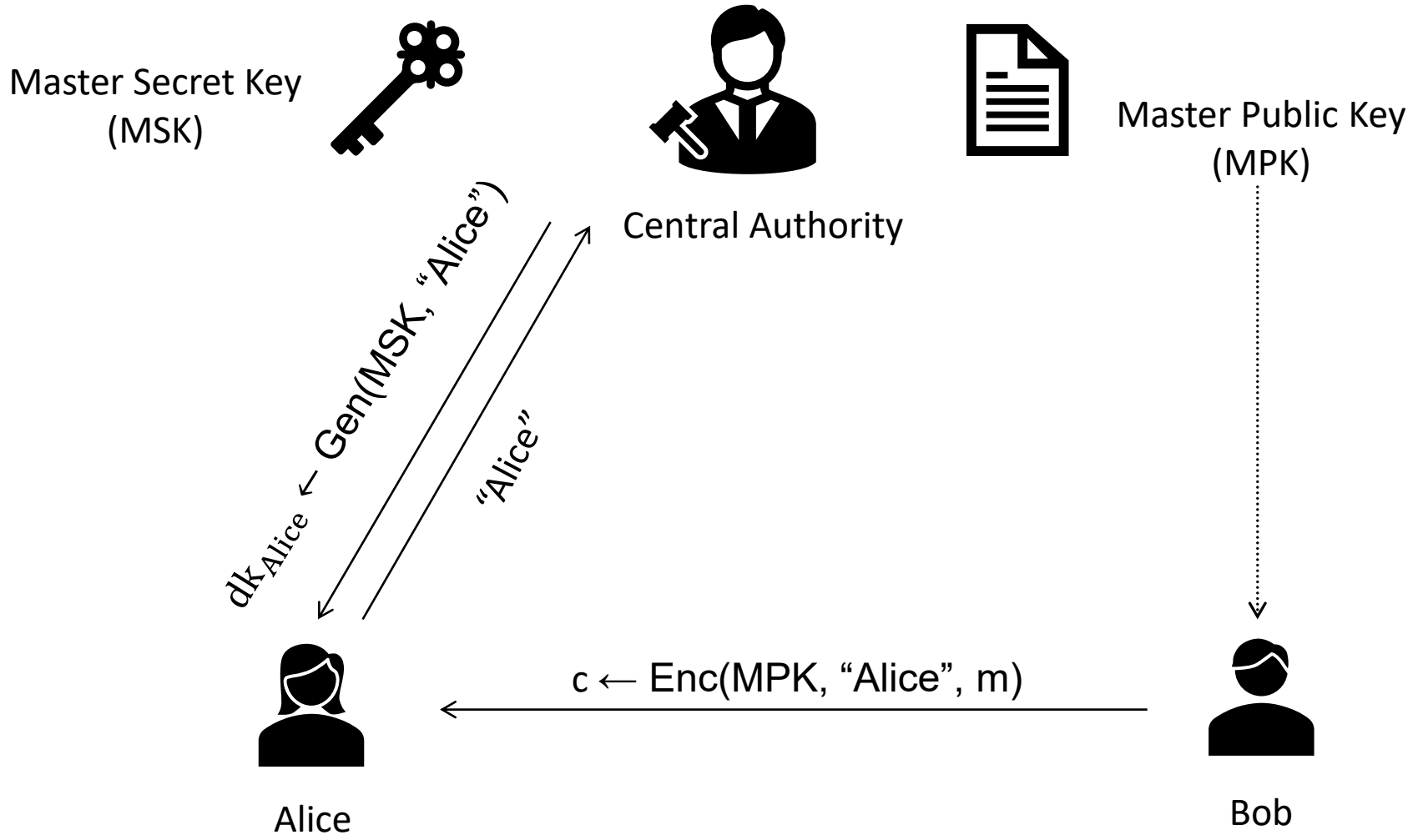
Identity Based Encryption [Shamir86, BF01]



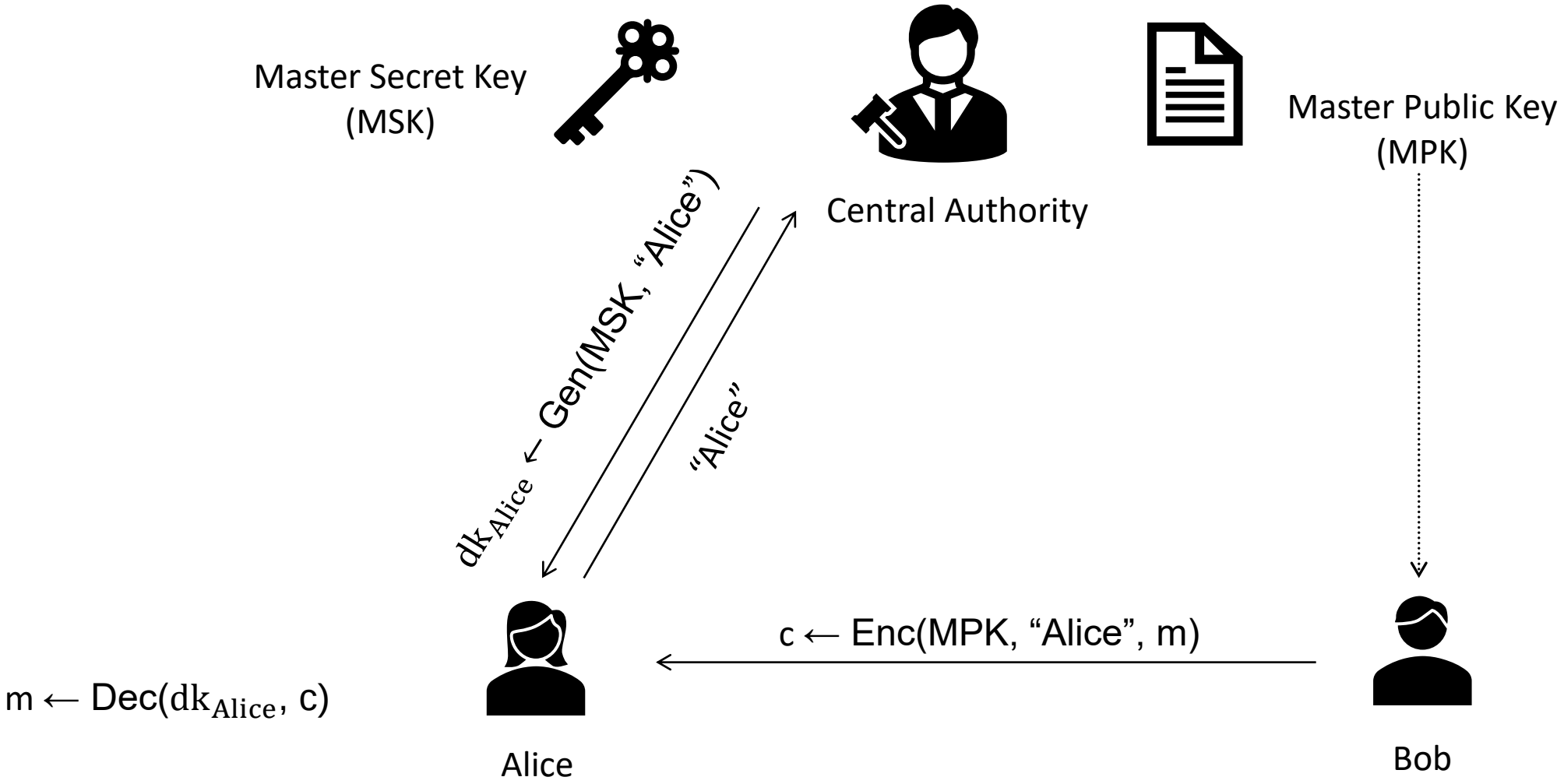
Identity Based Encryption [Shamir86, BF01]



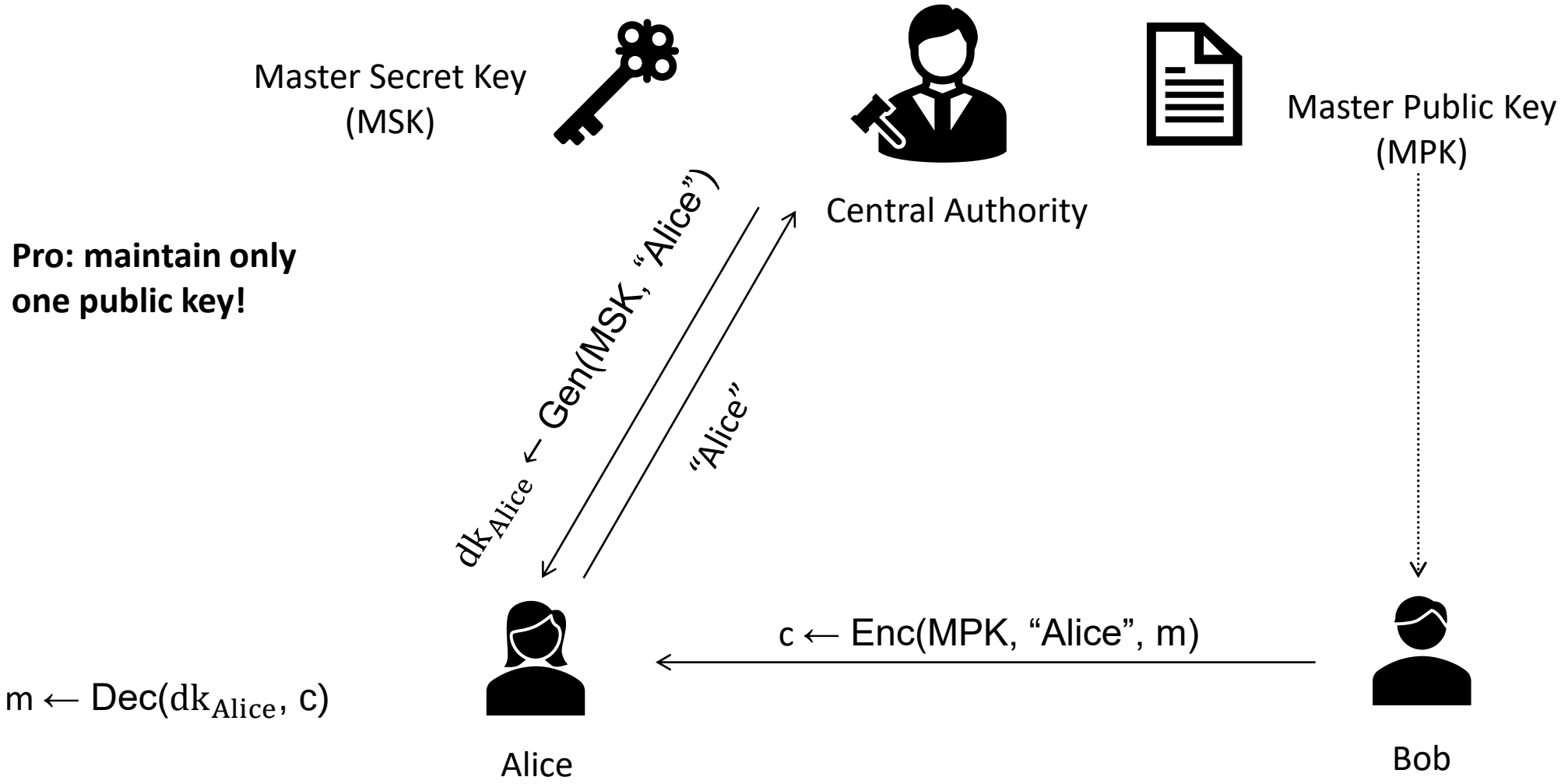
Identity Based Encryption [Shamir86, BF01]



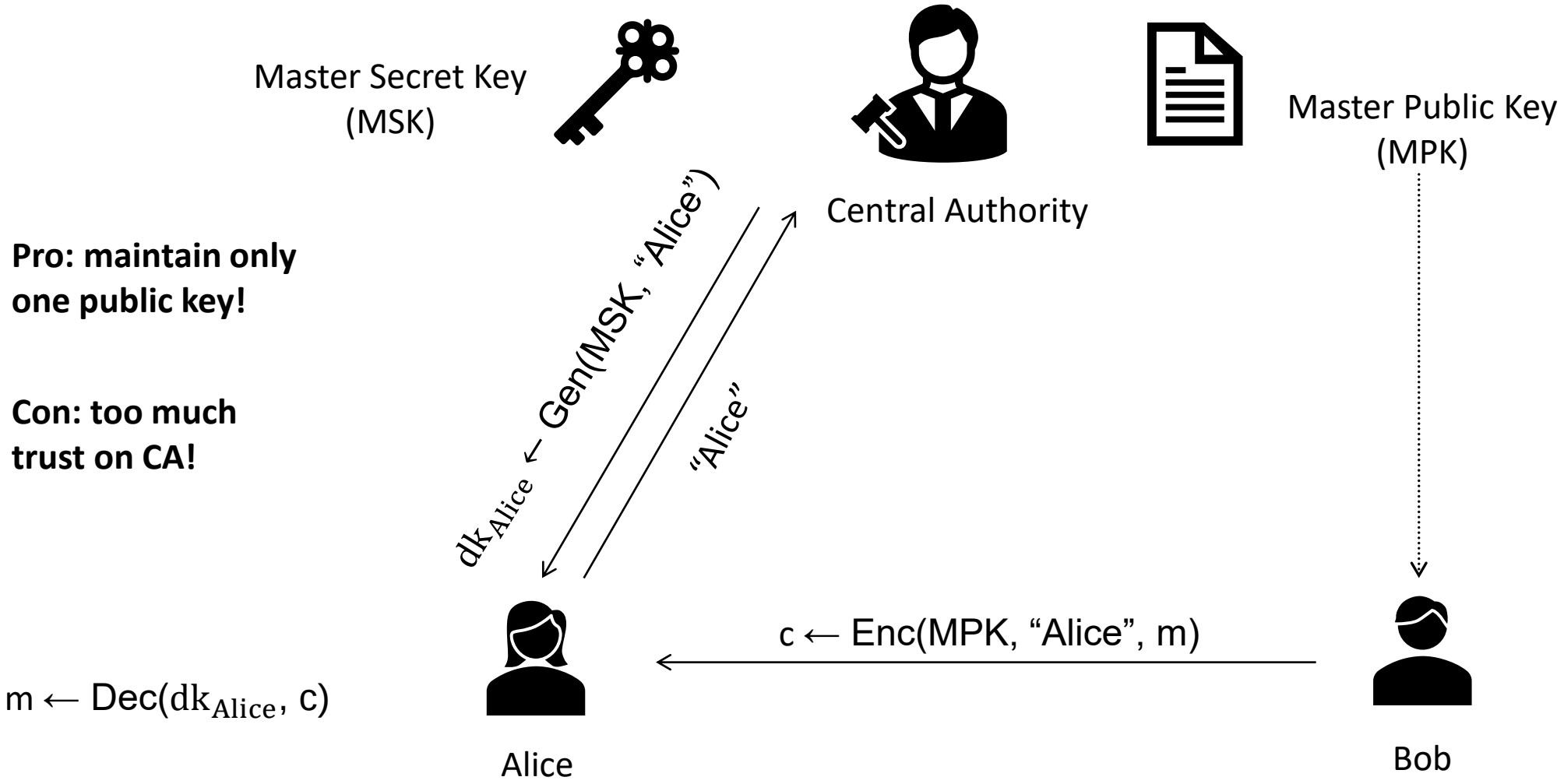
Identity Based Encryption [Shamir86, BF01]



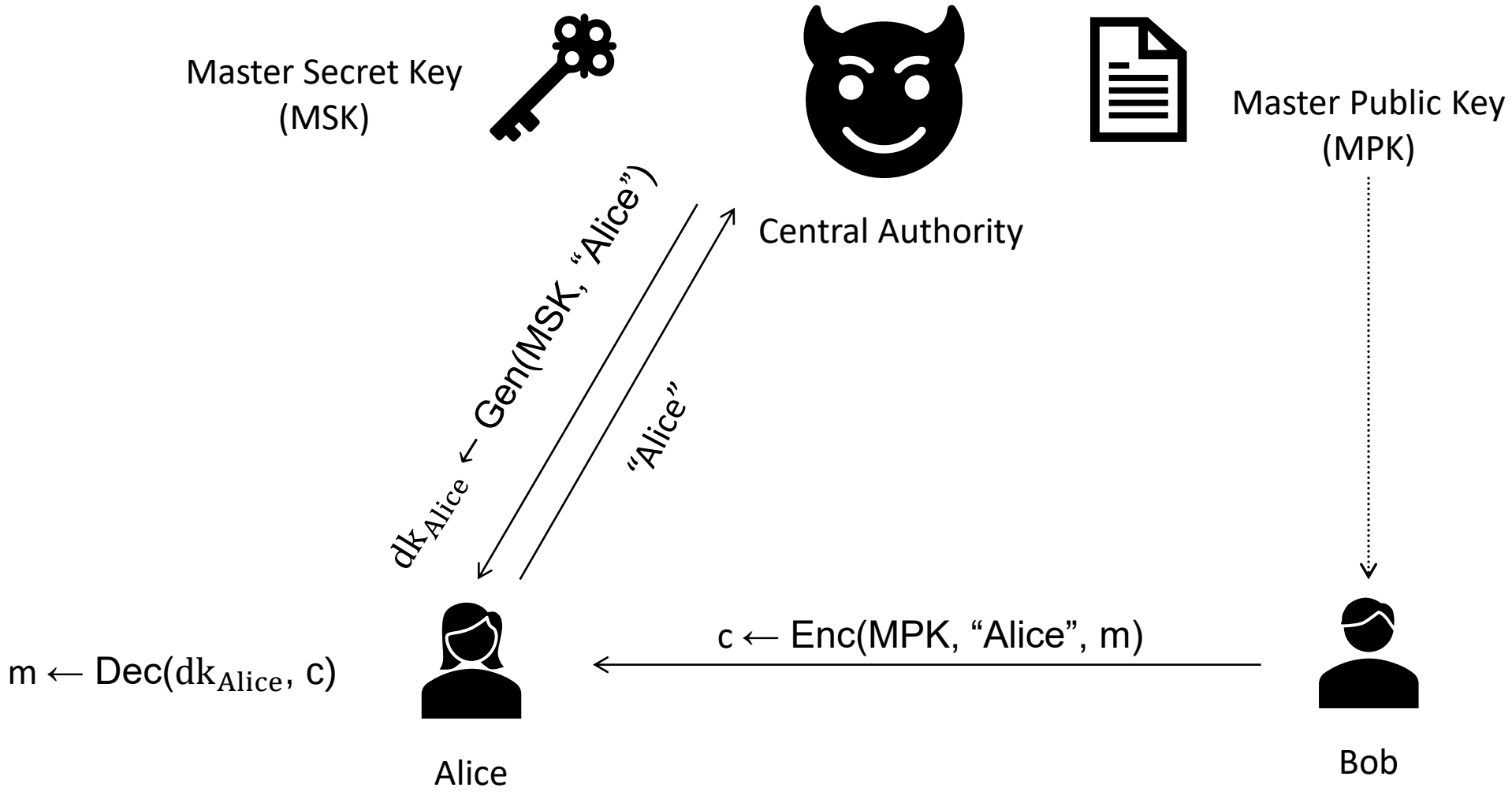
Identity Based Encryption [Shamir86, BF01]



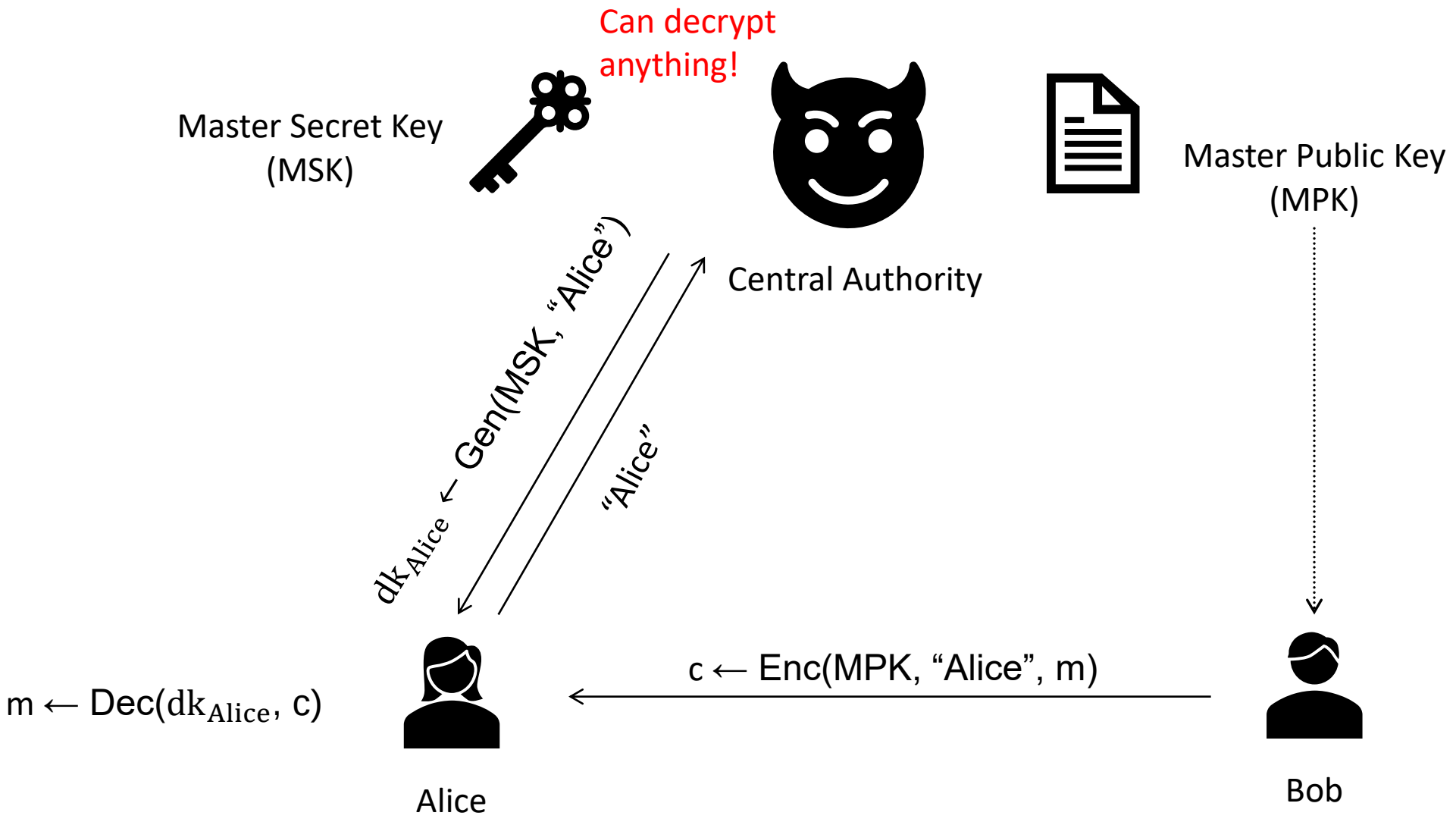
Identity Based Encryption [Shamir86, BF01]



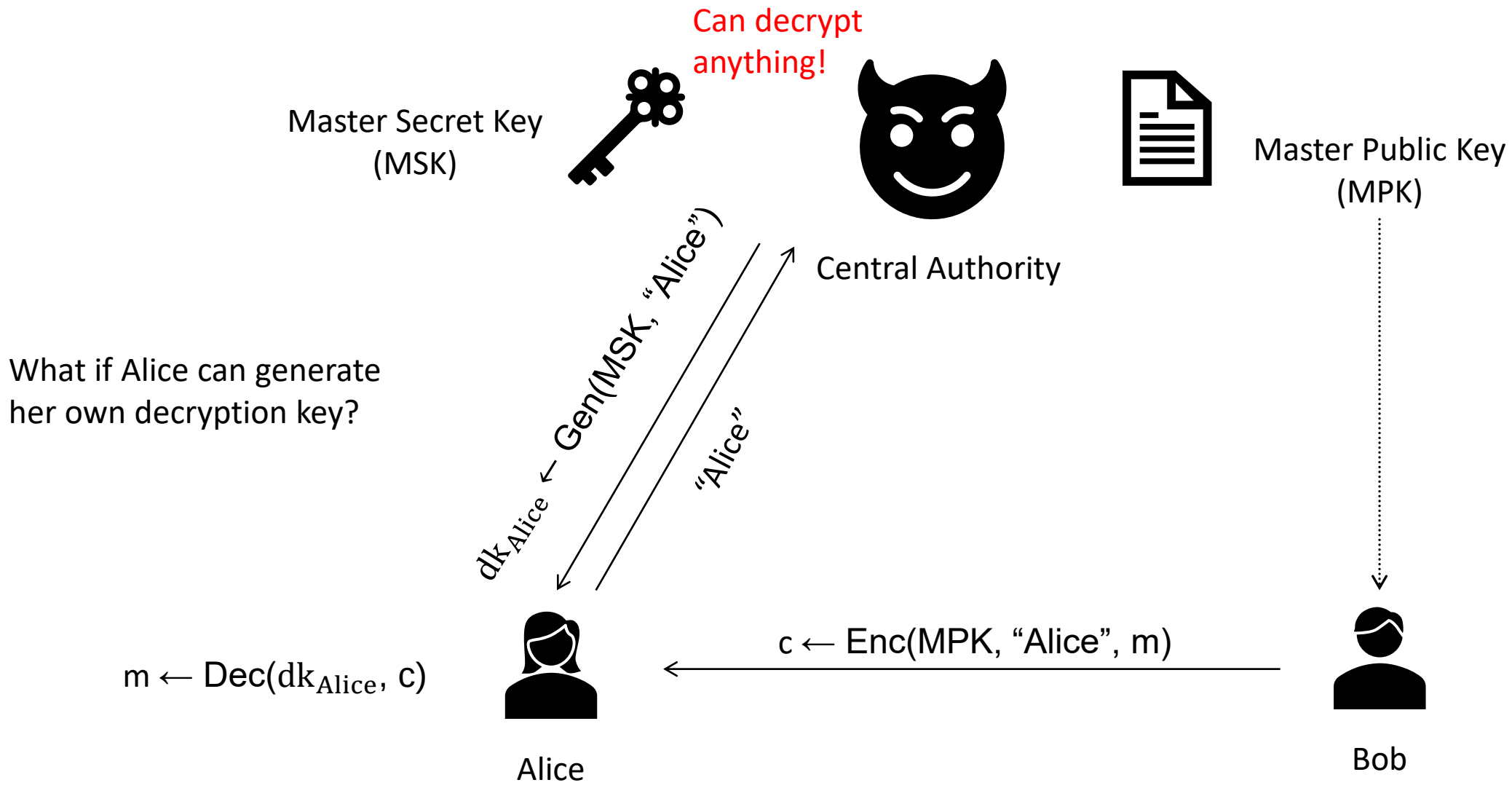
Key Escrow: What if the CA is bad?



Key Escrow: What if the CA is bad?

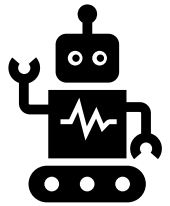


Key Escrow: What if the CA is bad?



Registration based Encryption [GHMR18]

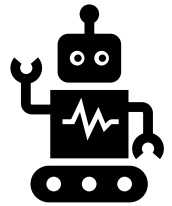
Registration based Encryption [GHMR18]



Key Curator

Registration based Encryption [GHMR18]

No secret state!

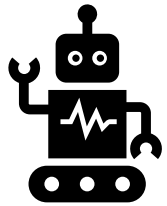


Key Curator

Registration based Encryption [GHMR18]

No secret state!

id_1



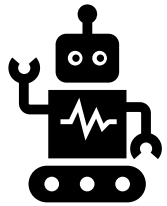
Key Curator

Registration based Encryption [GHMR18]

No secret state!

$(\mathrm{pk}_1, \mathrm{sk}_1) \leftarrow \mathrm{Gen}(1^\lambda)$

id_1



Key Curator

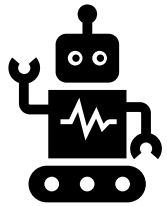
Registration based Encryption [GHMR18]

No secret state!

$(\mathrm{pk}_1, \mathrm{sk}_1) \leftarrow \mathrm{Gen}(1^\lambda)$

id_1

$(\mathrm{id}_1, \mathrm{pk}_1)$



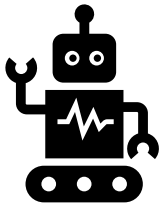
Key Curator

Registration based Encryption [GHMR18]

$(\mathrm{pk}_1, \mathrm{sk}_1) \leftarrow \mathrm{Gen}(1^\lambda)$

id_1

$(\mathrm{id}_1, \mathrm{pk}_1)$

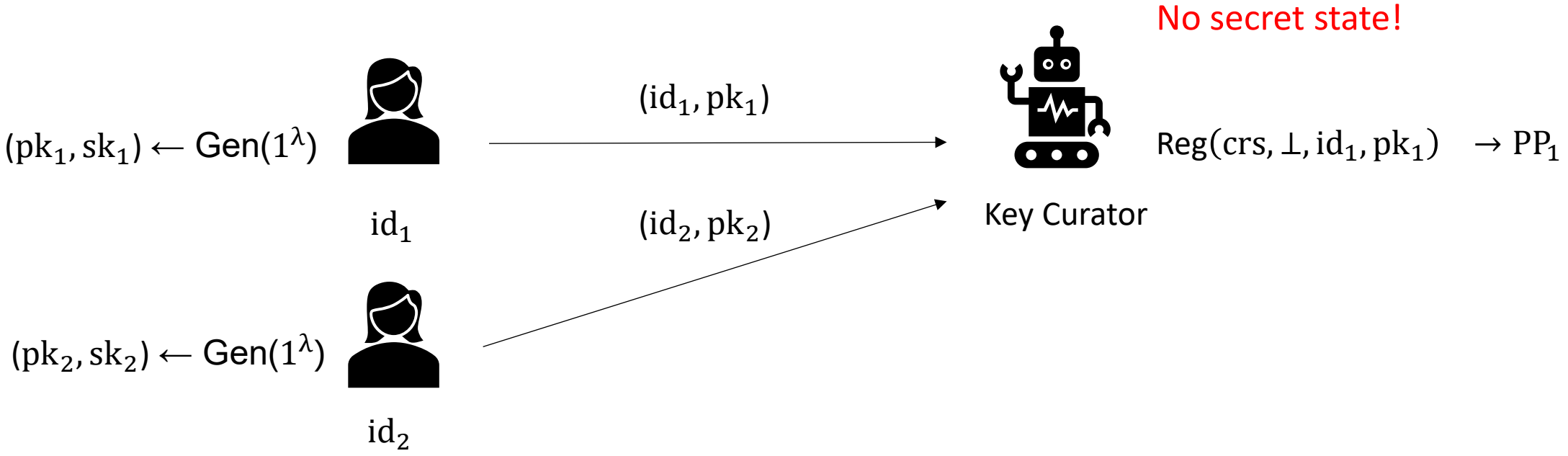


Key Curator

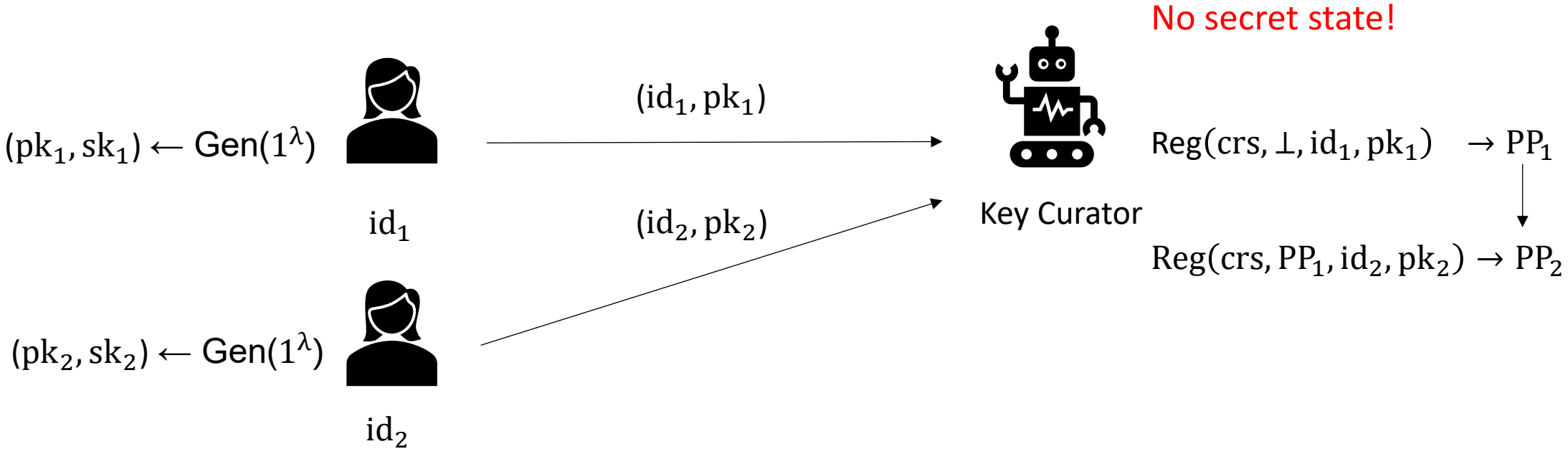
No secret state!

$\mathrm{Reg}(\mathrm{crs}, \perp, \mathrm{id}_1, \mathrm{pk}_1) \rightarrow \mathrm{PP}_1$

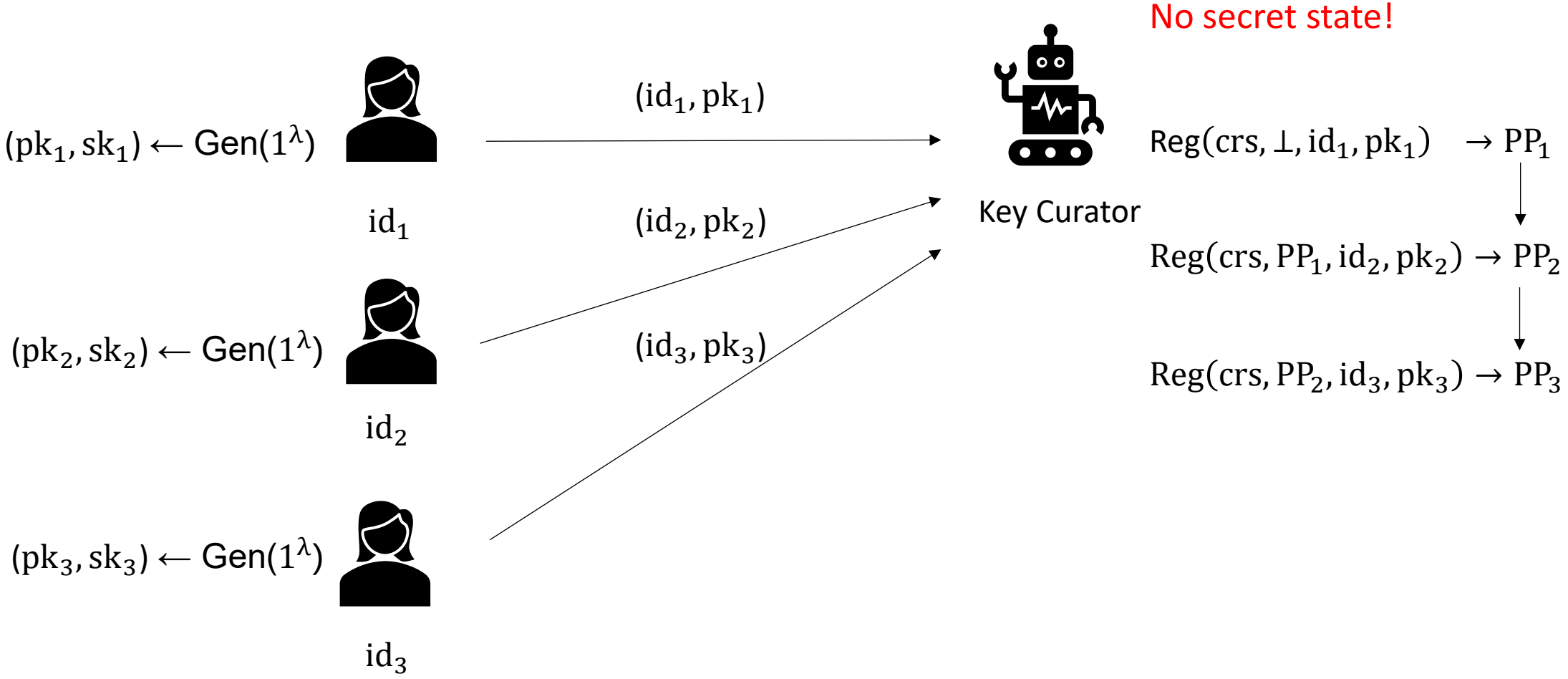
Registration based Encryption [GHMR18]



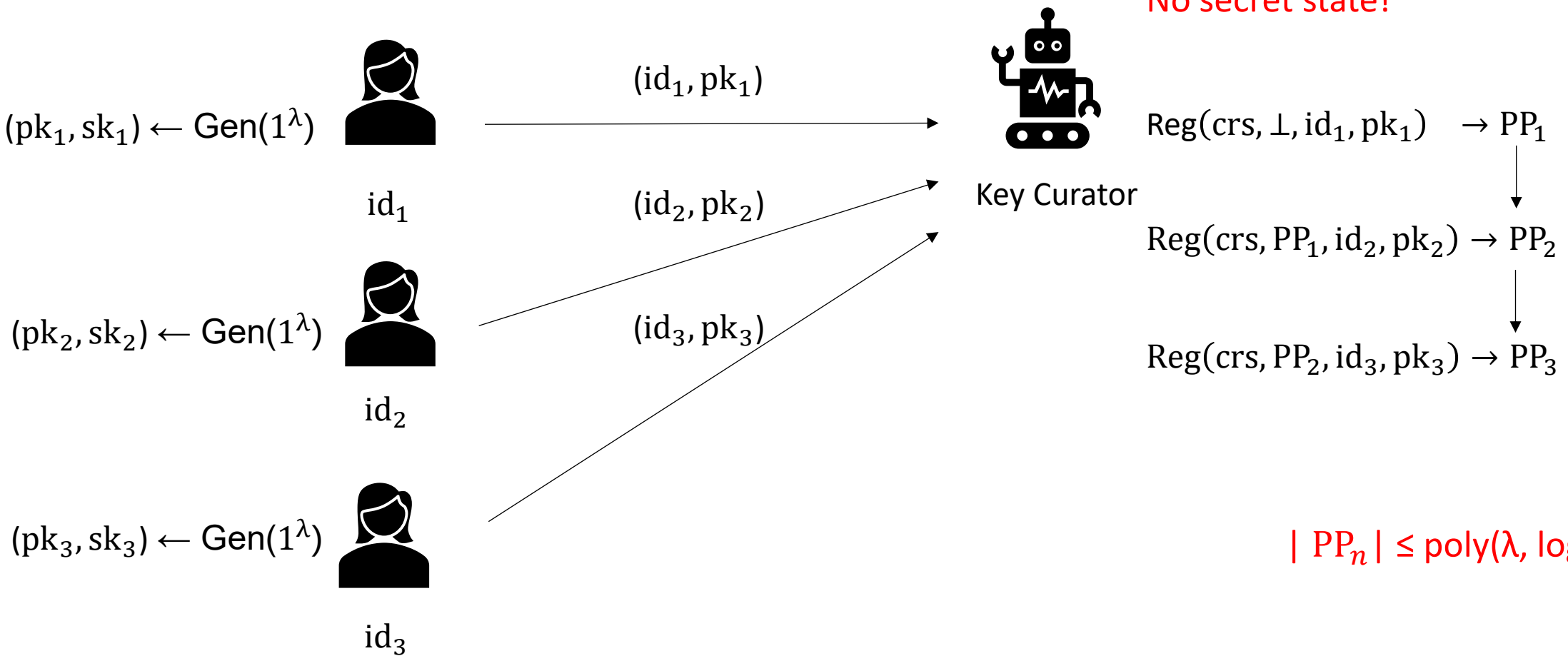
Registration based Encryption [GHMR18]



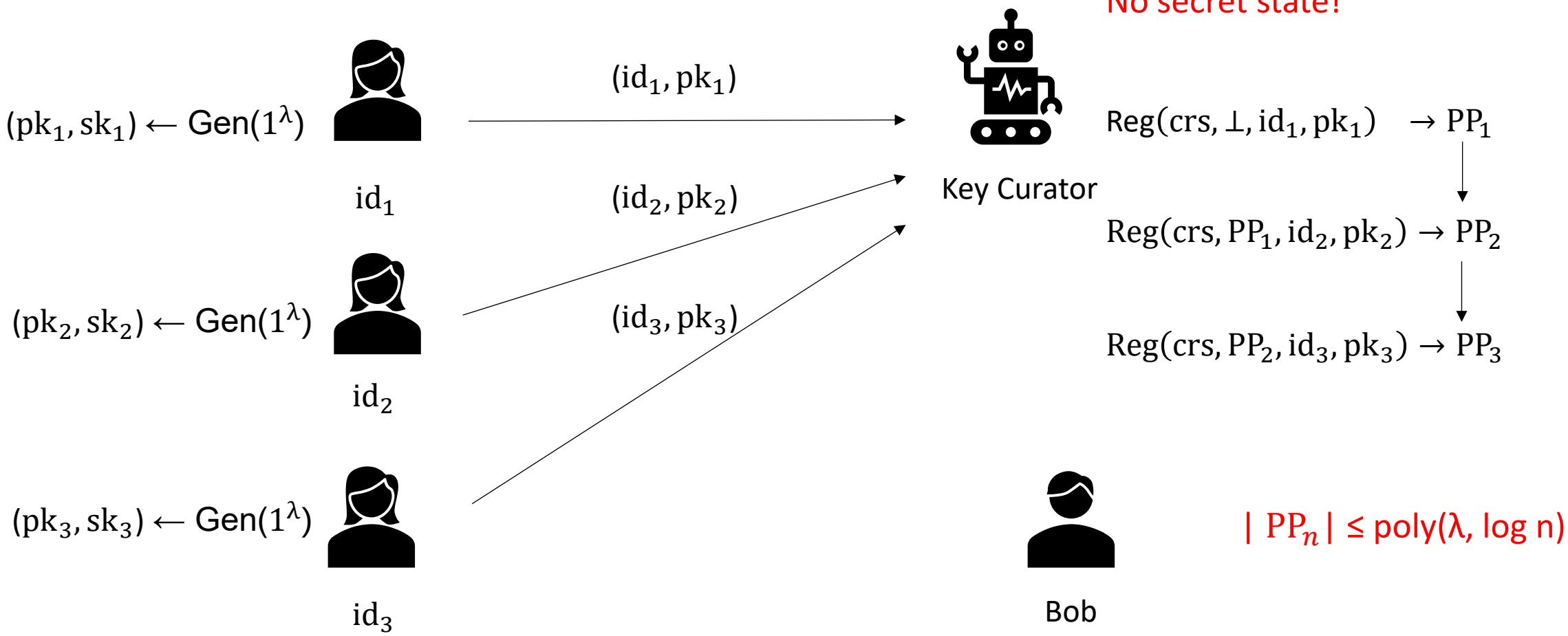
Registration based Encryption [GHMR18]



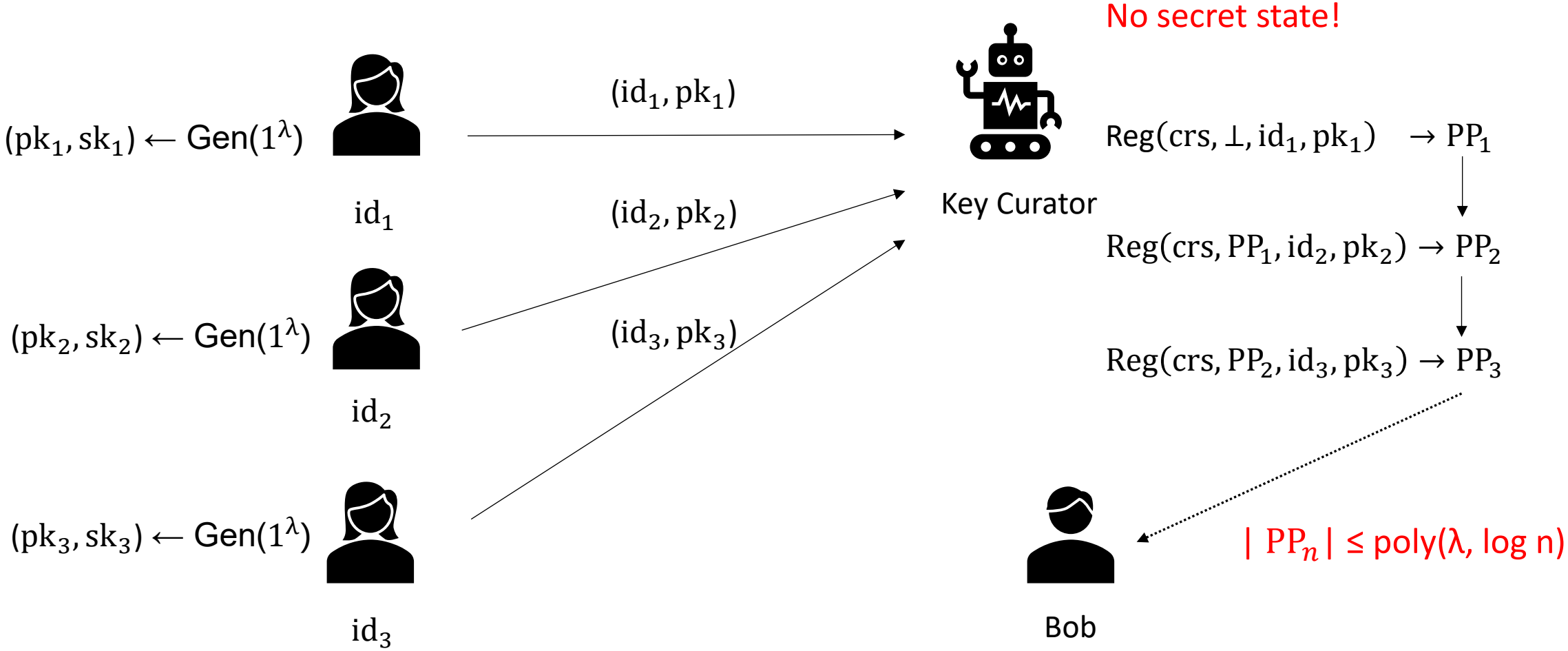
Registration based Encryption [GHMR18]



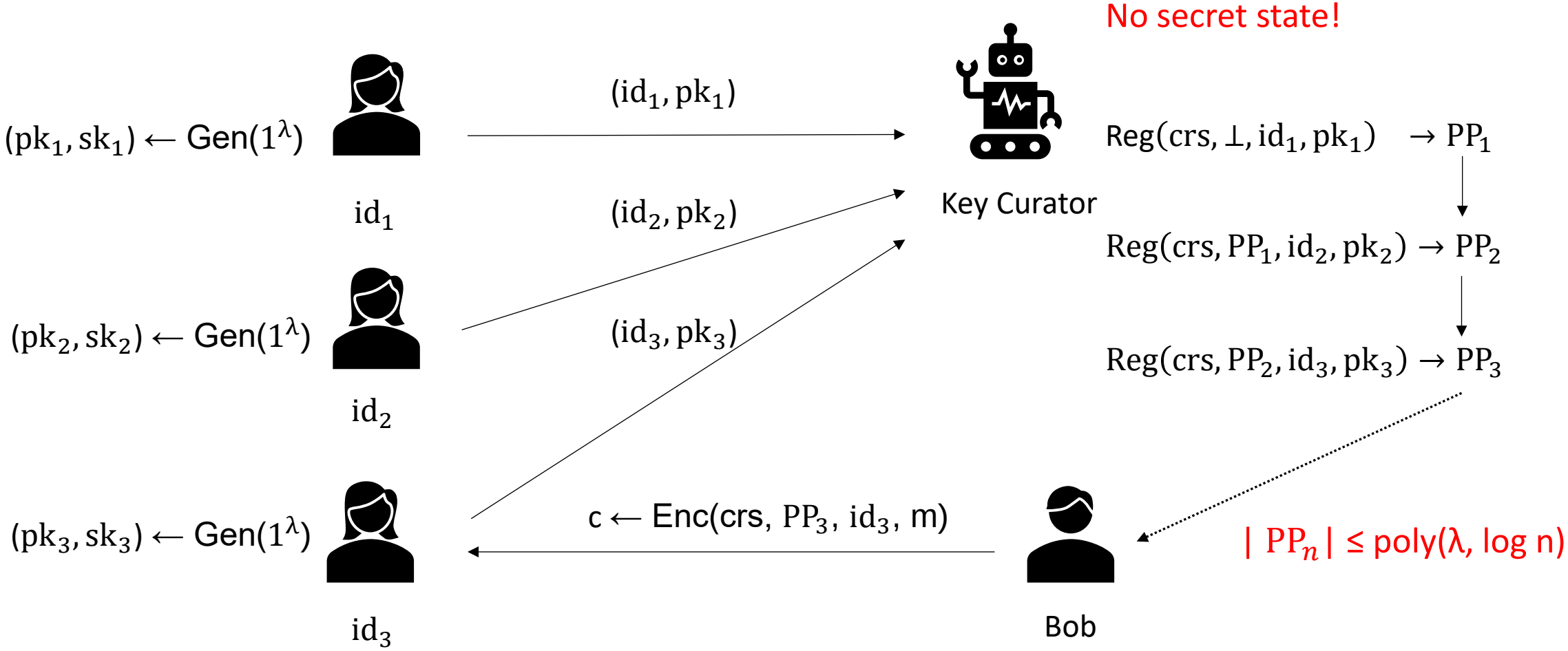
Registration based Encryption [GHMR18]



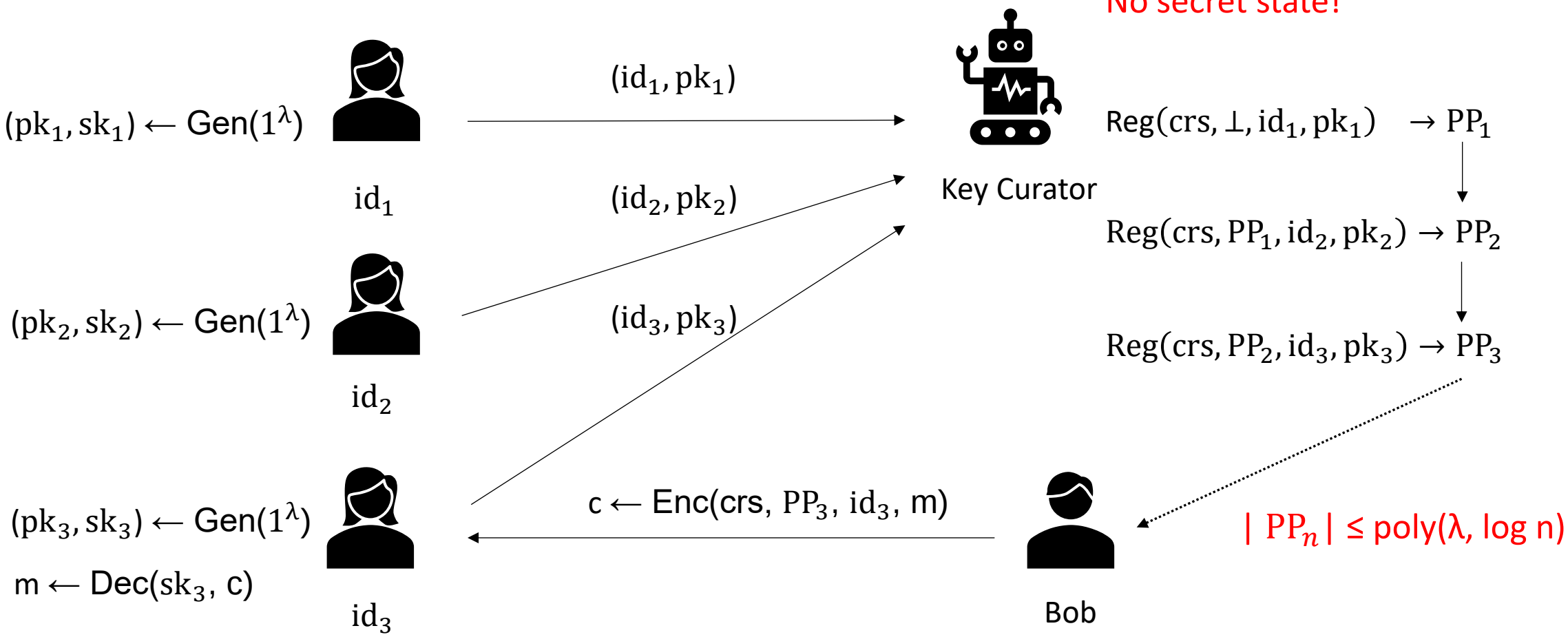
Registration based Encryption [GHMR18]



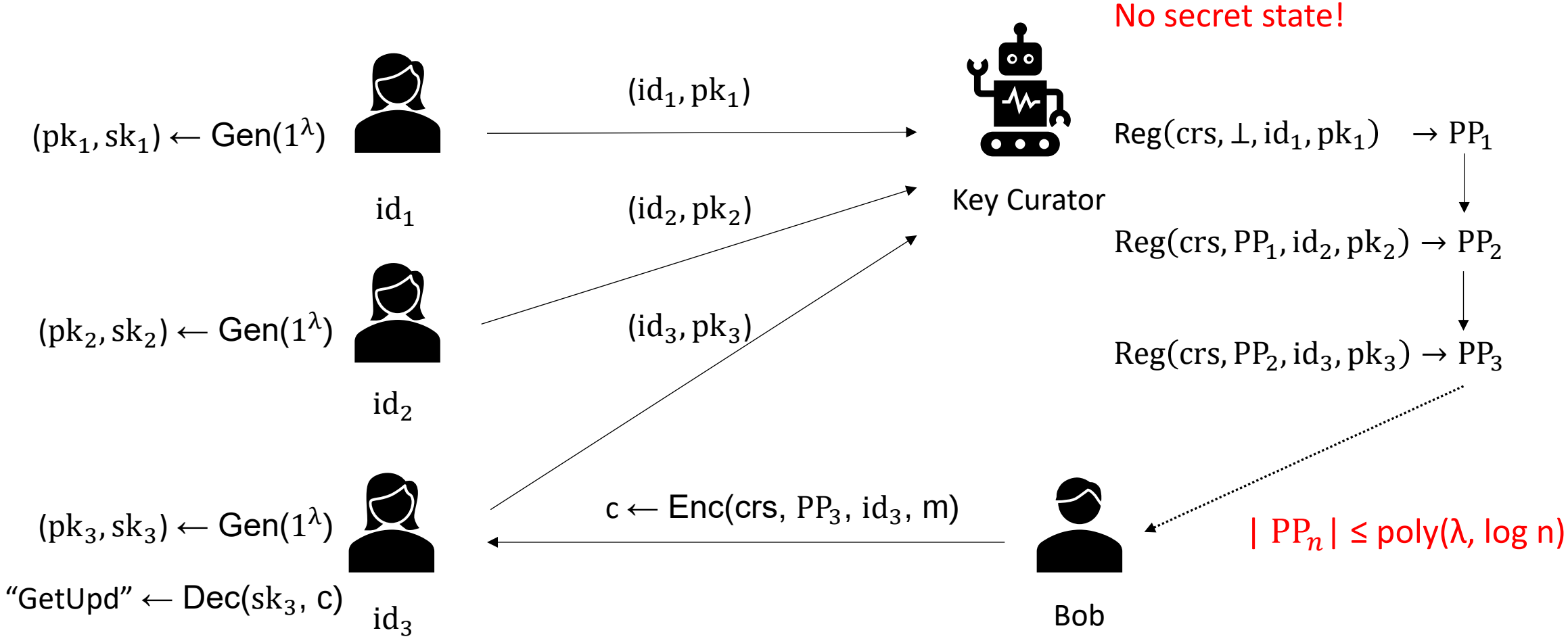
Registration based Encryption [GHMR18]



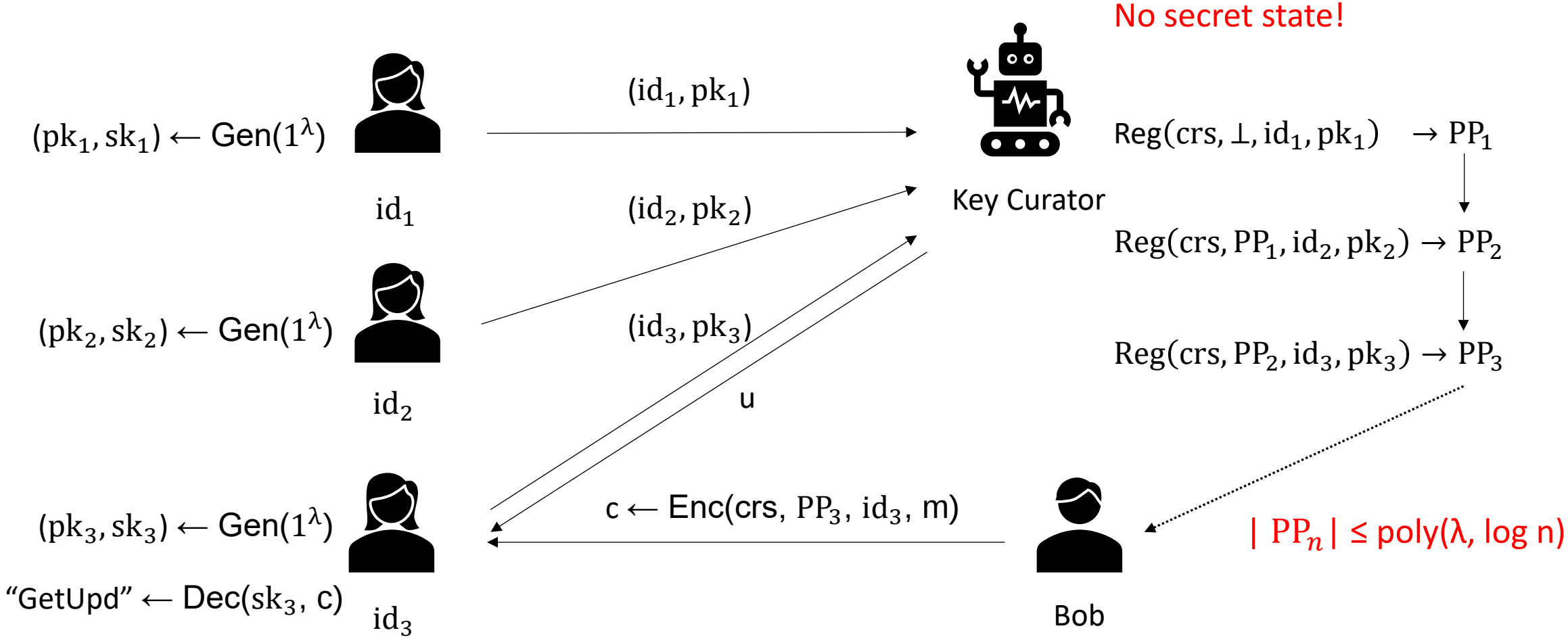
Registration based Encryption [GHMR18]



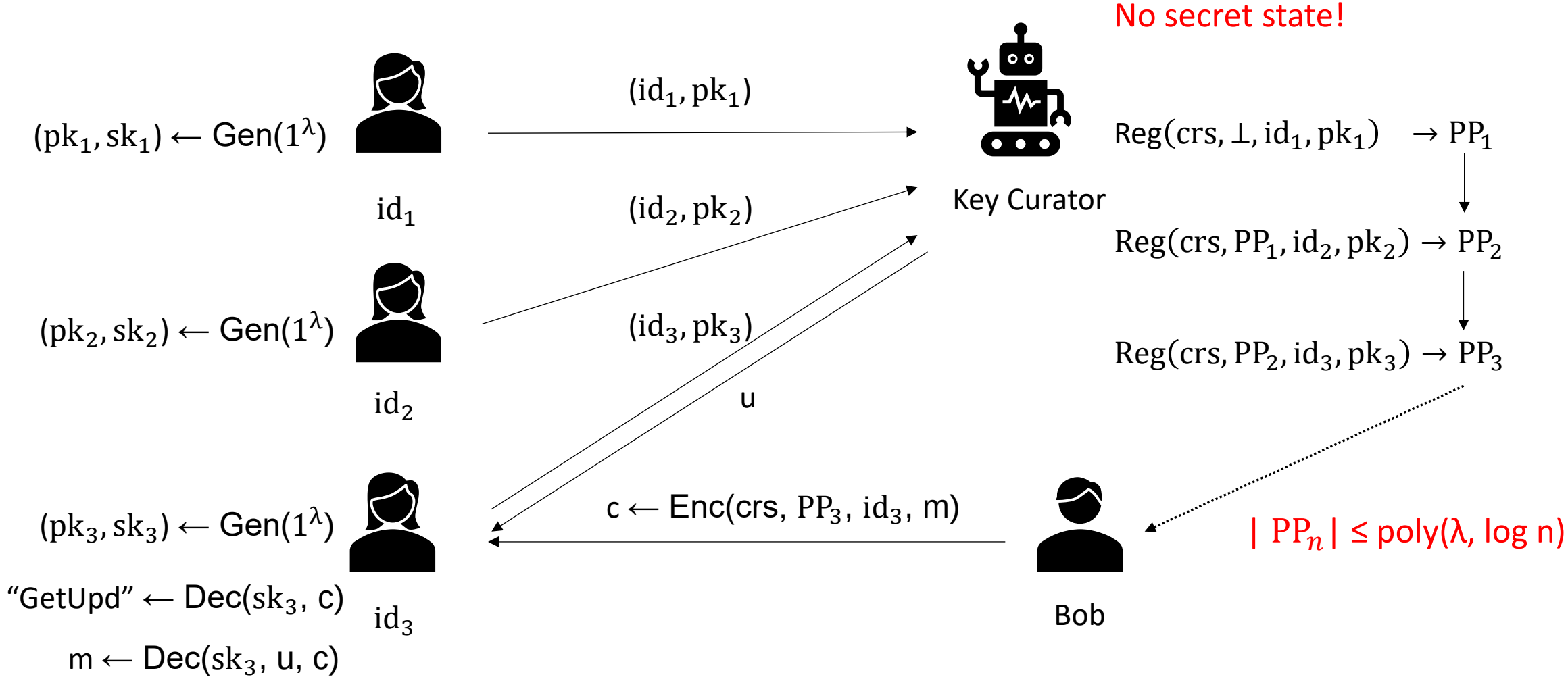
Registration based Encryption [GHMR18]



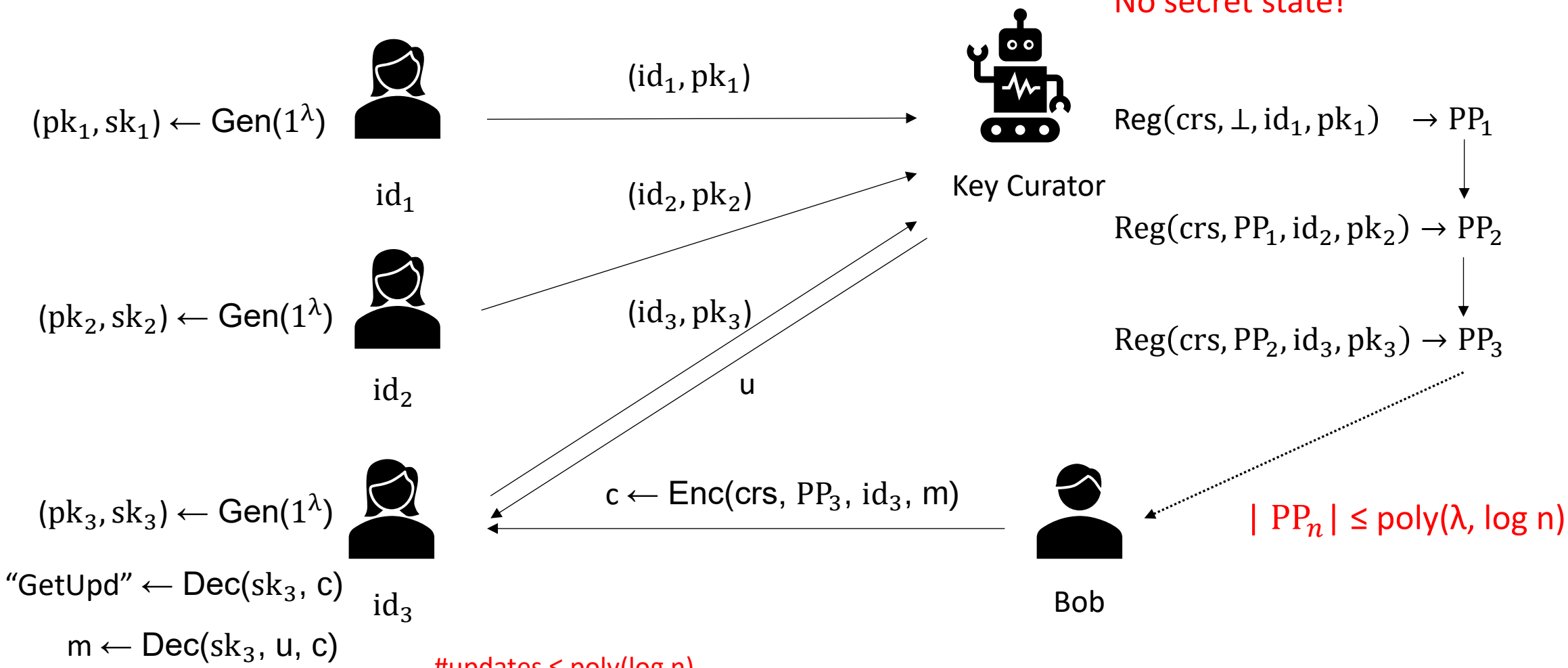
Registration based Encryption [GHMR18]



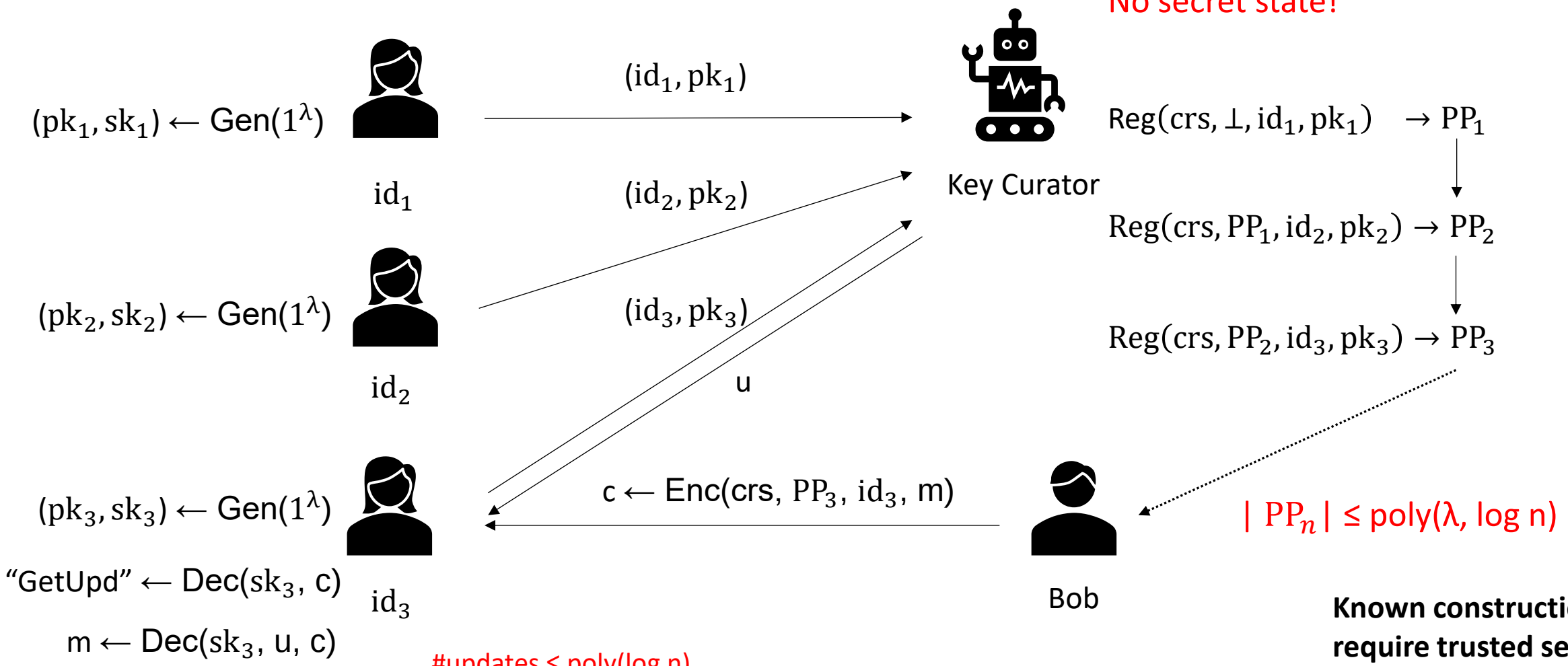
Registration based Encryption [GHMR18]



Registration based Encryption [GHMR18]



Registration based Encryption [GHMR18]



Can trusted setup be removed?

Can trusted setup be removed?

Our Results:

- Impossibility of standard RBE in plain model.

Can trusted setup be removed?

Our Results:

- Impossibility of standard RBE in plain model.
- New definition.

Can trusted setup be removed?

Our Results:

- Impossibility of standard RBE in plain model.
- New definition.
- Plain model construction.

Standard Security Definition

Standard Security Definition

1. Adv registers users:

Standard Security Definition

1. Adv registers users:
   a. Corrupted user. (id and key chosen by adv)

Standard Security Definition

1. Adv registers users:
   a. Corrupted user. (id and key chosen by adv)
   b. (Optional) Honest user. (id chosen by adv)

Standard Security Definition

1. Adv registers users:
   a. Corrupted user. (id and key chosen by adv)
   b. (Optional) Honest user. (id chosen by adv)

2. Adv asks Chal for encryption of a random bit to the honest user or an unregistered user.

Standard Security Definition

1. Adv registers users:
   a. Corrupted user. (id and key chosen by adv)
   b. (Optional) Honest user. (id chosen by adv)

2. Adv asks Chal for encryption of a random bit to the honest user or an unregistered user.

3. Adv wins if it outputs the correct bit.

Standard Security Definition

1. Adv registers users:
   a. Corrupted user. (id and key chosen by adv)
   b. (Optional) Honest user. (id chosen by adv)

2. Adv asks Chal for encryption of a random bit to the honest user or an unregistered user.

3. Adv wins if it outputs the correct bit.

Security must hold even when every registered user is corrupted!

Impossibility of Standard RBE

Impossibility of Standard RBE

Observation: PP_n is length-shrinking hash of n (id, pk) pairs, with CRS as the key.

Impossibility of Standard RBE

Observation: PP_n is length-shrinking hash of n (id, pk) pairs, with CRS as the key.

Idea: Adapt non-uniform attack for keyless CRHF.

Impossibility of Standard RBE

Observation: PP_n is length-shrinking hash of n (id, pk) pairs, with CRS as the key.

Idea: Adapt non-uniform attack for keyless CRHF.

Advice:
1. $(\mathrm{pk}_1, \mathrm{sk}_1)$, …, $(\mathrm{pk}_n, \mathrm{sk}_n)$.

2. $(\mathrm{id}_1, .., \mathrm{id}_n)$ and $(\mathrm{id}_1', .., \mathrm{id}_n')$ satisfying:

Impossibility of Standard RBE

Observation: PP_n is length-shrinking hash of n (id, pk) pairs, with CRS as the key.

Idea: Adapt non-uniform attack for keyless CRHF.

Advice:

1. $(\mathrm{pk}_1, \mathrm{sk}_1)$, ..., $(\mathrm{pk}_n, \mathrm{sk}_n)$.

2. $(\mathrm{id}_1, .., \mathrm{id}_n)$ and $(\mathrm{id}_1', .., \mathrm{id}_n')$ satisfying:

    a. $\mathrm{id}_j' \notin (\mathrm{id}_1, .., \mathrm{id}_n)$

Impossibility of Standard RBE

Observation: PP_n is length-shrinking hash of n (id, pk) pairs, with CRS as the key.

Idea: Adapt non-uniform attack for keyless CRHF.

Advice:

1. $(\mathrm{pk}_1, \mathrm{sk}_1)$, …, $(\mathrm{pk}_n, \mathrm{sk}_n)$.

2. $(\mathrm{id}_1, .., \mathrm{id}_n)$ and $(\mathrm{id}_1', .., \mathrm{id}_n')$ satisfying:

    a. $\mathrm{id}_j' \notin (\mathrm{id}_1, .., \mathrm{id}_n)$

    b. registering $(\mathrm{id}_1, \mathrm{pk}_1)$, …, $(\mathrm{id}_n, \mathrm{pk}_n)$ and $(\mathrm{id}_1', \mathrm{pk}_1)$, …, $(\mathrm{id}_n', \mathrm{pk}_n)$ yields the same PP_n

Impossibility of Standard RBE

Observation: PP_n is length-shrinking hash of n (id, pk) pairs, with CRS as the key.

Idea: Adapt non-uniform attack for keyless CRHF.

Advice:

1. $(\mathrm{pk}_1, \mathrm{sk}_1)$, …, $(\mathrm{pk}_n, \mathrm{sk}_n)$.

2. $(\mathrm{id}_1, .., \mathrm{id}_n)$ and $(\mathrm{id}_1', .., \mathrm{id}_n')$ satisfying:

    a. $\mathrm{id}_j' \notin (\mathrm{id}_1, .., \mathrm{id}_n)$

    b. registering $(\mathrm{id}_1, \mathrm{pk}_1)$, …, $(\mathrm{id}_n, \mathrm{pk}_n)$ and $(\mathrm{id}_1', \mathrm{pk}_1)$, …, $(\mathrm{id}_n', \mathrm{pk}_n)$ yields the same PP_n

Attack:

Impossibility of Standard RBE

Observation: PP_n is length-shrinking hash of n (id, pk) pairs, with CRS as the key.

Idea: Adapt non-uniform attack for keyless CRHF.

Advice:

1. $(\mathrm{pk}_1, \mathrm{sk}_1)$, …, $(\mathrm{pk}_n, \mathrm{sk}_n)$.

2. $(\mathrm{id}_1, .., \mathrm{id}_n)$ and $(\mathrm{id}_1', .., \mathrm{id}_n')$ satisfying:

    a. $\mathrm{id}_j' \notin (\mathrm{id}_1, .., \mathrm{id}_n)$

    b. registering $(\mathrm{id}_1, \mathrm{pk}_1)$, …, $(\mathrm{id}_n, \mathrm{pk}_n)$ and $(\mathrm{id}_1', \mathrm{pk}_1)$, …, $(\mathrm{id}_n', \mathrm{pk}_n)$ yields the same PP_n

Attack:

1. Register $(\mathrm{id}_1, \mathrm{pk}_1)$, …, $(\mathrm{id}_n, \mathrm{pk}_n)$.

Impossibility of Standard RBE

Observation: PP_n is length-shrinking hash of n (id, pk) pairs, with CRS as the key.

Idea: Adapt non-uniform attack for keyless CRHF.

Advice:

1. $(\mathrm{pk}_1, \mathrm{sk}_1)$, …, $(\mathrm{pk}_n, \mathrm{sk}_n)$.

2. $(\mathrm{id}_1, .., \mathrm{id}_n)$ and $(\mathrm{id}_1', .., \mathrm{id}_n')$ satisfying:

    a. $\mathrm{id}_j' \notin (\mathrm{id}_1, .., \mathrm{id}_n)$

    b. registering $(\mathrm{id}_1, \mathrm{pk}_1)$, …, $(\mathrm{id}_n, \mathrm{pk}_n)$ and $(\mathrm{id}_1', \mathrm{pk}_1)$, …, $(\mathrm{id}_n', \mathrm{pk}_n)$ yields the same PP_n

Attack:

1. Register $(\mathrm{id}_1, \mathrm{pk}_1)$, …, $(\mathrm{id}_n, \mathrm{pk}_n)$.

2. Ask for encryption to id_j'.

Impossibility of Standard RBE

Observation: PP_n is length-shrinking hash of n (id, pk) pairs, with CRS as the key.

Idea: Adapt non-uniform attack for keyless CRHF.

Advice:

1. $(\mathrm{pk}_1, \mathrm{sk}_1)$, …, $(\mathrm{pk}_n, \mathrm{sk}_n)$.

2. $(\mathrm{id}_1, .., \mathrm{id}_n)$ and $(\mathrm{id}_1', .., \mathrm{id}_n')$ satisfying:

    a. $\mathrm{id}_j' \notin (\mathrm{id}_1, .., \mathrm{id}_n)$

    b. registering $(\mathrm{id}_1, \mathrm{pk}_1)$, …, $(\mathrm{id}_n, \mathrm{pk}_n)$ and $(\mathrm{id}_1', \mathrm{pk}_1)$, …, $(\mathrm{id}_n', \mathrm{pk}_n)$ yields the same PP_n

Attack:

1. Register $(\mathrm{id}_1, \mathrm{pk}_1)$, …, $(\mathrm{id}_n, \mathrm{pk}_n)$.

2. Ask for encryption to id_j'.

3. Use sk_j to decrypt.

Impossibility of Standard RBE

Observation: PP_n is length-shrinking hash of n (id, pk) pairs, with CRS as the key.

Idea: Adapt non-uniform attack for keyless CRHF.

Advice:

1. $(\mathrm{pk}_1, \mathrm{sk}_1)$, …, $(\mathrm{pk}_n, \mathrm{sk}_n)$.

2. $(\mathrm{id}_1, .., \mathrm{id}_n)$ and $(\mathrm{id}_1', .., \mathrm{id}_n')$ satisfying:

    a. $\mathrm{id}_j' \notin (\mathrm{id}_1, .., \mathrm{id}_n)$

    b. registering $(\mathrm{id}_1, \mathrm{pk}_1)$, …, $(\mathrm{id}_n, \mathrm{pk}_n)$ and $(\mathrm{id}_1', \mathrm{pk}_1)$, …, $(\mathrm{id}_n', \mathrm{pk}_n)$ yields the same PP_n

Attack:

1. Register $(\mathrm{id}_1, \mathrm{pk}_1)$, …, $(\mathrm{id}_n, \mathrm{pk}_n)$.

2. Ask for encryption to id_j'.

3. Use sk_j to decrypt.

Valid since attacking unregistered user without honest user is legitimate!

New Security Definition

New Security Definition

1. Adv registers users:
   a. Corrupted user.
   b. Honest user.

New Security Definition

1. Adv registers users:
   a. Corrupted user.
   b. Honest user.

2. Adv asks Chal for encryption to honest user or unregistered user.

New Security Definition

1. Adv registers users:
   a. Corrupted user.
   b. Honest user.

2. Adv asks Chal for encryption to honest user or unregistered user.

3. Adv wins if outputs the correct bit and there is an honest user.

Construction

Construction

Idea:

Construction

Idea:

1. KC starts with a fixed dummy CRS.

Construction

Idea:

1. KC starts with a fixed dummy CRS.

2. User generates keys and randomness to refresh CRS.

Construction

Idea:

1. KC starts with a fixed dummy CRS.

2. User generates keys and randomness to refresh CRS.

3. KC re-randomizes the CRS and re-registers against the refreshed CRS.

Construction

Idea:

1. KC starts with a fixed dummy CRS.

2. User generates keys and randomness to refresh CRS.

3. KC re-randomizes the CRS and re-registers against the refreshed CRS.

4. Security holds thanks to the honest user.

Construction

Idea:

1. KC starts with a fixed dummy CRS.

2. User generates keys and randomness to refresh CRS.

3. KC re-randomizes the CRS and re-registers against the refreshed CRS.

4. Security holds thanks to the honest user.

Simplified description:
leaking randomness hurts security.

Construction

Idea:

1. KC starts with a fixed dummy CRS.

2. User generates keys and randomness to refresh CRS.

3. KC re-randomizes the CRS and re-registers against the refreshed CRS.

4. Security holds thanks to the honest user.

Simplified description:
leaking randomness hurts security.

Need more ideas for real construction!

Construction

Idea:

1. KC starts with a fixed dummy CRS.

2. User generates keys and randomness to refresh CRS.

3. KC re-randomizes the CRS and re-registers against the refreshed CRS.

4. Security holds thanks to the honest user.

Simplified description:
leaking randomness hurts security.

Need more ideas for real construction!
(Details in the paper.)

Construction

Construction

Like previous works on RBE, follow general Merkle-tree based paradigm.

Construction

Like previous works on RBE, follow general Merkle-tree based paradigm.

Construction

Like previous works on RBE, follow general Merkle-tree based paradigm.

Reg(crs, id, pk):

Construction

Like previous works on RBE, follow general Merkle-tree based paradigm.

Reg(crs, id, pk):

1. Read k from crs

Construction

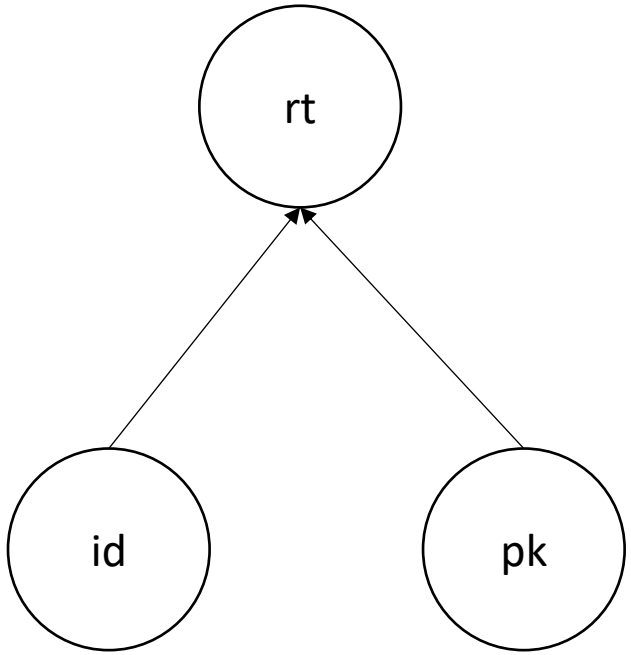
Like previous works on RBE, follow general Merkle-tree based paradigm.

Reg(crs, id, pk):

1. Read k from crs
2. Compute pp := rt = hash(k, id||pk)

Construction

Like previous works on RBE, follow general Merkle-tree based paradigm.

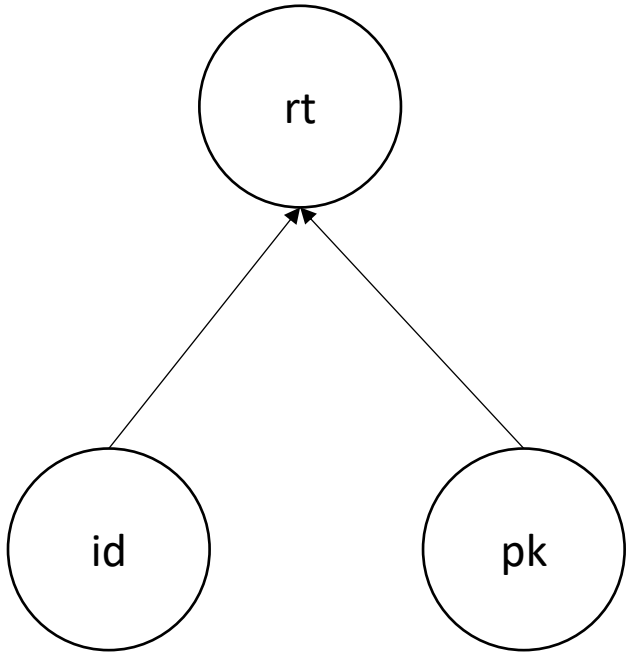


Reg(crs, id, pk):

1. Read k from crs
2. Compute pp := rt = hash(k, id||pk)

Construction

Like previous works on RBE, follow general Merkle-tree based paradigm.



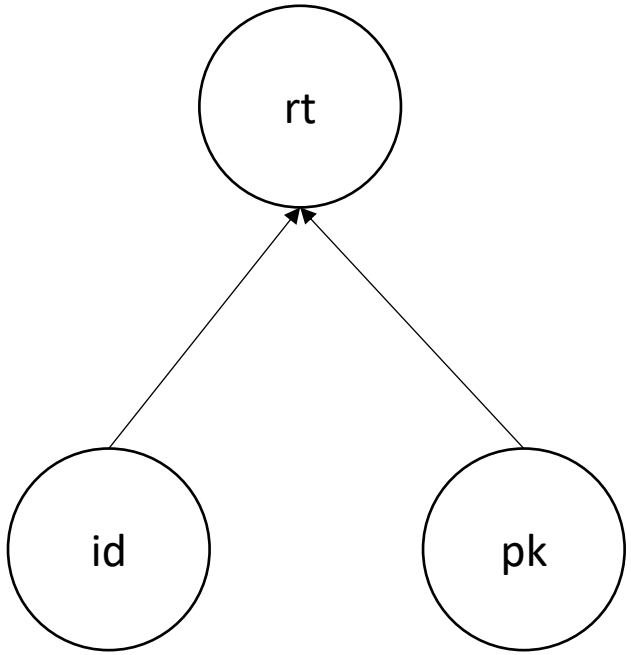
Reg(crs, id, pk):

1. Read k from crs
2. Compute pp := rt = hash(k, id||pk)

Enc(crs, pp, id, m):

Construction

Like previous works on RBE, follow general Merkle-tree based paradigm.



Reg(crs, id, pk):

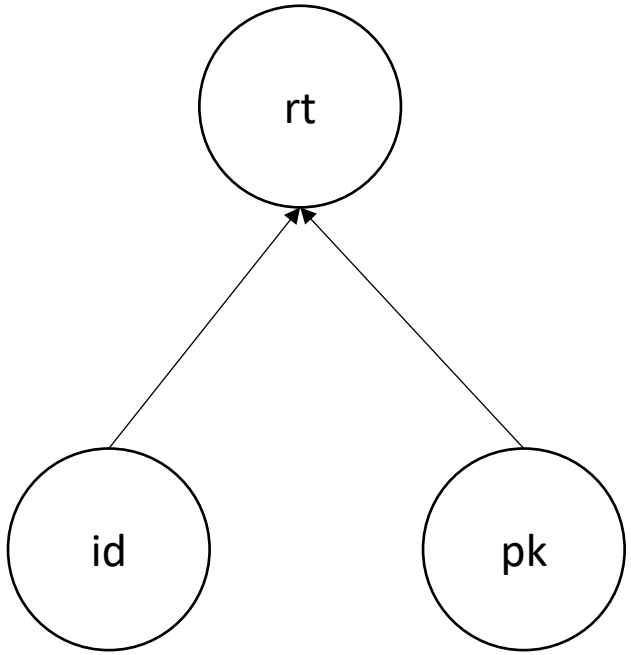
1. Read k from crs
2. Compute pp := rt = hash(k, id||pk)

Enc(crs, pp, id, m):

Return Obf($C_{k,rt,id,m}$)

Construction

Like previous works on RBE, follow general Merkle-tree based paradigm.



Reg(crs, id, pk):

1. Read k from crs
2. Compute pp := rt = hash(k, id||pk)

Enc(crs, pp, id, m):

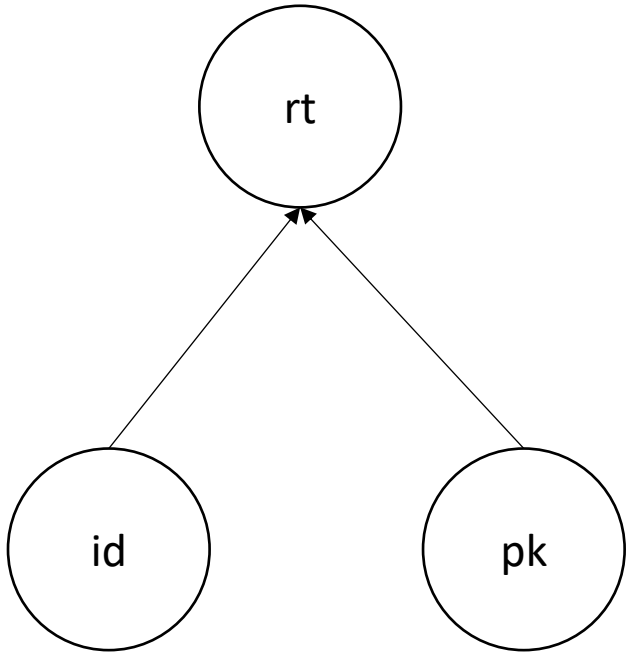
Return Obf($C_{k,rt,id,m}$)

$C_{k,rt,id,\,m}$(pk'):

1. Compute rt' = hash(k, id||pk')
2. If rt == rt', output Enc(pk', m)

Construction

Like previous works on RBE, follow general Merkle-tree based paradigm.



Observation: if crs contains only the hash key, then all we need is re-randomizable key!

Reg(crs, id, pk):

1. Read k from crs
2. Compute pp := rt = hash(k, id||pk)

Enc(crs, pp, id, m):

Return Obf($C_{k,rt,id,m}$)

$C_{k,rt,id,\,m}$(pk'):

1. Compute rt' = hash(k, id||pk')
2. If rt == rt', output Enc(pk', m)

Construction

Construction

Follow framework in [GHMRS19]:

PKE + Hash Encryption + Garbled Circuit → RBE

Construction

Follow framework in [GHMRS19]:

PKE + Hash Encryption + Garbled Circuit → RBE

CRS only contains the key for the hash encryption scheme.

Construction

Follow framework in [GHMRS19]:

PKE + Hash Encryption + Garbled Circuit → RBE

CRS only contains the key for the hash encryption scheme.

Suffice to build hash encryption with re-randomizable key!

Construction

Follow framework in [GHMRS19]:

PKE + Hash Encryption + Garbled Circuit → RBE

CRS only contains the key for the hash encryption scheme.

Suffice to build hash encryption with re-randomizable key!

The CDH chameleon encryption in [DG17], when interpreted as a hash encryption, has re-randomizable key.

Construction

Construction

Key generation: uniformly sample $\alpha_{i,j} \leftarrow Z_p^*$ and set $g_{i,j} = g^{\alpha_{i,j}}$.

$$k := \left(g, \begin{pmatrix} g_{1,0} & g_{2,0} & \cdots & g_{n,0} \\ g_{1,1} & g_{2,1} & \cdots & g_{n,1} \end{pmatrix} \right)$$

Construction

Key generation: uniformly sample $\alpha_{i,j} \leftarrow Z_p^*$ and set $g_{i,j} = g^{\alpha_{i,j}}$.

$$k \coloneqq \left(g, \begin{pmatrix} g_{1,0} & g_{2,0} & \cdots & g_{n,0} \\ g_{1,1} & g_{2,1} & \cdots & g_{n,1} \end{pmatrix}\right)$$

Key re-randomization: uniformly sample $\beta_{i,j} \leftarrow Z_p^*$.

Construction

Key generation: uniformly sample $\alpha_{i,j} \leftarrow Z_p^*$ and set $g_{i,j} = g^{\alpha_{i,j}}$.

$$k := \left(g, \begin{pmatrix} g_{1,0} & g_{2,0} & \cdots & g_{n,0} \\ g_{1,1} & g_{2,1} & \cdots & g_{n,1} \end{pmatrix} \right)$$

Key re-randomization: uniformly sample $\beta_{i,j} \leftarrow Z_p^*$.

$$k' := \left(g, \begin{pmatrix} g_{1,0}^{\beta_{1,0}} & g_{2,0}^{\beta_{2,0}} & \cdots & g_{n,0}^{\beta_{n,0}} \\ g_{1,1}^{\beta_{1,1}} & g_{2,1}^{\beta_{2,1}} & \cdots & g_{n,1}^{\beta_{n,1}} \end{pmatrix} \right)$$

Construction

Key generation: uniformly sample $\alpha_{i,j} \leftarrow Z_p^*$ and set $g_{i,j} = g^{\alpha_{i,j}}$.

$$k := \left(g, \begin{pmatrix} g_{1,0} & g_{2,0} & \cdots & g_{n,0} \\ g_{1,1} & g_{2,1} & \cdots & g_{n,1} \end{pmatrix}\right)$$

Key re-randomization: uniformly sample $\beta_{i,j} \leftarrow Z_p^*$.

$$k' := \left(g, \begin{pmatrix} g_{1,0}^{\beta_{1,0}} & g_{2,0}^{\beta_{2,0}} & \cdots & g_{n,0}^{\beta_{n,0}} \\ g_{1,1}^{\beta_{1,1}} & g_{2,1}^{\beta_{2,1}} & \cdots & g_{n,1}^{\beta_{n,1}} \end{pmatrix}\right)$$

Re-randomized keys are identically distributed as freshly sampled keys.

Construction

Key generation: uniformly sample $\alpha_{i,j} \leftarrow Z_p^*$ and set $g_{i,j} = g^{\alpha_{i,j}}$.

$$k := \left(g, \begin{pmatrix} g_{1,0} & g_{2,0} & \cdots & g_{n,0} \\ g_{1,1} & g_{2,1} & \cdots & g_{n,1} \end{pmatrix}\right)$$

Key re-randomization: uniformly sample $\beta_{i,j} \leftarrow Z_p^*$.

$$k' := \left(g, \begin{pmatrix} g_{1,0}^{\beta_{1,0}} & g_{2,0}^{\beta_{2,0}} & \cdots & g_{n,0}^{\beta_{n,0}} \\ g_{1,1}^{\beta_{1,1}} & g_{2,1}^{\beta_{2,1}} & \cdots & g_{n,1}^{\beta_{n,1}} \end{pmatrix}\right)$$

Re-randomized keys are identically distributed as freshly sampled keys.

Hard to invert the re-randomization.

Construction

Construction

$$k \coloneqq \left(g, \begin{pmatrix} g_{1,0} & g_{2,0} & \cdots & g_{n,0} \\ g_{1,1} & g_{2,1} & \cdots & g_{n,1} \end{pmatrix}\right)$$

Construction

$$k \coloneqq \left(g, \begin{pmatrix} g_{1,0} & g_{2,0} & \cdots & g_{n,0} \\ g_{1,1} & g_{2,1} & \cdots & g_{n,1} \end{pmatrix}\right)$$

Hash(k, x) = $\prod_{i=1}^{n} g_{\mathrm{i},x_{\mathrm{i}}}$

Construction

$$k := \left(g, \begin{pmatrix} g_{1,0} & g_{2,0} & \cdots & g_{n,0} \\ g_{1,1} & g_{2,1} & \cdots & g_{n,1} \end{pmatrix}\right)$$

Hash(k, x) = $\prod_{i=1}^{n} g_{\mathrm{i},x_{\mathrm{i}}}$

$$k' := \left(g, \begin{pmatrix} g_{1,0}^{\beta_{1,0}} & g_{2,0}^{\beta_{2,0}} & \cdots & g_{n,0}^{\beta_{n,0}} \\ g_{1,1}^{\beta_{1,1}} & g_{2,1}^{\beta_{2,1}} & \cdots & g_{n,1}^{\beta_{n,1}} \end{pmatrix}\right)$$

Construction

$$k := \left(g, \begin{pmatrix} g_{1,0} & g_{2,0} & \cdots & g_{n,0} \\ g_{1,1} & g_{2,1} & \cdots & g_{n,1} \end{pmatrix}\right)$$

Hash(k, x) = $\prod_{i=1}^{n} g_{\mathrm{i},x_{\mathrm{i}}}$

$$k' := \left(g, \begin{pmatrix} g_{1,0}^{\beta_{1,0}} & g_{2,0}^{\beta_{2,0}} & \cdots & g_{n,0}^{\beta_{n,0}} \\ g_{1,1}^{\beta_{1,1}} & g_{2,1}^{\beta_{2,1}} & \cdots & g_{n,1}^{\beta_{n,1}} \end{pmatrix}\right)$$

Hash(k', x) = $\prod_{i=1}^{n} g_{\mathrm{i},x_{\mathrm{i}}}^{\beta_{i,x_{\mathrm{i}}}}$

Construction

$$k \coloneqq \left(g, \begin{pmatrix} g_{1,0} & g_{2,0} & \cdots & g_{n,0} \\ g_{1,1} & g_{2,1} & \cdots & g_{n,1} \end{pmatrix}\right)$$

Hash(k, x) = $\prod_{i=1}^{n} g_{\mathrm{i},x_{\mathrm{i}}}$

$$k' \coloneqq \left(g, \begin{pmatrix} g_{1,0}^{\beta_{1,0}} & g_{2,0}^{\beta_{2,0}} & \cdots & g_{n,0}^{\beta_{n,0}} \\ g_{1,1}^{\beta_{1,1}} & g_{2,1}^{\beta_{2,1}} & \cdots & g_{n,1}^{\beta_{n,1}} \end{pmatrix}\right)$$

Hash(k', x) = $\prod_{i=1}^{n} g_{\mathrm{i},x_{\mathrm{i}}}^{\beta_{i,x_{\mathrm{i}}}}$

By hardness of DL, Hash(k', $\cdot$) is collision resistant for honestly sampled k.

Open Problems

Open Problems

- Require $\Omega(n)$ number of decryption updates.

Open Problems

- Require $\Omega(n)$ number of decryption updates.

  Necessary under assumptions satisfied by known constructions.

Open Problems

- Require $\Omega(n)$ number of decryption updates.

  Necessary under assumptions satisfied by known constructions.

- Require strong cryptographic tools while standard RBE can be constructed from CDH.

Open Problems

- Require $\Omega(n)$ number of decryption updates.

  Necessary under assumptions satisfied by known constructions.

- Require strong cryptographic tools while standard RBE can be constructed from CDH.

- Schemes with better concrete efficiency?

Thanks for listening!