Lower Bounds for the Number of Decryption Updates in Registration-Based Encryption

Mohammad Mahmoody,

UNIVERSITY of VIRGINIA

Wei Qi,

UNIVERSITY of VIRGINIA

Ahmadreza Rahimi

MAX PLANCK INSTITUTE FOR SECURITY AND PRIVACY

Public Key Encryption [DH76,RSA78,GM82]

Public Key Encryption [DH76,RSA78,GM82]

Alice

Public Key Encryption [DH76,RSA78,GM82]

Alice

Bob

Public Key Encryption [DH76,RSA78,GM82]

$(pk_1,\ sk_1) \leftarrow \mathrm{Gen}(1^\lambda)$

Alice

Bob

Public Key Encryption [DH76,RSA78,GM82]

$(pk_1,\ sk_1) \leftarrow \mathrm{Gen}(1^\lambda)$

Alice

pk_1

Bob

Public Key Encryption [DH76,RSA78,GM82]

$(pk_1,\ sk_1) \leftarrow \mathrm{Gen}(1^\lambda)$

Alice

pk_1

$c_1 \leftarrow \mathrm{Enc}(pk_1,\ m_1)$

Bob

Public Key Encryption [DH76,RSA78,GM82]

$(pk_1,\ sk_1) \leftarrow \mathrm{Gen}(1^\lambda)$

Alice

pk_1

$c_1 \leftarrow \mathrm{Enc}(pk_1,\ m_1)$

$(pk_2,\ sk_2) \leftarrow \mathrm{Gen}(1^\lambda)$

Christine

pk_2

$c_2 \leftarrow \mathrm{Enc}(pk_2,\ m_2)$

Bob

Public Key Encryption [DH76,RSA78,GM82]

$(pk_1,\ sk_1) \leftarrow \mathrm{Gen}(1^\lambda)$

Alice

pk_1

$c_1 \leftarrow \mathrm{Enc}(pk_1,\ m_1)$

$(pk_2,\ sk_2) \leftarrow \mathrm{Gen}(1^\lambda)$

Christine

pk_2

$c_2 \leftarrow \mathrm{Enc}(pk_2,\ m_2)$

Bob

$(pk_3,\ sk_3) \leftarrow \mathrm{Gen}(1^\lambda)$

Susan

pk_3

$c_3 \leftarrow \mathrm{Enc}(pk_3,\ m_3)$

Public Key Encryption [DH76,RSA78,GM82]

$(pk_1,\ sk_1) \leftarrow \mathrm{Gen}(1^\lambda)$

Alice

pk_1

$c_1 \leftarrow \mathrm{Enc}(pk_1,\ m_1)$

Too many public keys!

$(pk_2,\ sk_2) \leftarrow \mathrm{Gen}(1^\lambda)$

Christine

pk_2

$c_2 \leftarrow \mathrm{Enc}(pk_2,\ m_2)$

Bob

$(pk_3,\ sk_3) \leftarrow \mathrm{Gen}(1^\lambda)$

Susan

pk_3

$c_3 \leftarrow \mathrm{Enc}(pk_3,\ m_3)$

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

Public Parameter
(PP)

Identity Based Encryption [Shamir86, BF01]

Master Secret Key
(MSK)

Central Authority

Public Parameter
(PP)

Alice

Bob

Identity Based Encryption [Shamir86, BF01]

Master Secret Key
(MSK)

Central Authority

Public Parameter
(PP)

Alice

Bob

Identity Based Encryption [Shamir86, BF01]

Master Secret Key
(MSK)

Central Authority

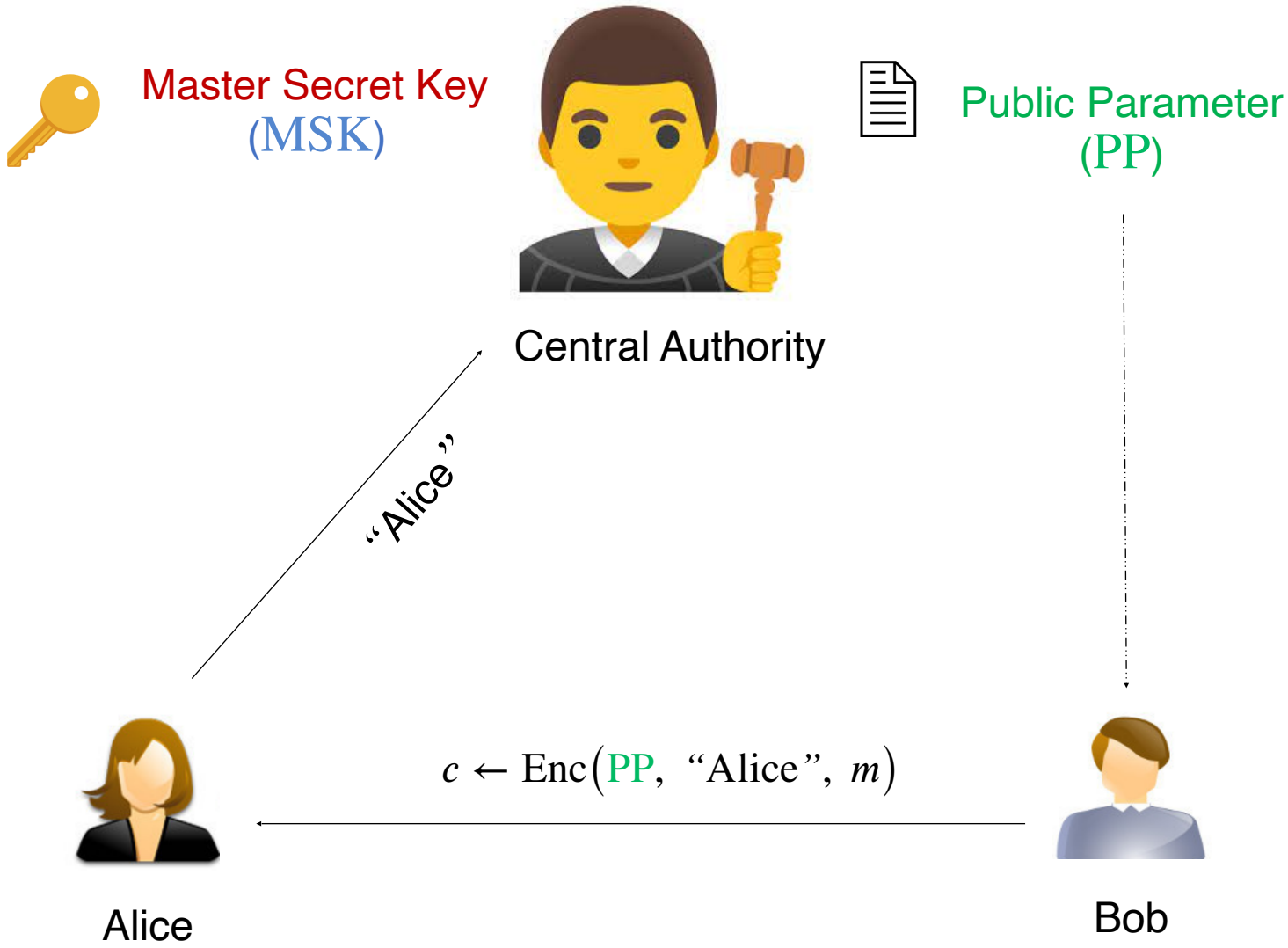
Public Parameter
(PP)

Alice

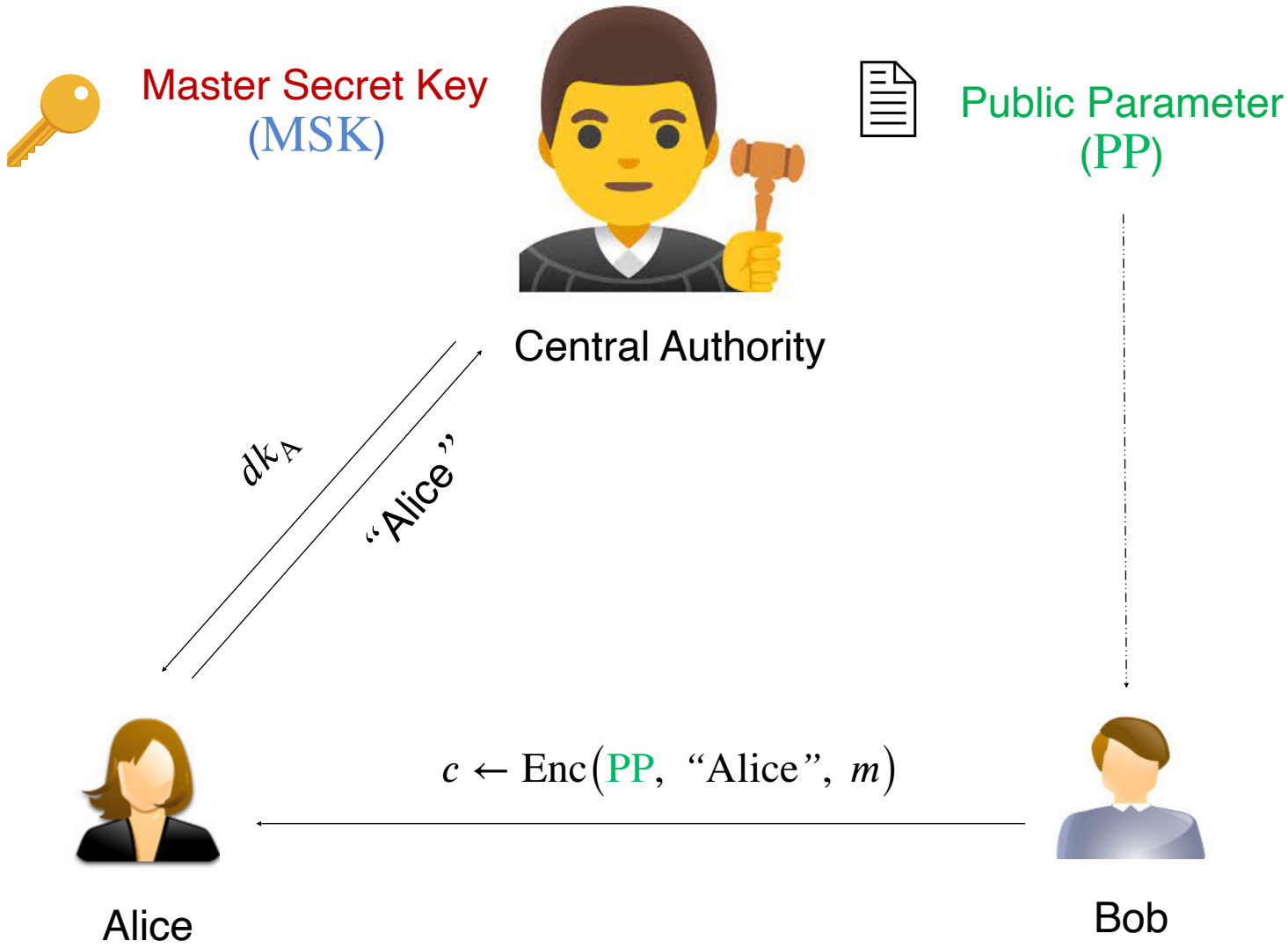
$$c \leftarrow \mathrm{Enc}(\mathrm{PP},\ \text{“Alice”},\ m)$$

Bob

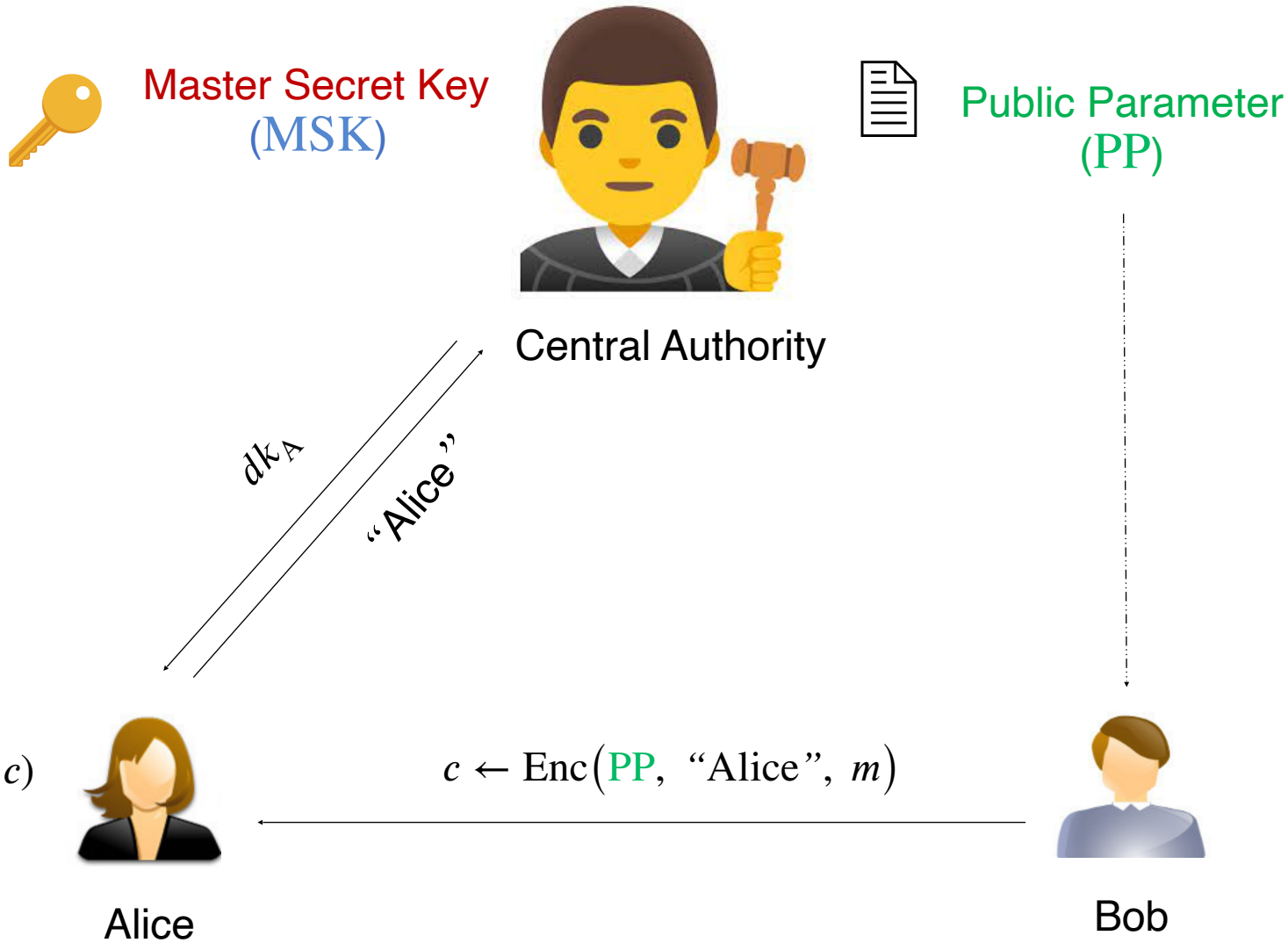
Identity Based Encryption [Shamir86, BF01]



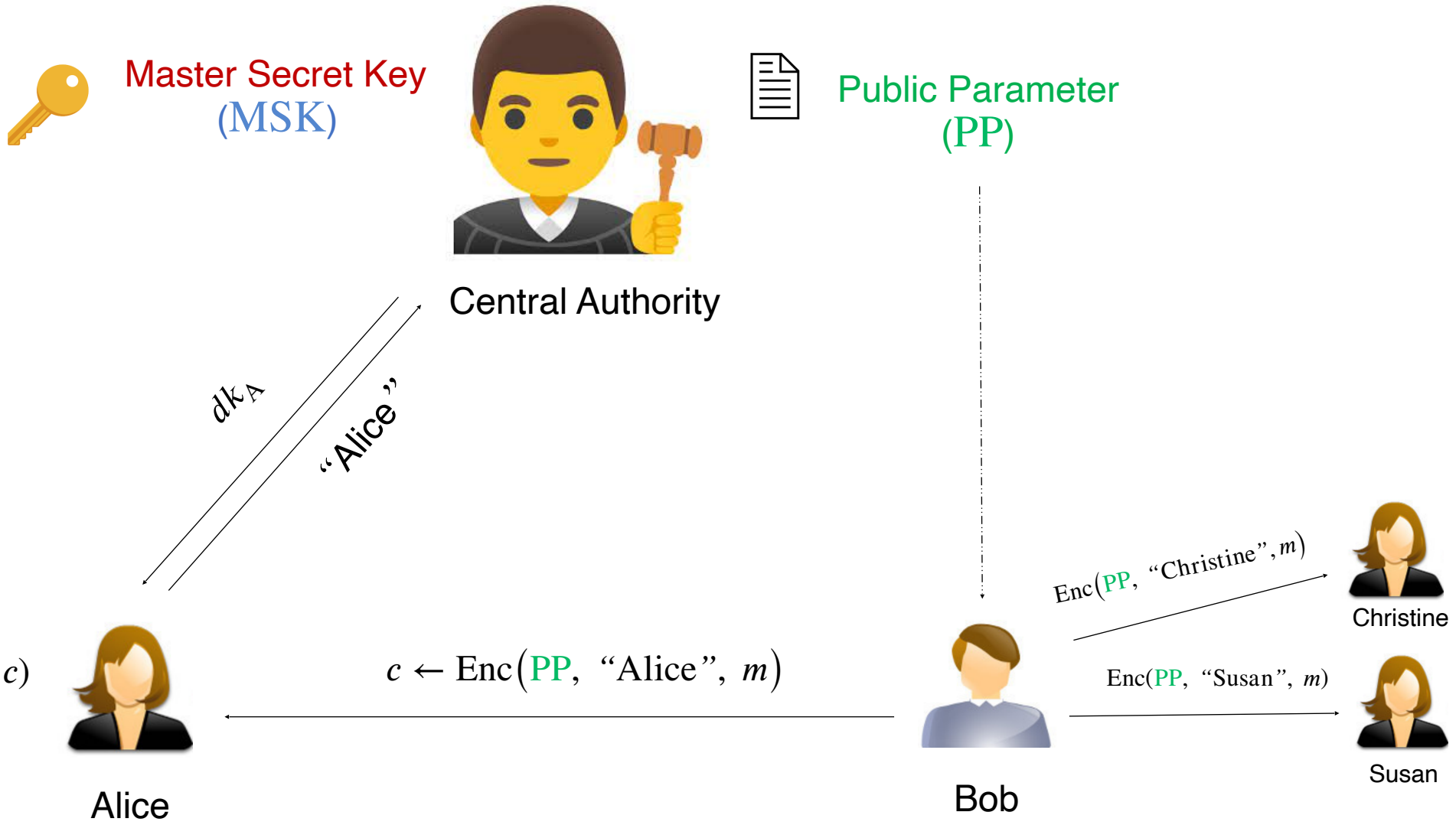
Identity Based Encryption [Shamir86, BF01]



Identity Based Encryption [Shamir86, BF01]



Identity Based Encryption [Shamir86, BF01]



Identity Based Encryption [Shamir86, BF01]

Master Secret Key
(MSK)

Central Authority

Public Parameter
(PP)

One public key

dk_A

"Alice"

$m \leftarrow \mathrm{Dec}(dk_A, c)$

Alice

$c \leftarrow \mathrm{Enc}(\mathrm{PP}, \text{"Alice"}, m)$

Bob

$\mathrm{Enc}(\mathrm{PP}, \text{"Christine"}, m)$

Christine

$\mathrm{Enc}(\mathrm{PP}, \text{"Susan"}, m)$

Susan

Identity Based Encryption [Shamir86, BF01]

Master Secret Key
(MSK)

Central Authority

Public Parameter
(PP)

One public key

dk_A

"Alice"

$m \leftarrow \mathrm{Dec}(dk_A, c)$

Alice

$c \leftarrow \mathrm{Enc}(\mathrm{PP}, \text{"Alice"}, m)$

Bob

$\mathrm{Enc}(\mathrm{PP}, \text{"Christine"}, m)$

Christine

$\mathrm{Enc}(\mathrm{PP}, \text{"Susan"}, m)$

Susan

The Key Escrow Problem with IBE

Master Secret Key
(MSK)

Central Authority

Public Parameter
(PP)

Too much power!
It can decrypt everything.

dk_A

"Alice"

$m \leftarrow \mathrm{Dec}(dk_A, c)$

Alice

$c \leftarrow \mathrm{Enc}(\mathrm{PP}, \text{"Alice"}, m)$

Bob

Registration Based Encryption (RBE) [GHMR18]
Aiming for compact public parameter without key escrow

Identities generate their secret keys and register their public keys.

id_1

id_2

id_3

Key Curator (KC)

Registration Based Encryption (RBE) [GHMR18]
Aiming for compact public parameter without key escrow

Identities generate their secret keys and register their public keys.

$(pk_1, sk_1) \leftarrow \text{Gen}(1^\lambda)$

id_1

$(pk_2, sk_2) \leftarrow \text{Gen}(1^\lambda)$

id_2

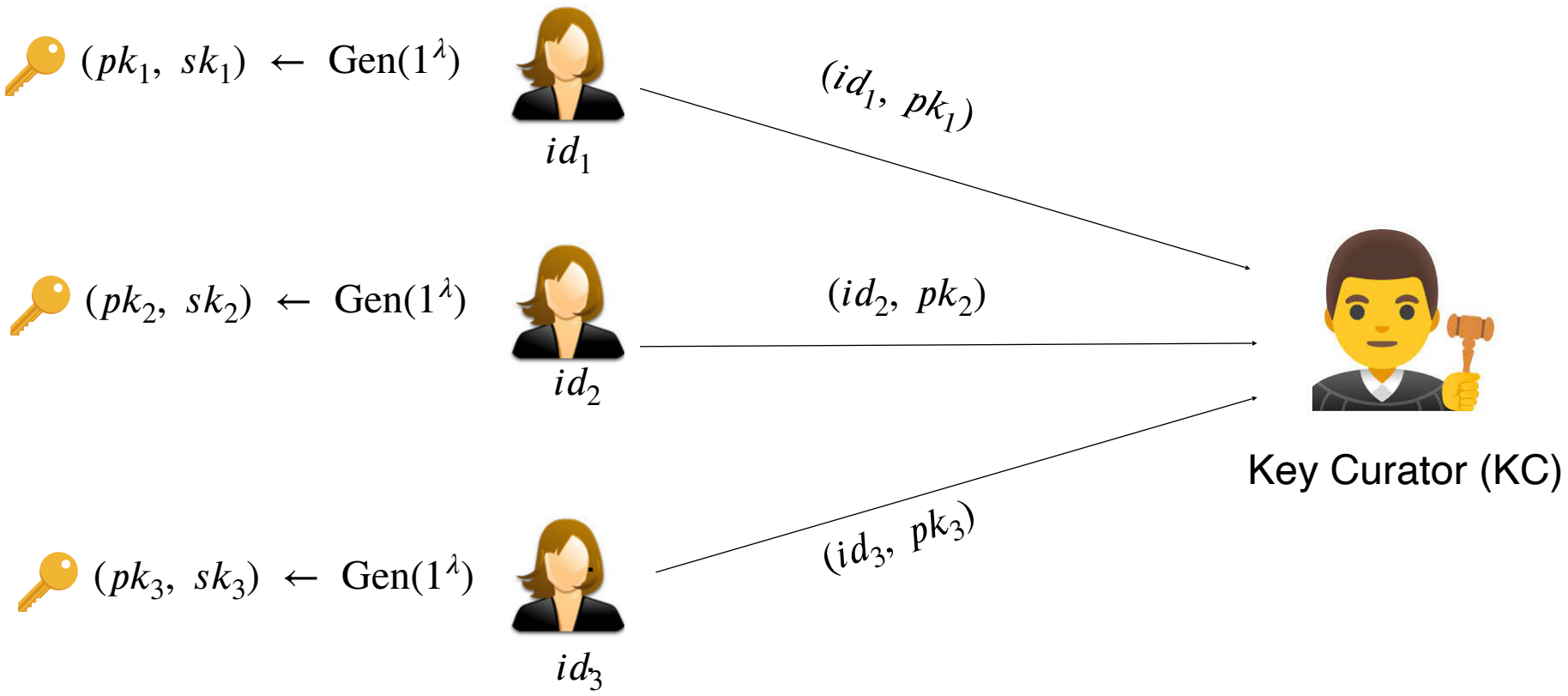
$(pk_3, sk_3) \leftarrow \text{Gen}(1^\lambda)$

id_3

Key Curator (KC)

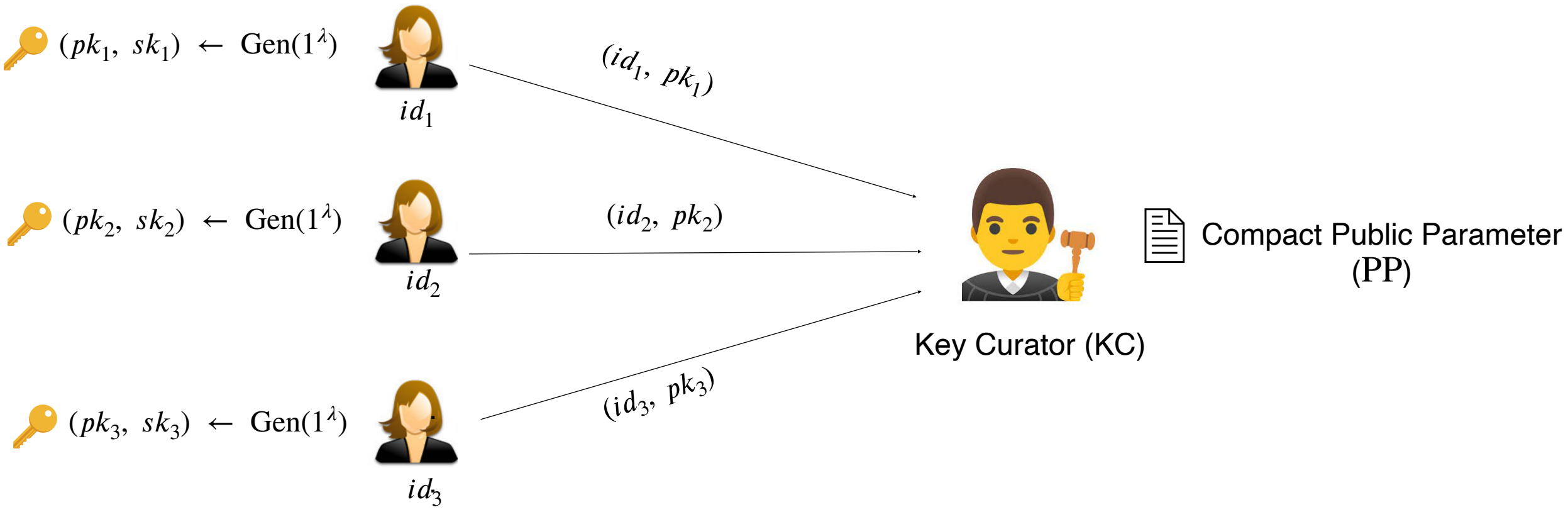
Registration Based Encryption (RBE) [GHMR18]
Aiming for compact public parameter without key escrow

Identities generate their secret keys and register their public keys.



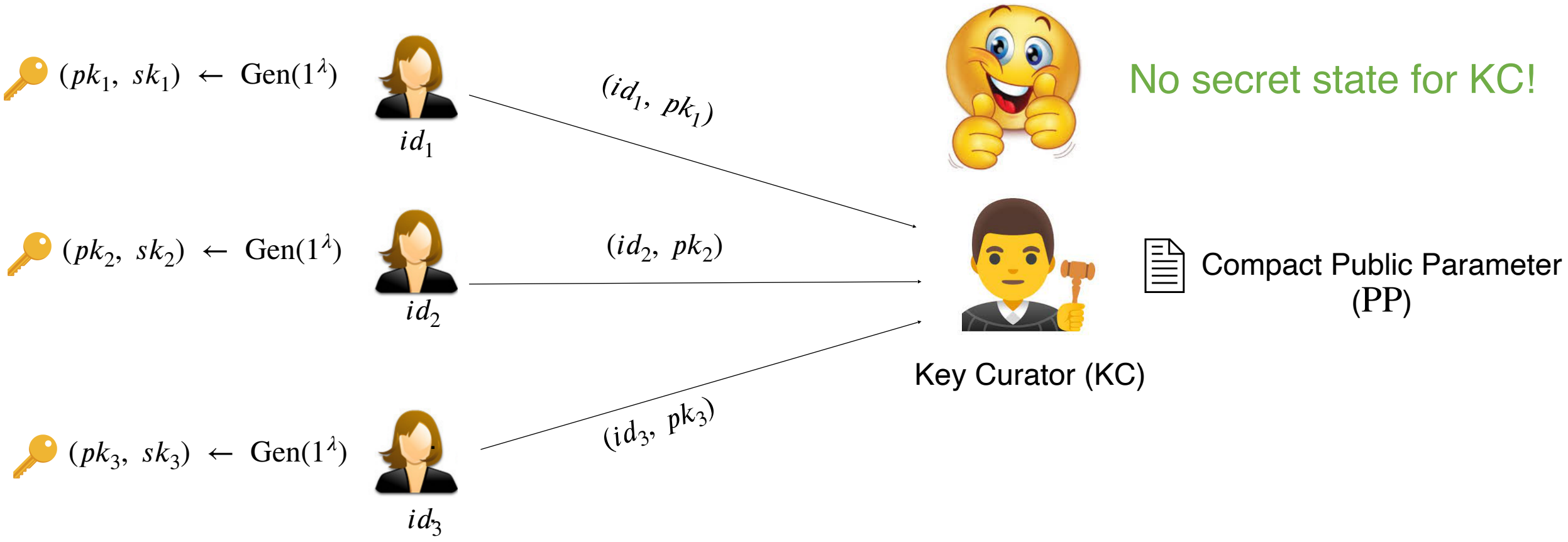
Registration Based Encryption (RBE) [GHMR18]
Aiming for compact public parameter without key escrow

Identities generate their secret keys and register their public keys.



Registration Based Encryption (RBE) [GHMR18]
Aiming for compact public parameter without key escrow

Identities generate their secret keys and register their public keys.



Registration Based Encryption (RBE) [GHMR18]

More formally:

Key Curator

Registration Based Encryption (RBE) [GHMR18]

More formally:

$(pk_1,\ sk_1) \leftarrow \mathrm{Gen}(1^\lambda)$

id_1

$(id_1,\ pk_1)$

Key Curator

Registration Based Encryption (RBE) [GHMR18]

More formally:

$(pk_1,\ sk_1) \leftarrow \mathrm{Gen}(1^\lambda)$

id_1

$(id_1,\ pk_1)$

Key Curator

$\mathrm{Reg}(\perp,\ id_1,\ pk_1) \rightarrow \mathrm{PP}_1$

Registration Based Encryption (RBE) [GHMR18]

More formally:

$(pk_1,\ sk_1) \leftarrow \mathrm{Gen}(1^\lambda)$

id_1

$(id_1,\ pk_1)$

Key Curator

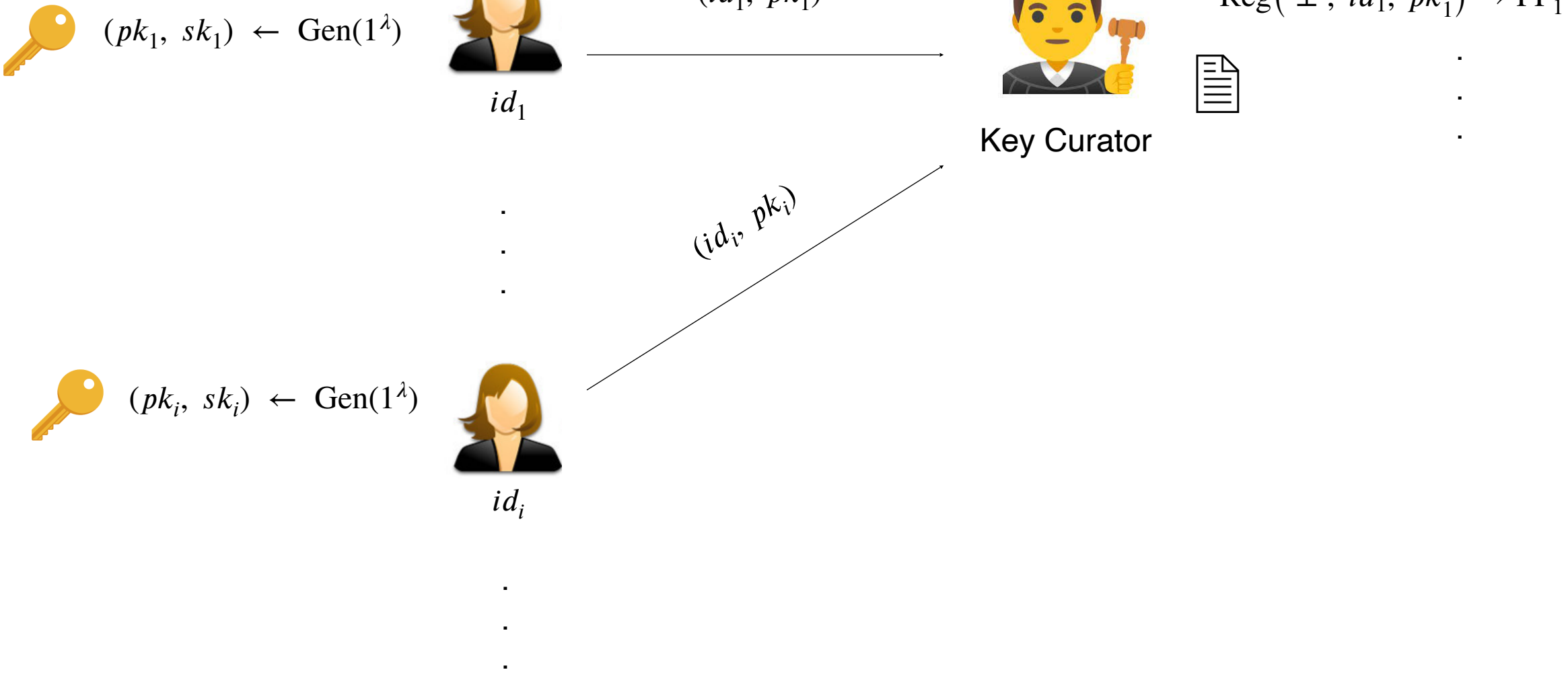
$\mathrm{Reg}(\perp,\ id_1,\ pk_1) \rightarrow \mathrm{PP}_1$

. . .

. . .

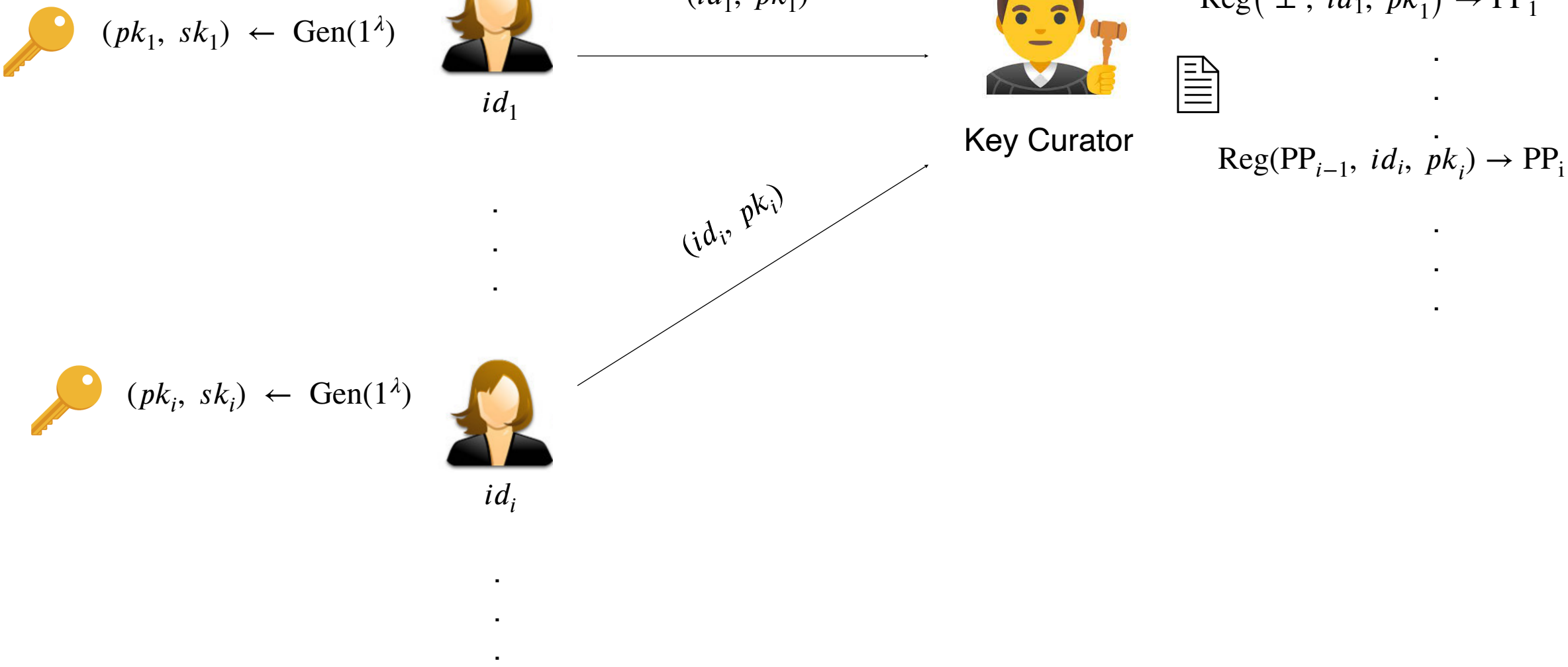
Registration Based Encryption (RBE) [GHMR18]

More formally:



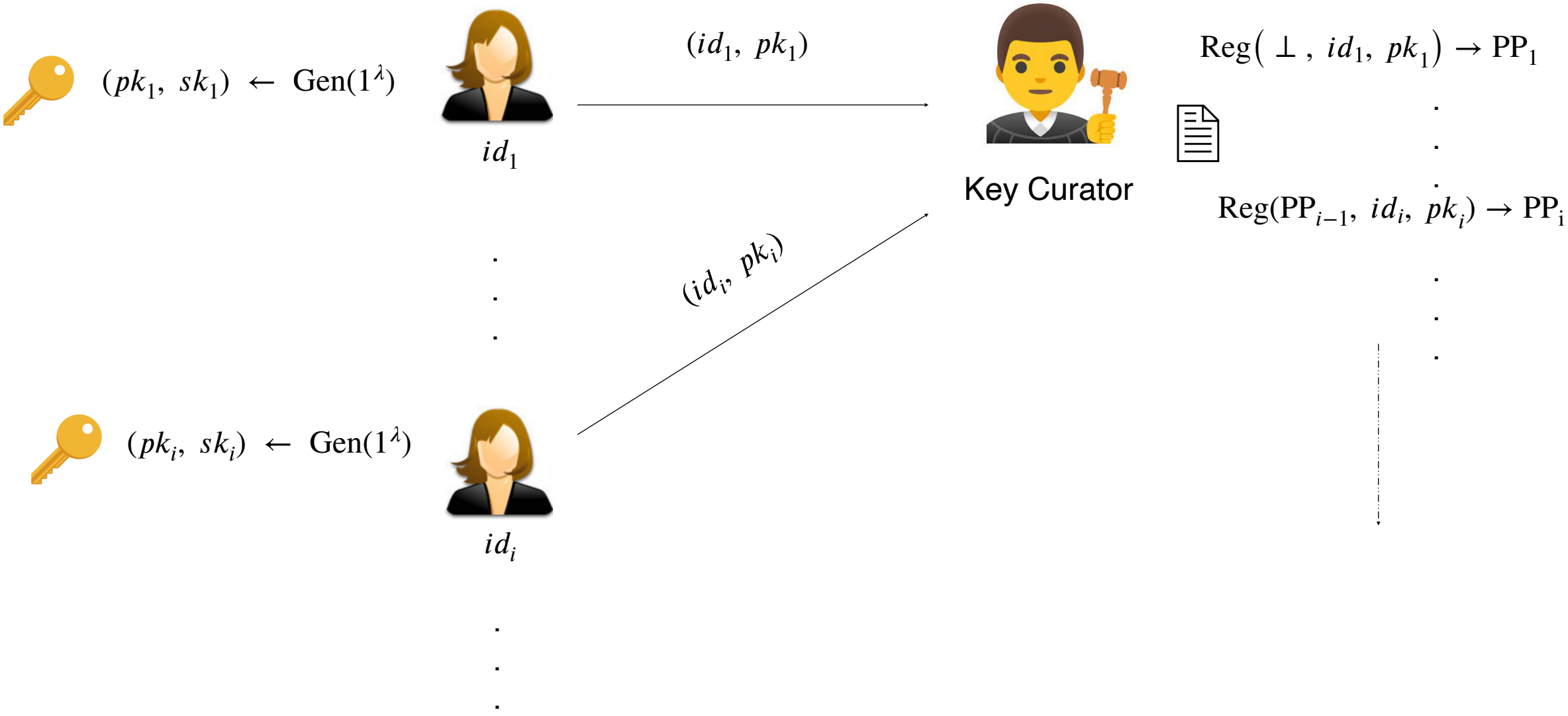
Registration Based Encryption (RBE) [GHMR18]

More formally:



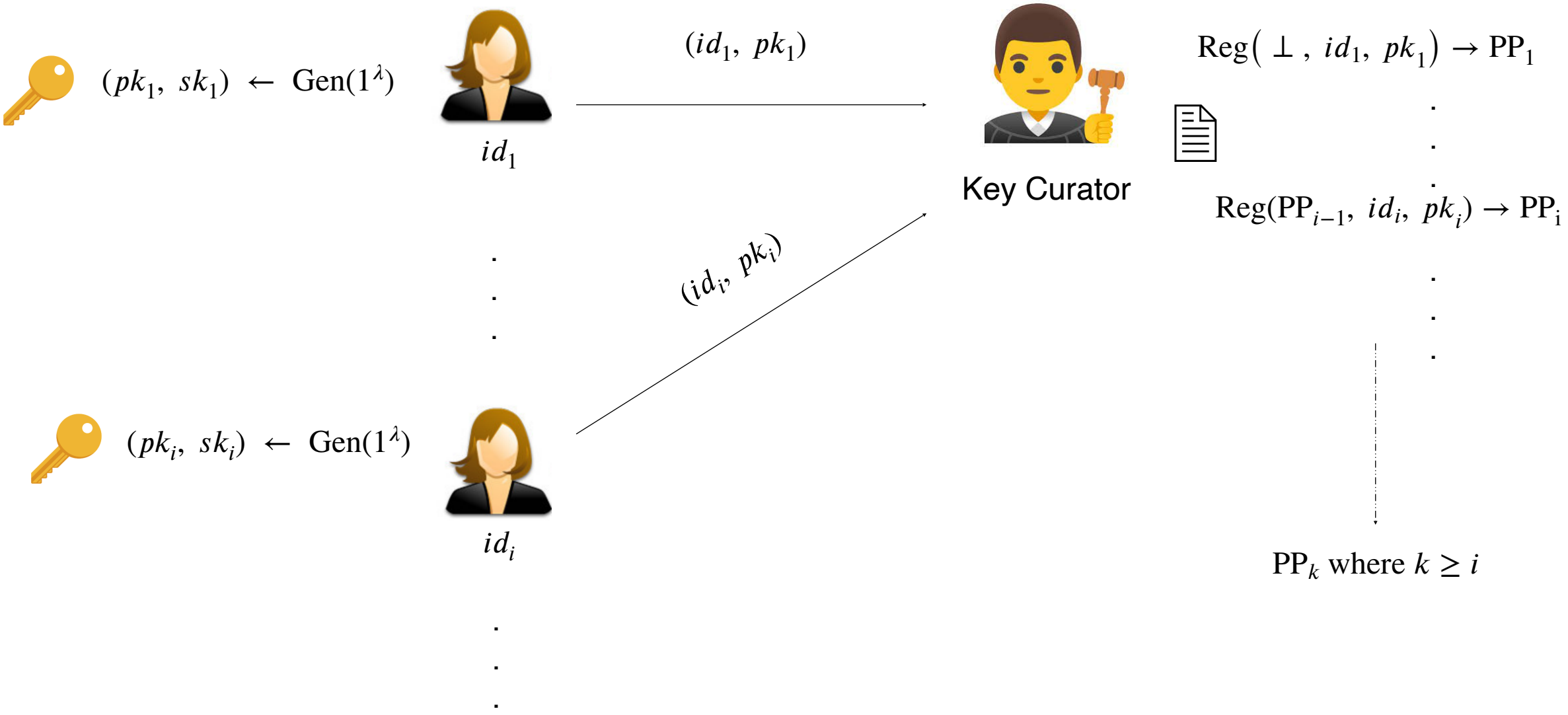
Registration Based Encryption (RBE) [GHMR18]

More formally:



Registration Based Encryption (RBE) [GHMR18]

More formally:



Registration Based Encryption (RBE) [GHMR18]

More formally:

$(pk_1,\ sk_1) \leftarrow \text{Gen}(1^\lambda)$

id_1

$(id_1,\ pk_1)$

Key Curator

$\text{Reg}(\perp,\ id_1,\ pk_1) \rightarrow \text{PP}_1$

$\text{Reg}(\text{PP}_{i-1},\ id_i,\ pk_i) \rightarrow \text{PP}_i$

$(id_i,\ pk_i)$

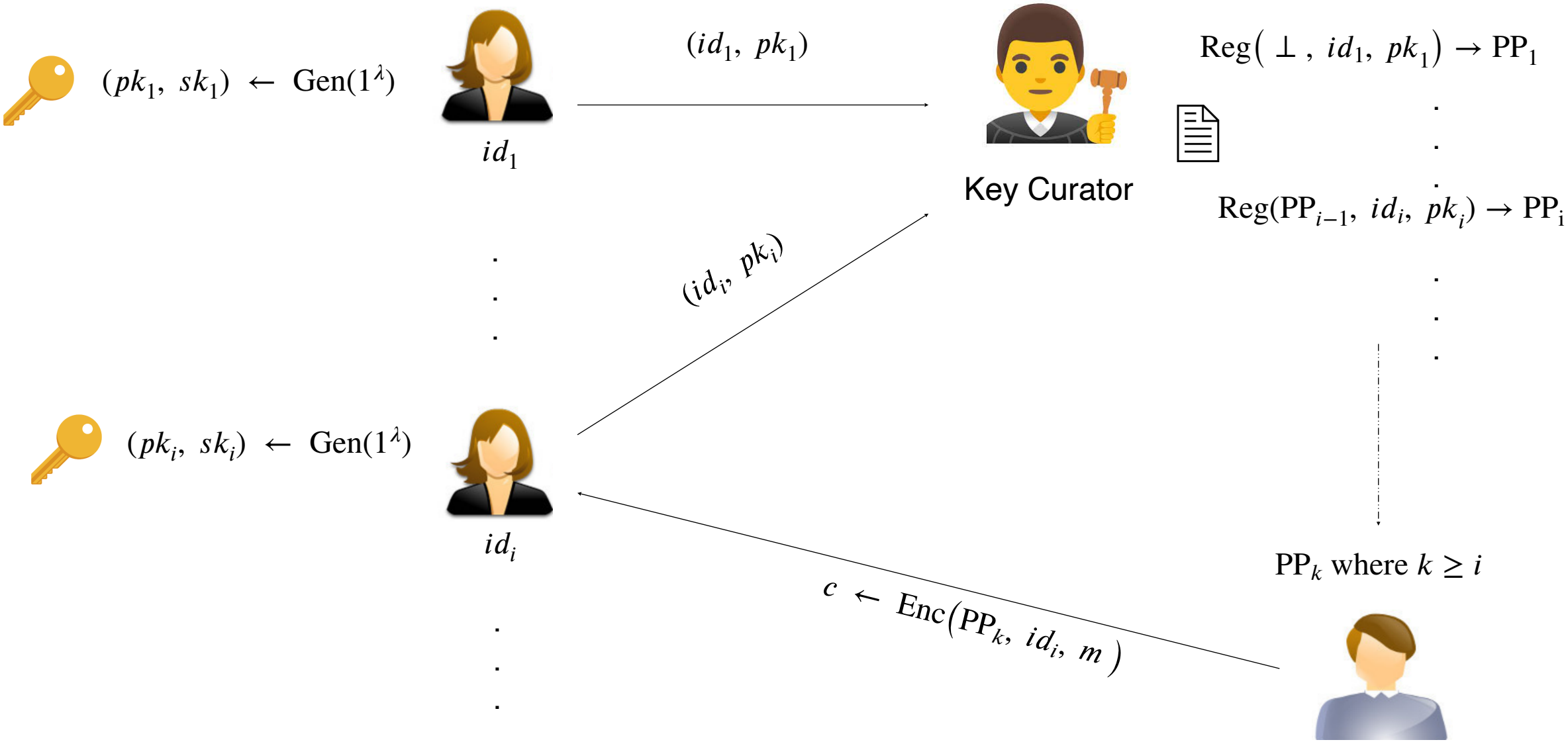
$(pk_i,\ sk_i) \leftarrow \text{Gen}(1^\lambda)$

id_i

PP_k where $k \geq i$

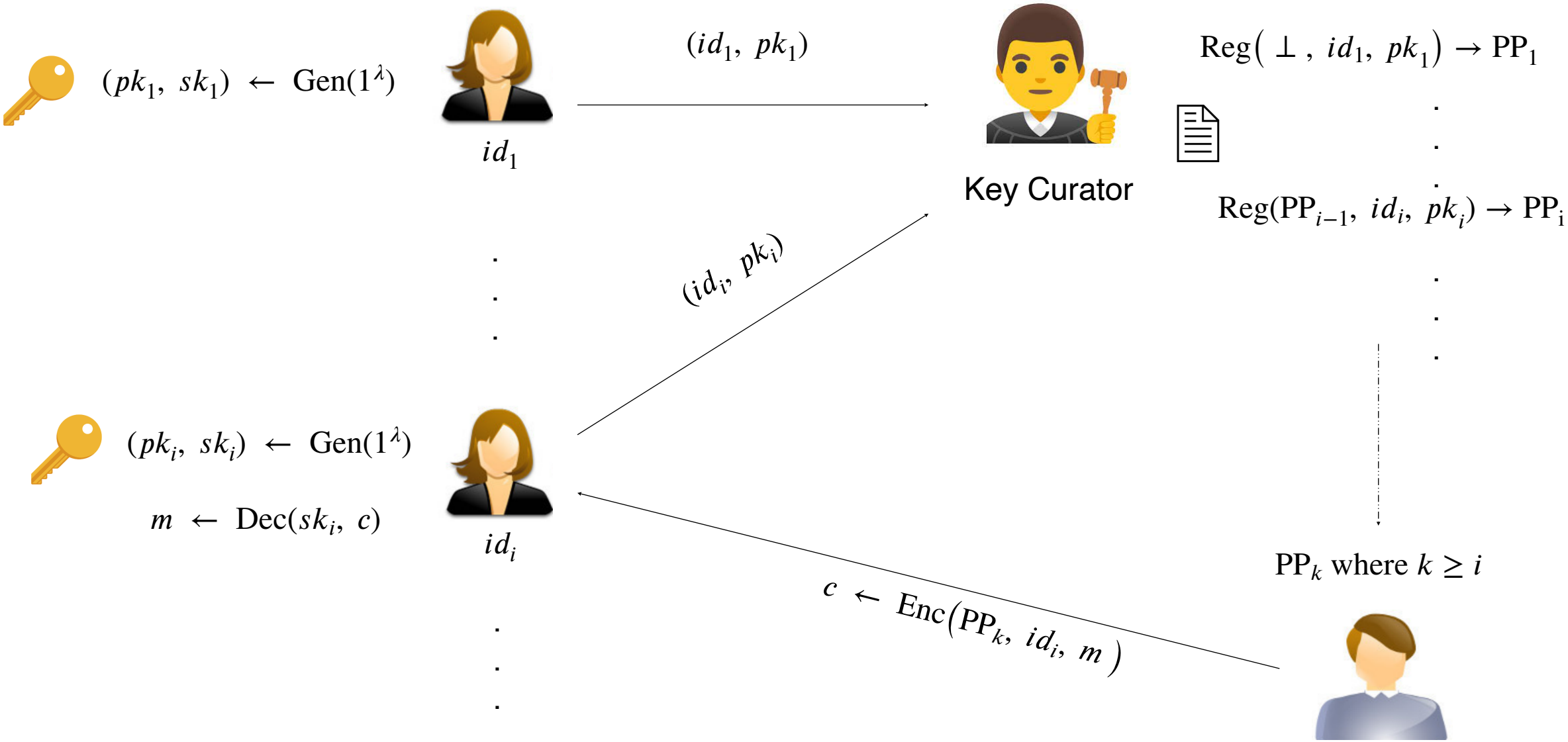
Registration Based Encryption (RBE) [GHMR18]

More formally:



Registration Based Encryption (RBE) [GHMR18]

More formally:



Registration Based Encryption (RBE) [GHMR18]

Decryption updates:

$(pk_1,\ sk_1) \leftarrow \text{Gen}(1^\lambda)$

id_1

$(id_1,\ pk_1)$

Key Curator

$\text{Reg}(\perp,\ id_1,\ pk_1) \rightarrow \text{PP}_1$

$\text{Reg}(\text{PP}_{i-1},\ id_i,\ pk_i) \rightarrow \text{PP}_i$

$(id_i,\ pk_i)$

$(pk_i,\ sk_i) \leftarrow \text{Gen}(1^\lambda)$

$m \leftarrow \text{Dec}(sk_i,\ c)$

id_i

$c \leftarrow \text{Enc}(\text{PP}_k,\ id_i,\ m)$

PP_k where $k \geq i$

Registration Based Encryption (RBE) [GHMR18]

Decryption updates:

$(pk_1,\ sk_1) \leftarrow \mathrm{Gen}(1^\lambda)$

id_1

$(id_1,\ pk_1)$

Key Curator

$\mathrm{Reg}(\perp,\ id_1,\ pk_1) \rightarrow \mathrm{PP}_1$

$\mathrm{Reg}(\mathrm{PP}_{i-1},\ id_i,\ pk_i) \rightarrow \mathrm{PP}_i$

$(id_i,\ pk_i)$

u : decryption update

$(pk_i,\ sk_i) \leftarrow \mathrm{Gen}(1^\lambda)$

$m \leftarrow \mathrm{Dec}(sk_i,\ c)$

id_i

$c \leftarrow \mathrm{Enc}(\mathrm{PP}_k,\ id_i,\ m)$

PP_k where $k \geq i$

Registration Based Encryption (RBE) [GHMR18]

Decryption updates:

$(pk_1,\ sk_1) \leftarrow \mathrm{Gen}(1^\lambda)$

id_1

$(id_1,\ pk_1)$

Key Curator

$\mathrm{Reg}(\perp,\ id_1,\ pk_1) \rightarrow \mathrm{PP}_1$

$\mathrm{Reg}(\mathrm{PP}_{i-1},\ id_i,\ pk_i) \rightarrow \mathrm{PP}_i$

$(id_i,\ pk_i)$

u : decryption update

$(pk_i,\ sk_i) \leftarrow \mathrm{Gen}(1^\lambda)$

id_i

$m \leftarrow \mathrm{Dec}(sk_i,\ c)$

$m \leftarrow \mathrm{Dec}(sk_i,\ c, u)$

$c \leftarrow \mathrm{Enc}(\mathrm{PP}_k,\ id_i,\ m)$

PP_k where $k \geq i$

Registration Based Encryption (RBE) [GHMR18]

Decryption updates:

$(pk_1,\ sk_1) \leftarrow \mathrm{Gen}(1^\lambda)$

id_1

$(id_1,\ pk_1)$

Key Curator

$\mathrm{Reg}(\perp,\ id_1,\ pk_1) \rightarrow \mathrm{PP}_1$

$\mathrm{Reg}(\mathrm{PP}_{i-1},\ id_i,\ pk_i) \rightarrow \mathrm{PP}_i$

$(id_i,\ pk_i)$

u : decryption update

$(pk_i,\ sk_i) \leftarrow \mathrm{Gen}(1^\lambda)$

id_i

$m \leftarrow \mathrm{Dec}(sk_i,\ c)$

$m \leftarrow \mathrm{Dec}(sk_i,\ c, u)$

$c \leftarrow \mathrm{Enc}(\mathrm{PP}_k,\ id_i,\ m)$

PP_k where $k \geq i$

Registration Based Encryption (RBE) [GHMR18]

Decryption updates:

$(pk_1,\ sk_1) \leftarrow \mathrm{Gen}(1^\lambda)$

id_1

$(id_1,\ pk_1)$

Key Curator

$\mathrm{Reg}(\perp,\ id_1,\ pk_1) \rightarrow \mathrm{PP}_1$

$\mathrm{Reg}(\mathrm{PP}_{i-1},\ id_i,\ pk_i) \rightarrow \mathrm{PP}_i$

$(id_i,\ pk_i)$

u : decryption update

$(pk_i,\ sk_i) \leftarrow \mathrm{Gen}(1^\lambda)$

id_i

$m \leftarrow \mathrm{Dec}(sk_i,\ c)$

$m \leftarrow \mathrm{Dec}(sk_i,\ c, u)$

$c \leftarrow \mathrm{Enc}(\mathrm{PP}_k,\ id_i,\ m)$

PP_k where $k \geq i$

RBE security : similar to IBE:

No colluding parties can decrypt others' messages.

Efficiency & Compactness Requirements of RBE

Let n := number of registered identities

We need:

Efficiency & Compactness Requirements of RBE

Let n := number of registered identities

We need:

- Compact Public Parameter: $|\mathrm{PP}| \leq \mathrm{polylog}(n)$

Efficiency & Compactness Requirements of RBE

Let $n :=$ number of registered identities

We need:

- Compact Public Parameter: $|\mathrm{PP}| \leq \mathrm{polylog}(n)$
- Rare updates: Number of updates $\leq O(\log n)$

Efficiency & Compactness Requirements of RBE

Let $n :=$ number of registered identities

We need:

- Compact Public Parameter: $|\mathrm{PP}| \leq \mathrm{polylog}(n)$
- Rare updates: Number of updates $\leq O(\log n)$

Moreover:

Efficiency & Compactness Requirements of RBE

Let $n :=$ number of registered identities

We need:

- Compact Public Parameter: $|\mathrm{PP}| \leq \mathrm{polylog}(n)$
- Rare updates: Number of updates $\leq O(\log n)$

Moreover:

- Updates are compact and have size $\leq \mathrm{polylog}(n)$
- Generating updates and registration can be done in time $\mathrm{polylog}(n)$

Main Question

Are $\approx \log n$ decryption updates necessary?

Main Question

Are $\approx \log n$ decryption updates necessary?

All known constructions require $\Omega(\log n)$ updates.

Our Result: almost tight lower bound

- $\frac{\log n}{\log\log n}$ updates are needed in RBE.

Our Result: almost tight lower bound

- $\frac{\log n}{\log\log n}$ updates are needed in RBE.

Assumption: We assume updates arrive at fixed times.

Our Result: almost tight lower bound

- $\frac{\log n}{\log\log n}$ updates are needed in RBE.

Assumption: We assume updates arrive at fixed times.

Satisfied by all known constructions.

Our Result: almost tight lower bound

- $\frac{\log n}{\log\log n}$ updates are needed in RBE.

Assumption: We assume updates arrive at fixed times.

Satisfied by all known constructions.

- More generally, if number of identities $\geq \binom{\ell + d}{d+1}$ and $\leq d$ updates:

Our Result: almost tight lower bound

- $\frac{\log n}{\log\log n}$ updates are needed in RBE.

Assumption: We assume updates arrive at fixed times.

Satisfied by all known constructions.

- More generally, if number of identities $\geq \binom{\ell + d}{d+1}$ and $\leq d$ updates:

Then length of public parameter $\left|\mathrm{PP}\right| \geq \Omega(\ell)$.

Our Result: almost tight lower bound

- $\frac{\log n}{\log\log n}$ updates are needed in RBE.

Assumption: We assume updates arrive at fixed times.

Satisfied by all known constructions.

- More generally, if number of identities $\geq \binom{\ell + d}{d+1}$ and $\leq d$ updates:

Then length of public parameter $\left|\mathrm{PP}\right| \geq \Omega(\ell)$.

Our Result: almost tight lower bound

- $\frac{\log n}{\log\log n}$ updates are needed in RBE.

⚠ Assumption: We assume updates arrive at fixed times.

✓ Satisfied by all known constructions.

- More generally, if number of identities $\geq \binom{\ell + d}{d+1}$ and $\leq d$ updates:

Then length of public parameter $\left|\mathrm{PP}\right| \geq \Omega(\ell)$.

- Remark: Result holds even if some parties get many updates.
  ** Thanks to the reviewer for asking if this extension holds

Our Result: almost tight lower bound

- $\frac{\log n}{\log\log n}$ updates are needed in RBE.

⚠ Assumption: We assume updates arrive at fixed times.

✓ Satisfied by all known constructions.

- More generally, if number of identities $\geq \binom{\ell + d}{d+1}$ and $\leq d$ updates:

Then length of public parameter $\left|\mathrm{PP}\right| \geq \Omega(\ell)$.

- Remark: Result holds even if some parties get many updates.
  ** Thanks to the reviewer for asking if this extension holds

Outline

- Introduction
- Our Results
- Proof Ideas

High Level Steps of the Proof

High Level Steps of the Proof

(1) Define a DAG based on times of updates

High Level Steps of the Proof

(1) Define a DAG based on times of updates

(2) Prove there exists a **structure** in such DAGs

High Level Steps of the Proof

(1) Define a DAG based on times of updates

(2) Prove there exists a **structure** in such DAGs

(3) Use the **structure** to attack successfully, using **info theory**

Step1: Defining a DAG based on updates

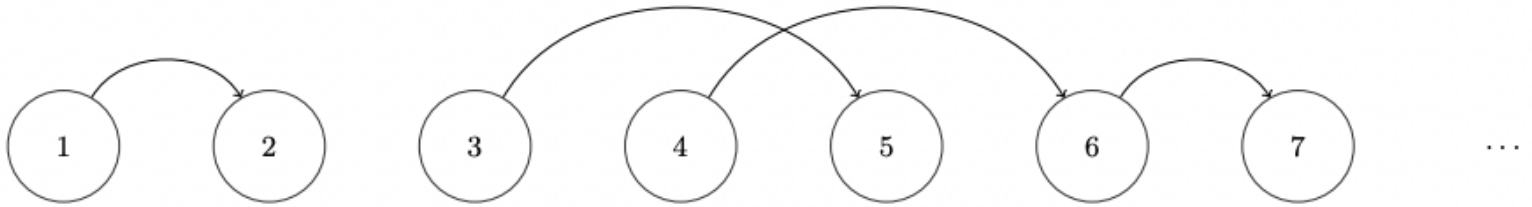
- n users → DAG of n vertices

Step1: Defining a DAG based on updates

- n users → DAG of n vertices
- i-th user → i-th vertex

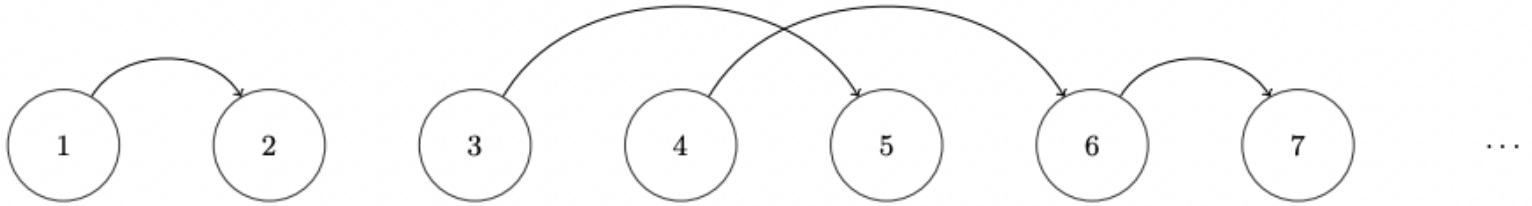
Step1: Defining a DAG based on updates

- n users → DAG of n vertices
- i-th user → i-th vertex



Step1: Defining a DAG based on updates

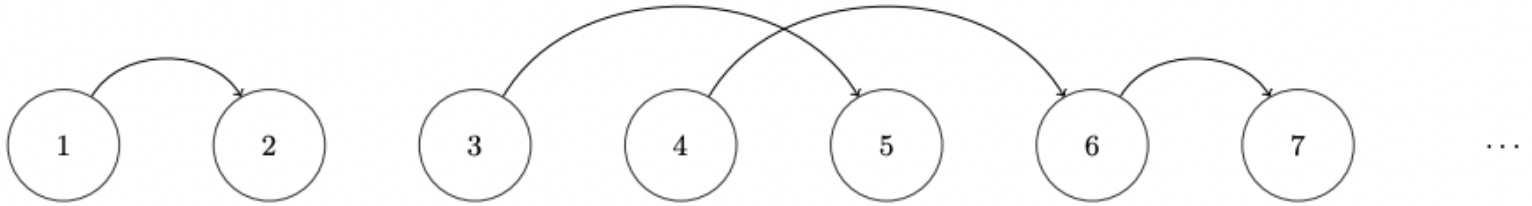
- n users → DAG of n vertices
- i-th user → i-th vertex



- Edge $(i \to j)$ exists iff user i needs an update after user j is registered.

Step1: Defining a DAG based on updates

- n users → DAG of n vertices
- i-th user → i-th vertex



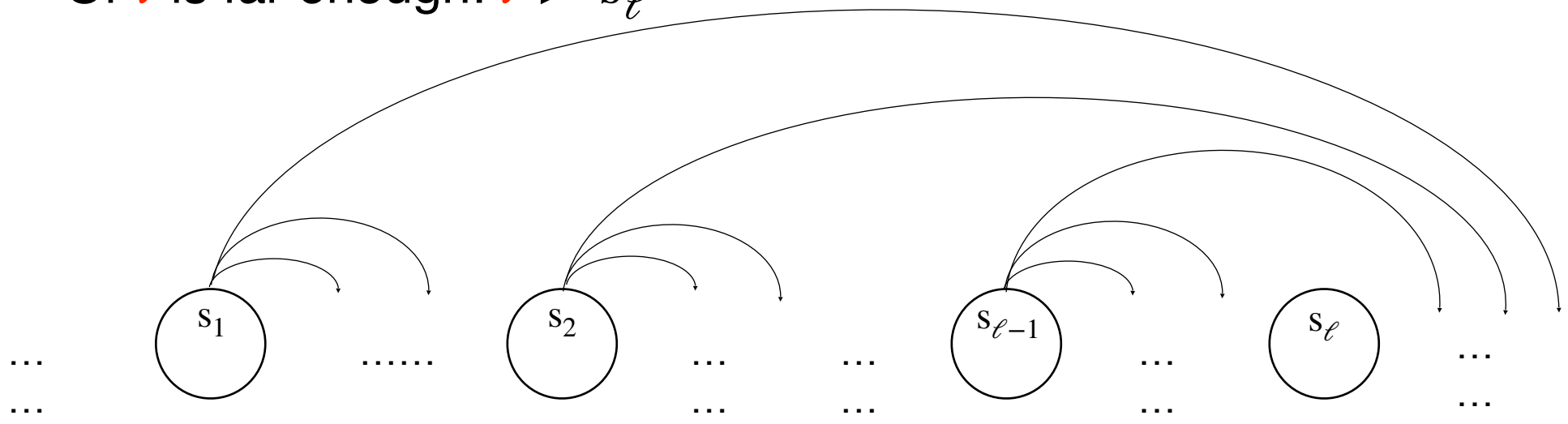
- Edge $(i \rightarrow j)$ exists iff user i needs an update after user j is registered.

By assumption, we know when users need update.

Step 2: Defining a Useful Structure in DAGs

- $\{s_1 < \ldots < s_\ell\}$ is a **skipping sequence** if

  for any outgoing edge (s_i, t)
  - Either t is close enough: $t < s_{i+1}$
  - Or t is far enough: $t > s_\ell$

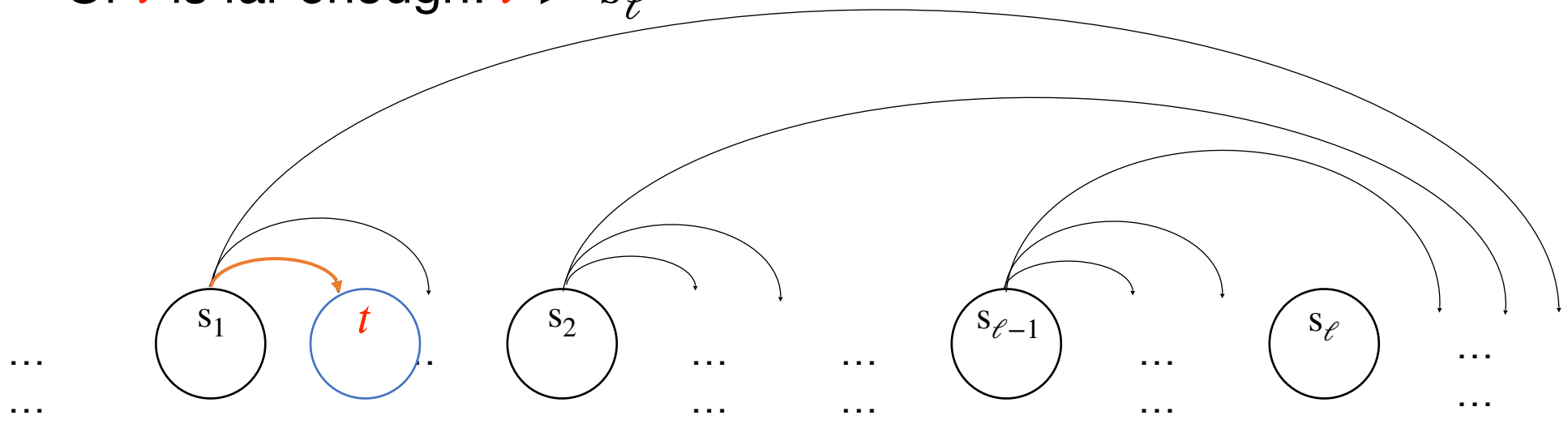


Step 2: Defining a Useful Structure in DAGs

- $\{s_1 < \ldots < s_\ell\}$ is a **skipping sequence** if

  for any outgoing edge (s_i, t)

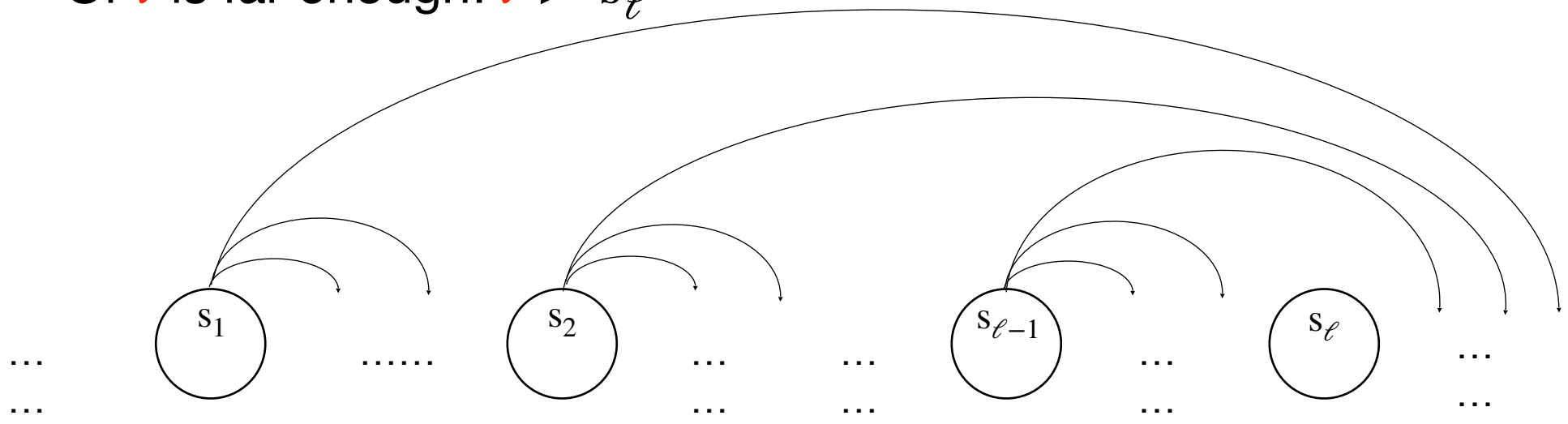
  - Either t is close enough: $t < s_{i+1}$
  - Or t is far enough: $t > s_\ell$



Step 2: Defining a Useful Structure in DAGs

- $\{s_1 < \ldots < s_\ell\}$ is a **skipping sequence** if

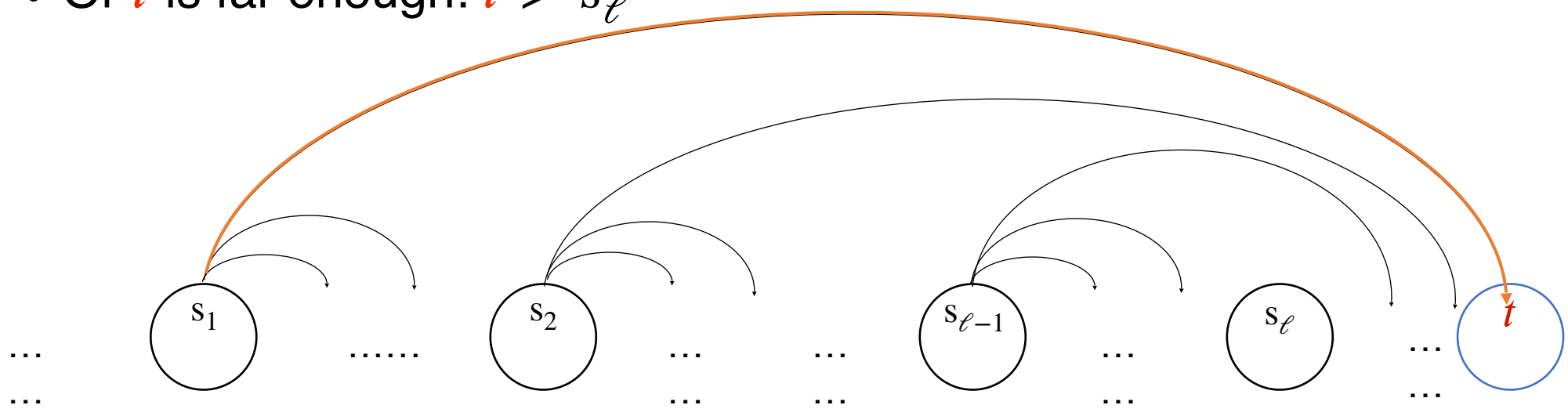
  for any outgoing edge (s_i, t)
  - Either t is close enough: $t < s_{i+1}$
  - Or t is far enough: $t > s_\ell$



Step 2: Defining a Useful Structure in DAGs

- $\{s_1 < \ldots < s_\ell\}$ is a **skipping sequence** if

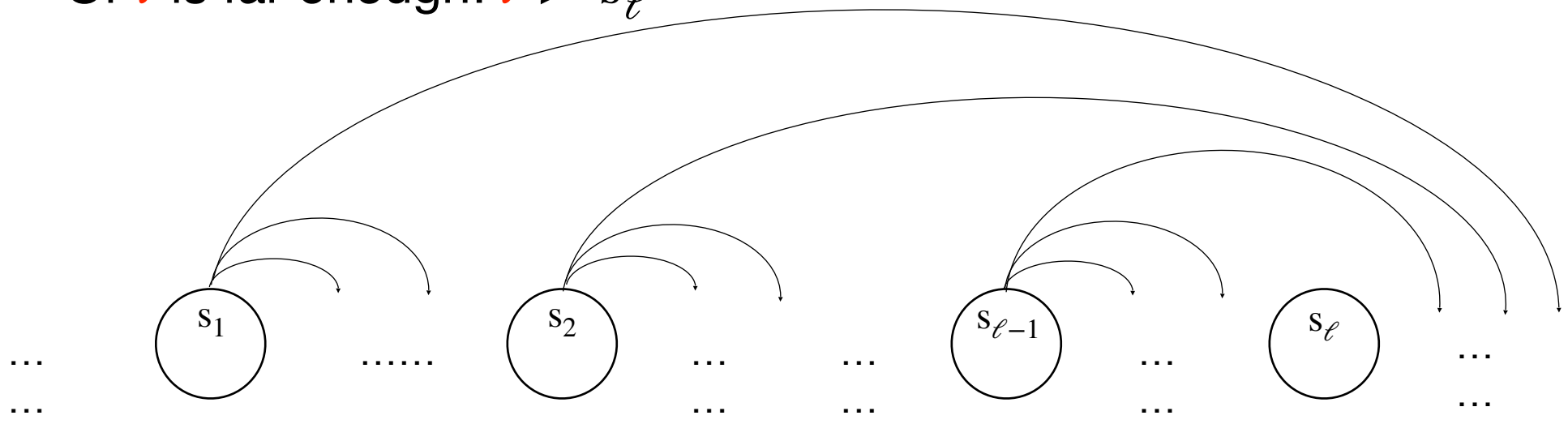
  for any outgoing edge (s_i, t)
  - Either t is close enough: $t < s_{i+1}$
  - Or t is far enough: $t > s_\ell$



Step 2: Defining a Useful Structure in DAGs

- $\{s_1 < \ldots < s_\ell\}$ is a **skipping sequence** if

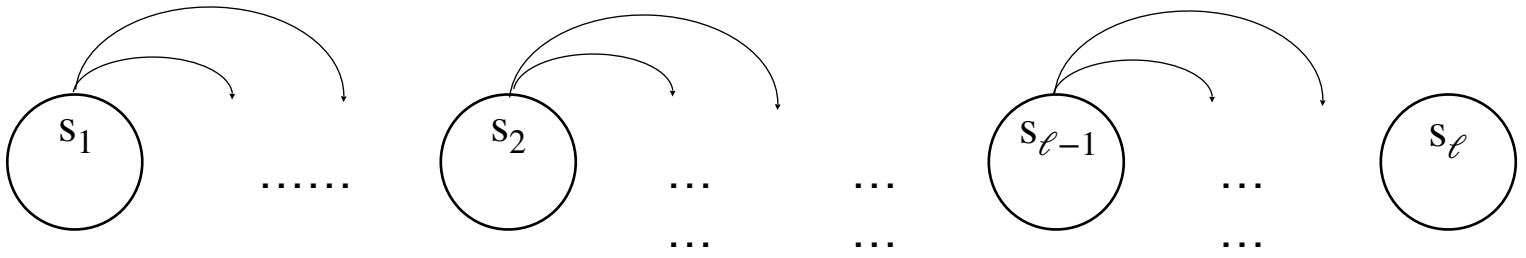
  for any outgoing edge (s_i, t)
  - Either t is close enough: $t < s_{i+1}$
  - Or t is far enough: $t > s_\ell$



Step 2: Defining a Useful Structure in DAGs

- $\{s_1 < \ldots < s_\ell\}$ is a **skipping sequence** if

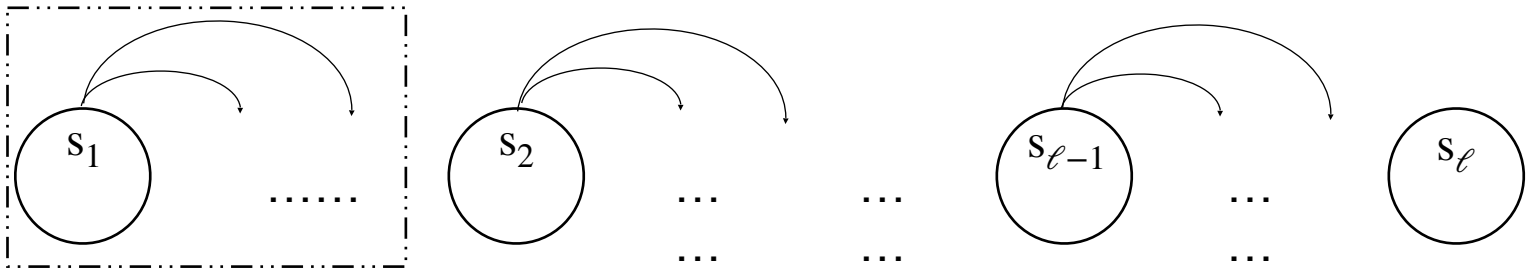
  for any outgoing edge (s_i, t)
  - Either t is close enough: $t < s_{i+1}$
  - Or t is far enough: $t > s_\ell$



Step 2: Defining a Useful Structure in DAGs

- $\{s_1 < \ldots < s_\ell\}$ is a **skipping sequence** if

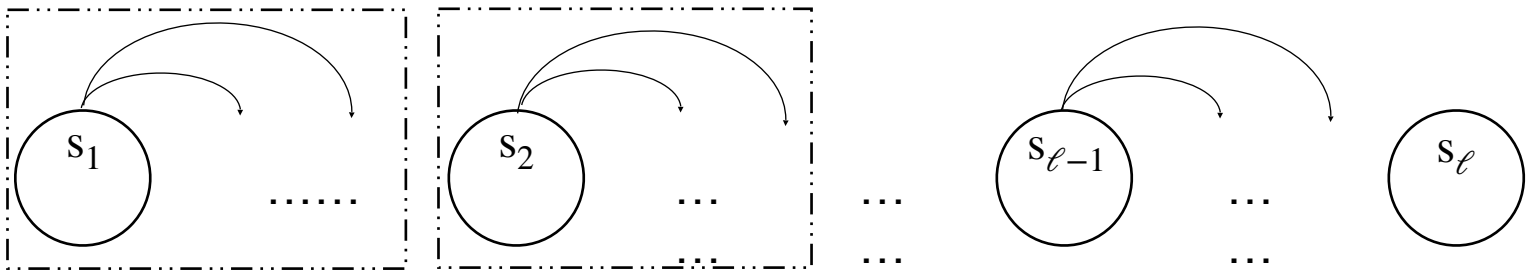
  for any outgoing edge (s_i, t)
  - Either t is close enough: $t < s_{i+1}$
  - Or t is far enough: $t > s_\ell$



Step 2: Defining a Useful Structure in DAGs

- $\{s_1 < \ldots < s_\ell\}$ is a **skipping sequence** if

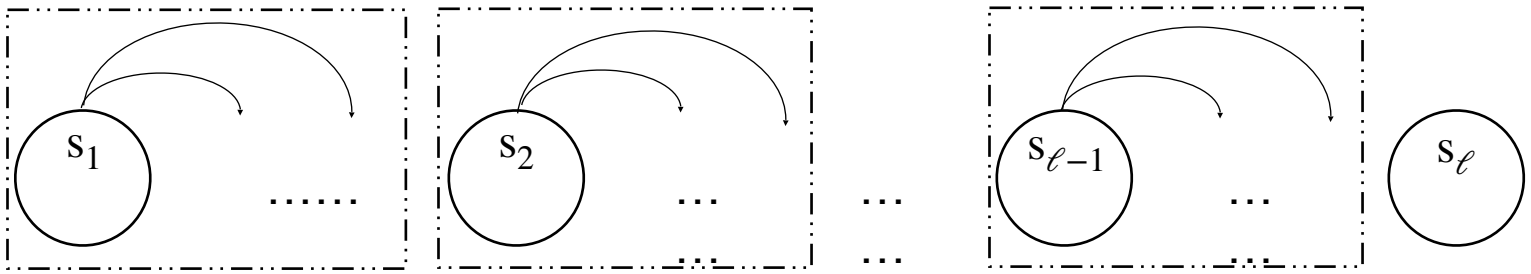
  for any outgoing edge (s_i, t)
  - Either t is close enough: $t < s_{i+1}$
  - Or t is far enough: $t > s_\ell$



Step 2: Defining a Useful Structure in DAGs

- $\{s_1 < \ldots < s_\ell\}$ is a **skipping sequence** if

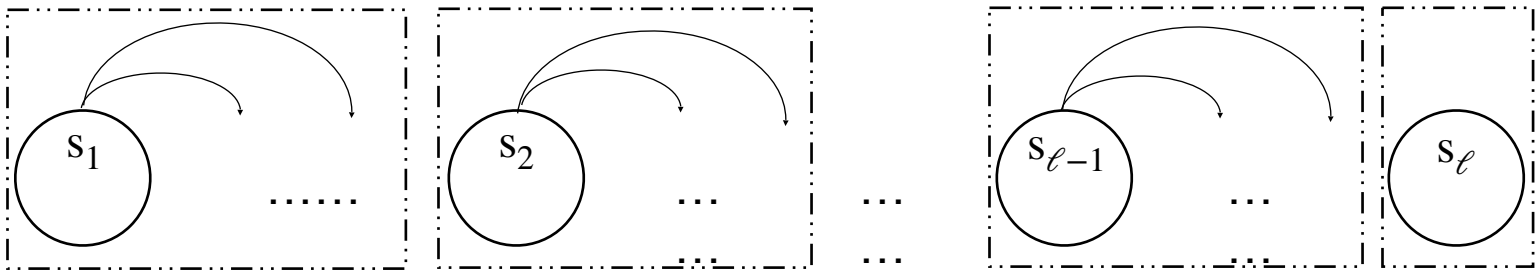
  for any outgoing edge (s_i, t)
  - Either t is close enough: $t < s_{i+1}$
  - Or t is far enough: $t > s_\ell$



Step 2: Defining a Useful Structure in DAGs

- $\{s_1 < \ldots < s_\ell\}$ is a **skipping sequence** if

  for any outgoing edge (s_i, t)
  - Either t is close enough: $t < s_{i+1}$
  - Or t is far enough: $t > s_\ell$



Step 2: Skipping Sequence Exists in Low-Degree DAGs

Step 2: Skipping Sequence Exists in Low-Degree DAGs

Theorem:

Step 2: Skipping Sequence Exists in Low-Degree DAGs

Theorem:

If G has $\geq \binom{\ell + d}{d + 1}$ vertices and out-degrees are all $\leq d$,

Step 2: Skipping Sequence Exists in Low-Degree DAGs

Theorem:

If G has $\geq \binom{\ell + d}{d+1}$ vertices and out-degrees are all $\leq d$,

then G has a skipping sequence of length $\geq \ell$

Step 2: Skipping Sequence Exists in Low-Degree DAGs

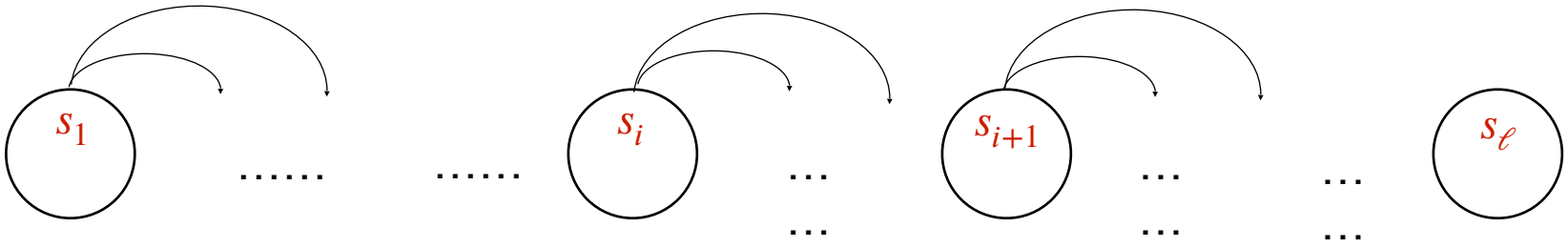
Theorem:

If G has $\geq \binom{\ell + d}{d+1}$ vertices and out-degrees are all $\leq d$,

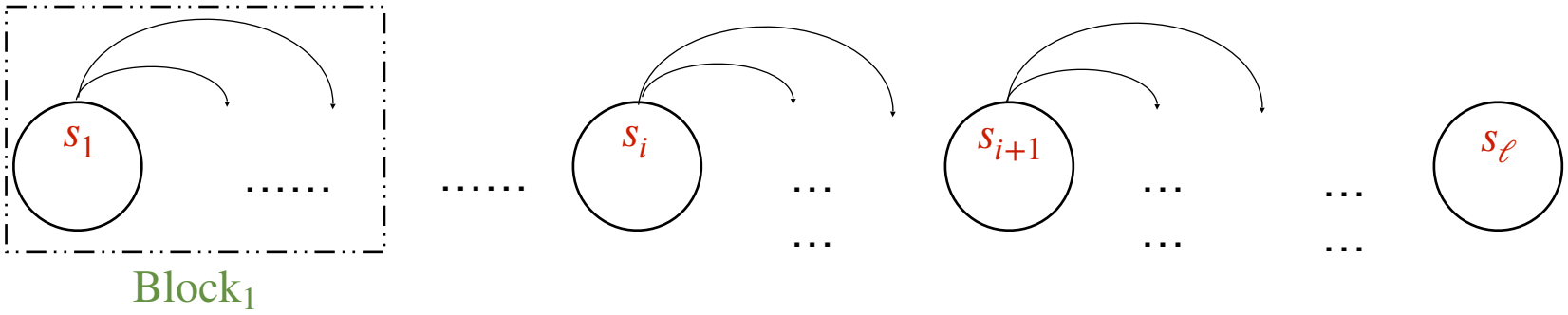
then G has a skipping sequence of length $\geq \ell$

Proof: Induction on d.

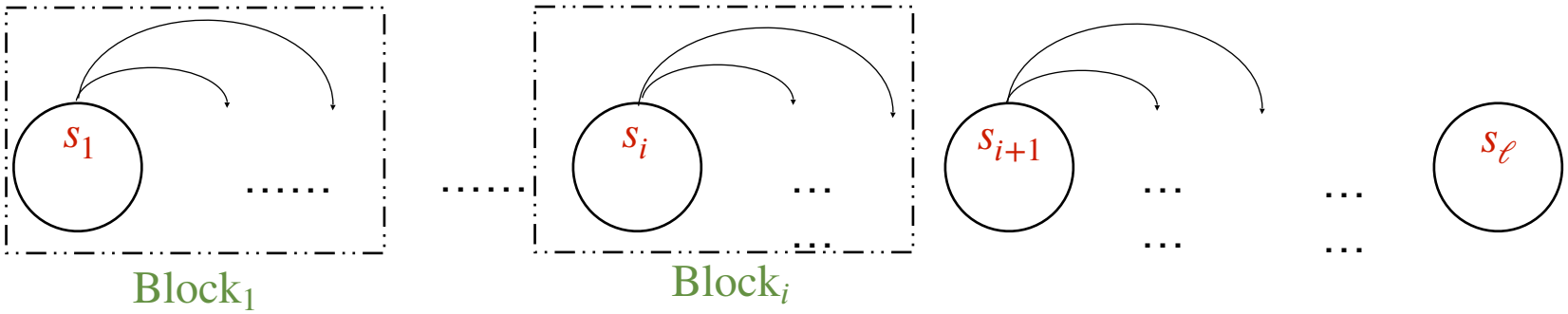
Step 3: Using skipping sequence to attack



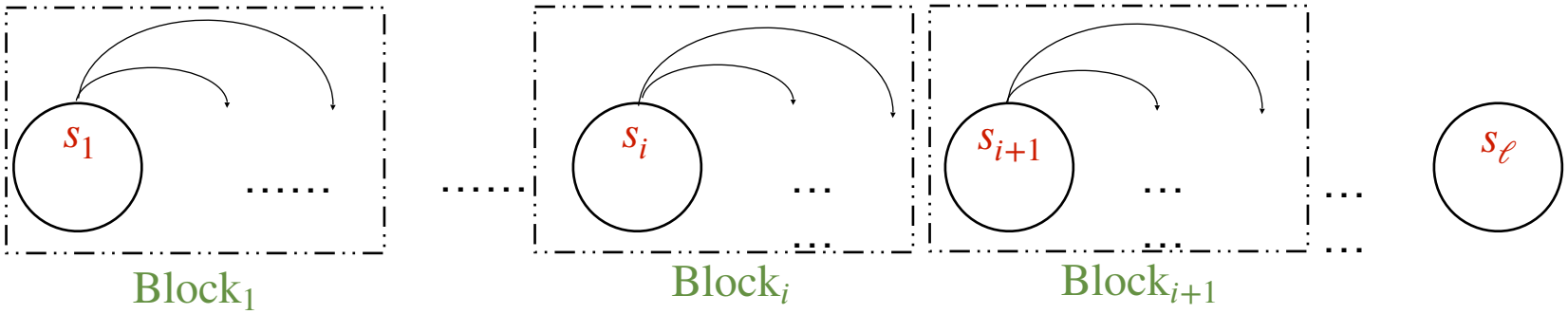
Step 3: Using skipping sequence to attack



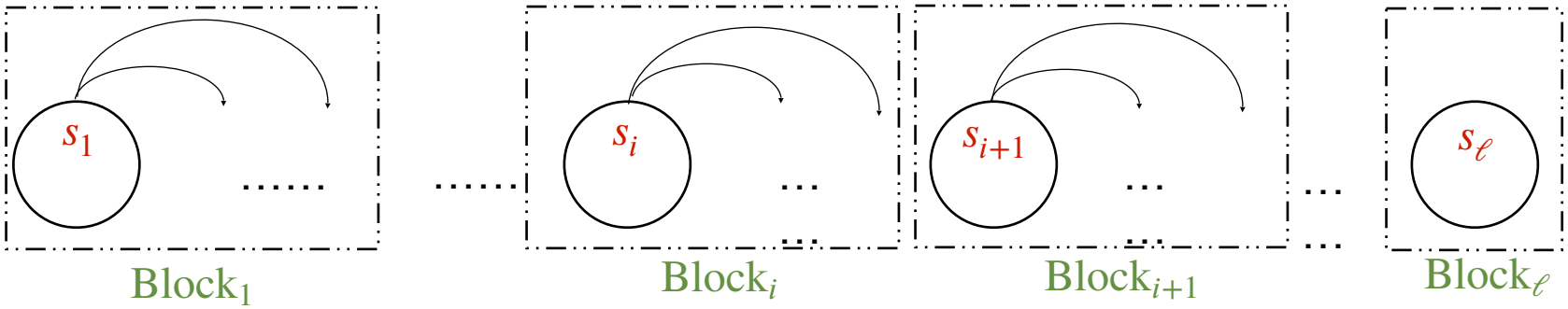
Step 3: Using skipping sequence to attack



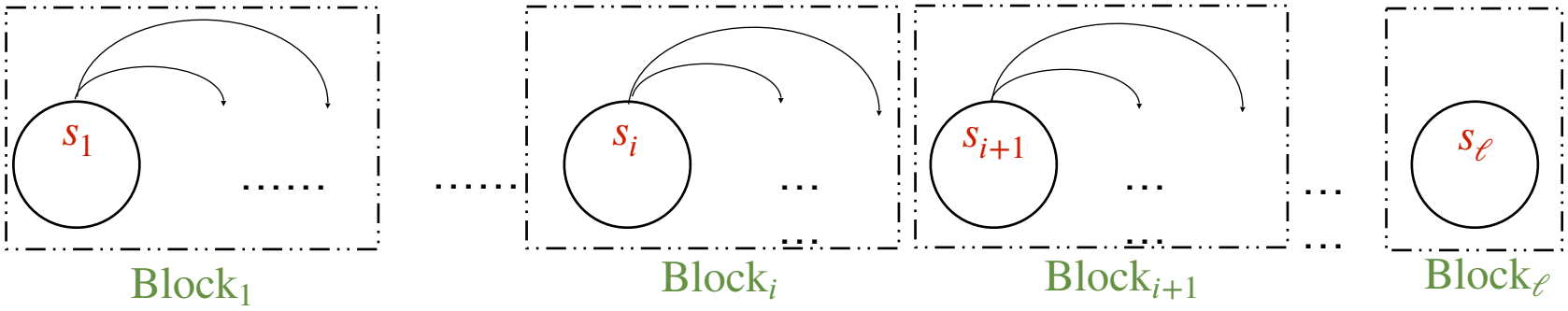
Step 3: Using skipping sequence to attack



Step 3: Using skipping sequence to attack

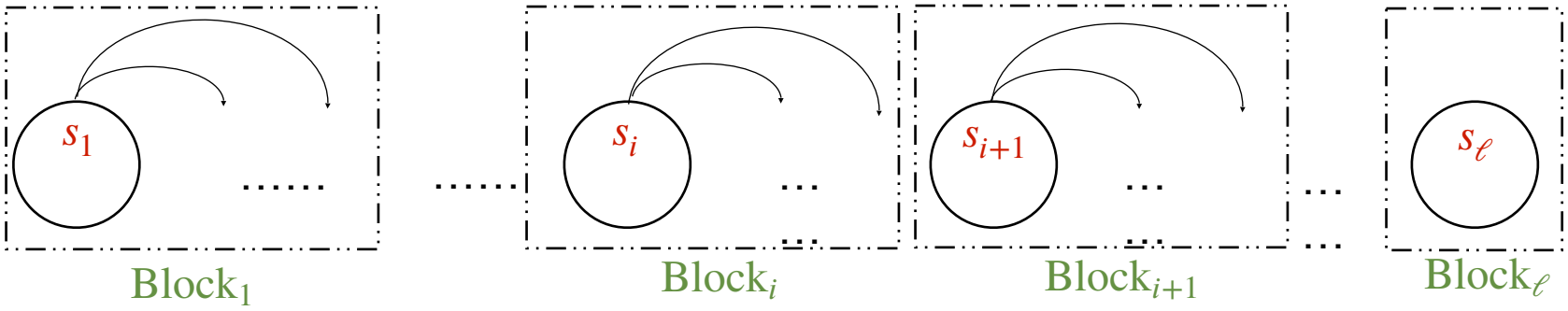


Step 3: Using skipping sequence to attack



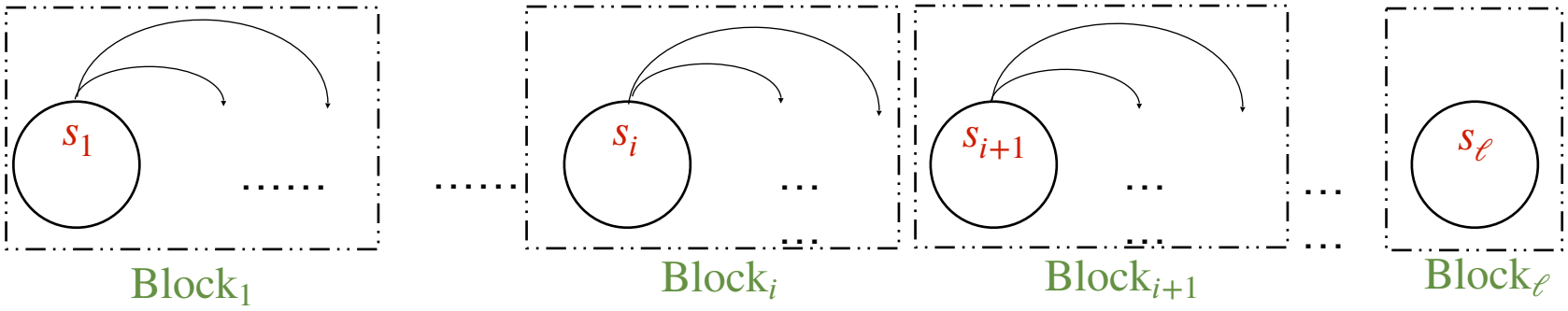
- Call Block_i good if: all the keys of users in Block_i are (almost) independent of PP_{s_ℓ}.

Step 3: Using skipping sequence to attack



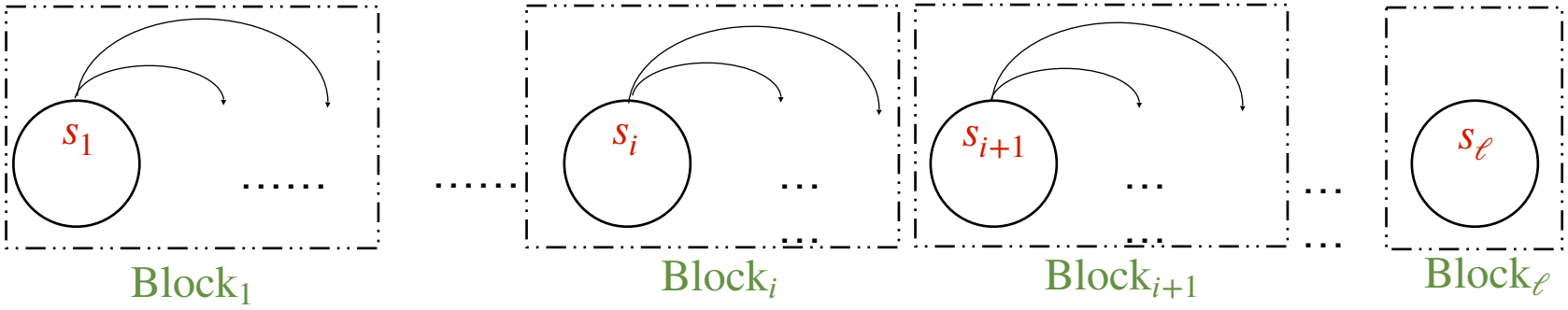
- Call Block_i good if: all the keys of users in Block_i are (almost) independent of PP_{s_ℓ}.
- Claim: If $\ell \gg |\text{PP}_{s_\ell}|$, then a good block exists (**info-theoretical argument**).

Step 3: Using skipping sequence to attack



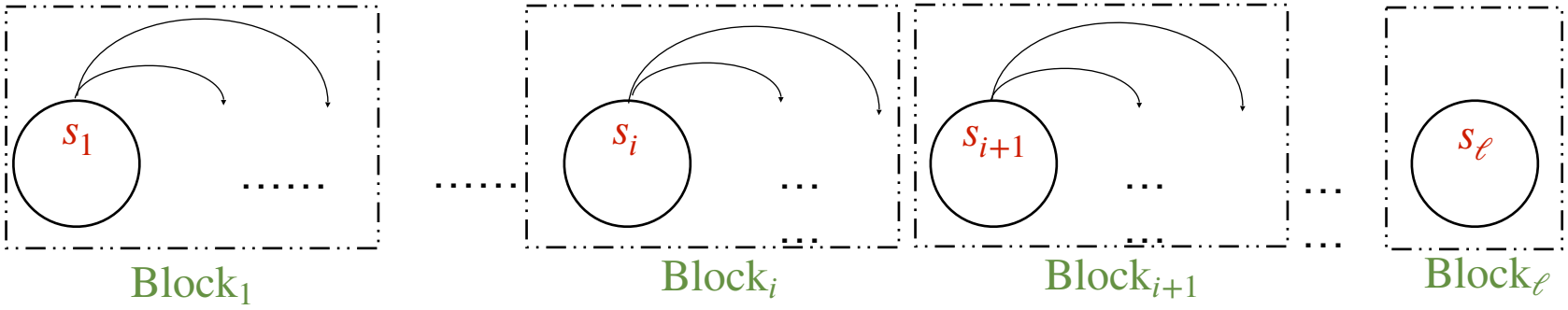
- Call Block_i good if: all the keys of users in Block_i are (almost) independent of PP_{s_ℓ}.
- Claim: If $\ell \gg |\text{PP}_{s_\ell}|$, then a good block exists (**info-theoretical argument**).
- Conclusion: re-sampled fresh fake keys for identities in Block_i look legitimate in relation with PP_{s_ℓ}.

Step 3: Using skipping sequence to attack



- Call Block_i good if: all the keys of users in Block_i are (almost) independent of PP_{s_ℓ}.
- Claim: If $\ell \gg |\text{PP}_{s_\ell}|$, then a good block exists (**info-theoretical argument**).
- Conclusion: re-sampled fresh fake keys for identities in Block_i look legitimate in relation with PP_{s_ℓ}.
- The attack on identity s_i, assuming Block_i is good:

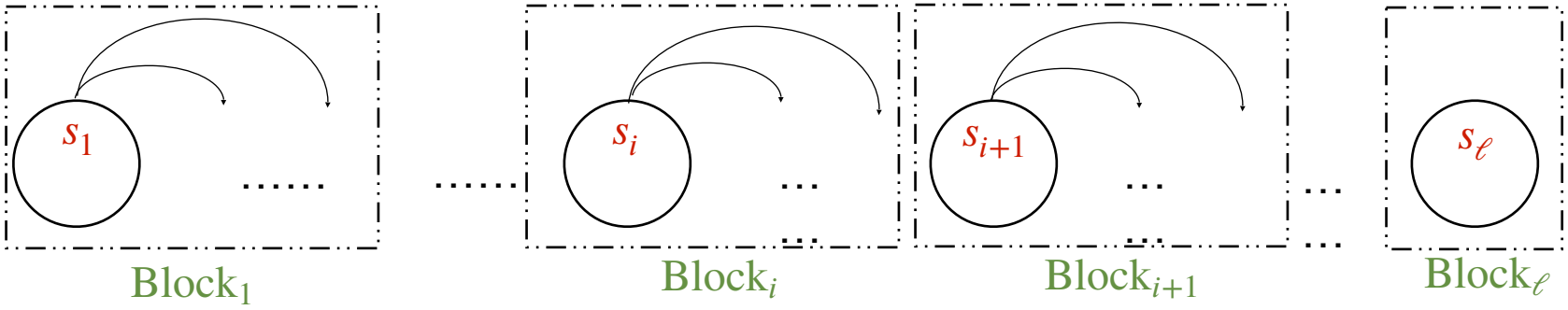
Step 3: Using skipping sequence to attack



- Call Block_i good if: all the keys of users in Block_i are (almost) independent of PP_{s_ℓ}.
- Claim: If $\ell \gg |\text{PP}_{s_\ell}|$, then a good block exists (**info-theoretical argument**).
- Conclusion: re-sampled fresh fake keys for identities in Block_i look legitimate in relation with PP_{s_ℓ}.
- The attack on identity s_i, assuming Block_i is good:

(1) re-sample fake keys for all identities in Block_i

Step 3: Using skipping sequence to attack

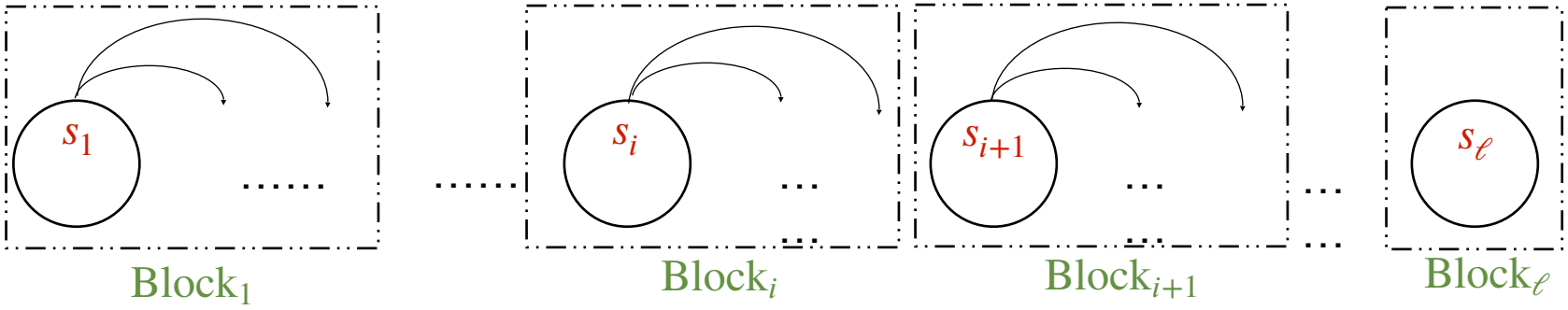


- Call Block_i good if: all the keys of users in Block_i are (almost) independent of PP_{s_ℓ}.
- Claim: If $\ell \gg |\text{PP}_{s_\ell}|$, then a good block exists (**info-theoretical argument**).
- Conclusion: re-sampled fresh fake keys for identities in Block_i look legitimate in relation with PP_{s_ℓ}.
- The attack on identity s_i, assuming Block_i is good:

  (1) re-sample fake keys for all identities in Block_i

  (2) generate all the decryption updates for s_i

Step 3: Using skipping sequence to attack



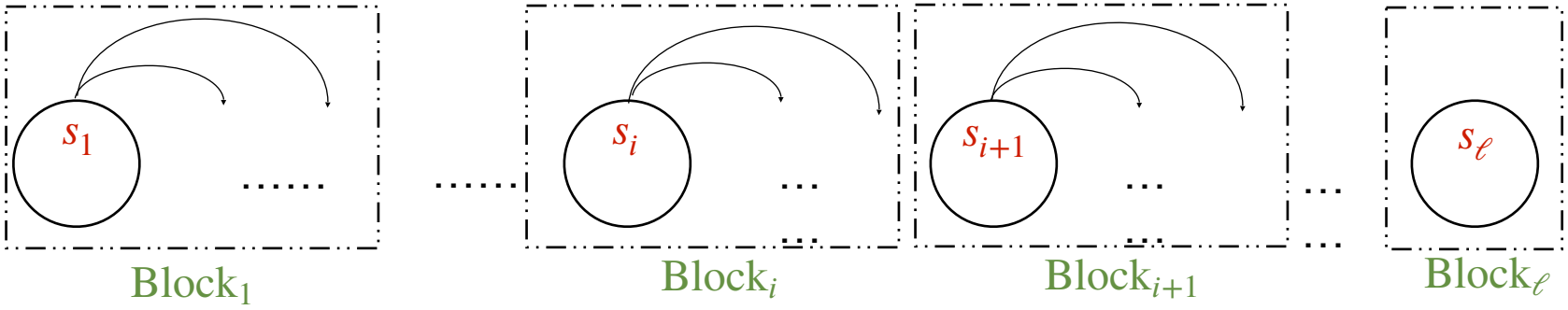
- Call Block_i good if: all the keys of users in Block_i are (almost) independent of PP_{s_ℓ}.
- Claim: If $\ell \gg |\text{PP}_{s_\ell}|$, then a good block exists (**info-theoretical argument**).
- Conclusion: re-sampled fresh fake keys for identities in Block_i look legitimate in relation with PP_{s_ℓ}.
- The attack on identity s_i, assuming Block_i is good:

  (1) re-sample fake keys for all identities in Block_i

  (2) generate all the decryption updates for s_i

  (3) decrypt challenge encrypted for s_i (under PP_{s_ℓ})

Step 3: Using skipping sequence to attack



- Call Block_i good if: all the keys of users in Block_i are (almost) independent of PP_{s_ℓ}.
- Claim: If $\ell \gg |\text{PP}_{s_\ell}|$, then a good block exists (**info-theoretical argument**).
- Conclusion: re-sampled fresh fake keys for identities in Block_i look legitimate in relation with PP_{s_ℓ}.
- The attack on identity s_i, assuming Block_i is good:

  (1) re-sample fake keys for all identities in Block_i

  (2) generate all the decryption updates for s_i

  (3) decrypt challenge encrypted for s_i (under PP_{s_ℓ})

Conclusion

- $\tilde{\Omega}(\log n)$ updates are necessary in RBEs with fixed update times.

Conclusion

- $\tilde{\Omega}(\log n)$ updates are necessary in RBEs with fixed update times.

- Key idea: characterizing *skipping sequences* in DAGs.

Conclusion

- $\tilde{\Omega}(\log n)$ updates are necessary in RBEs with fixed update times.

- Key idea: characterizing *skipping sequences* in DAGs.

- Main question: break this barrier using dynamic update times (that depend on the registered public keys) ?

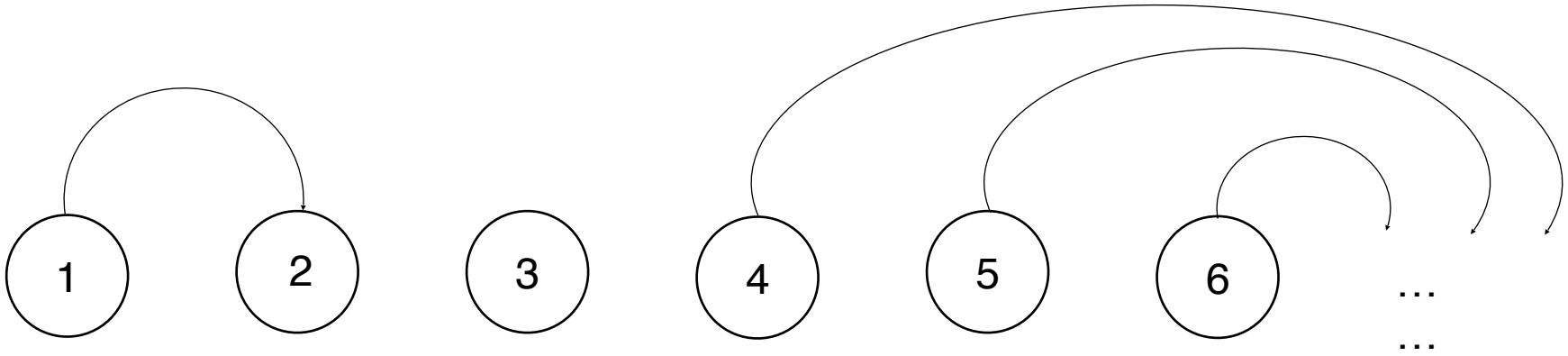
Thanks!

Proof idea of combinatorial result

Proof idea of combinatorial result

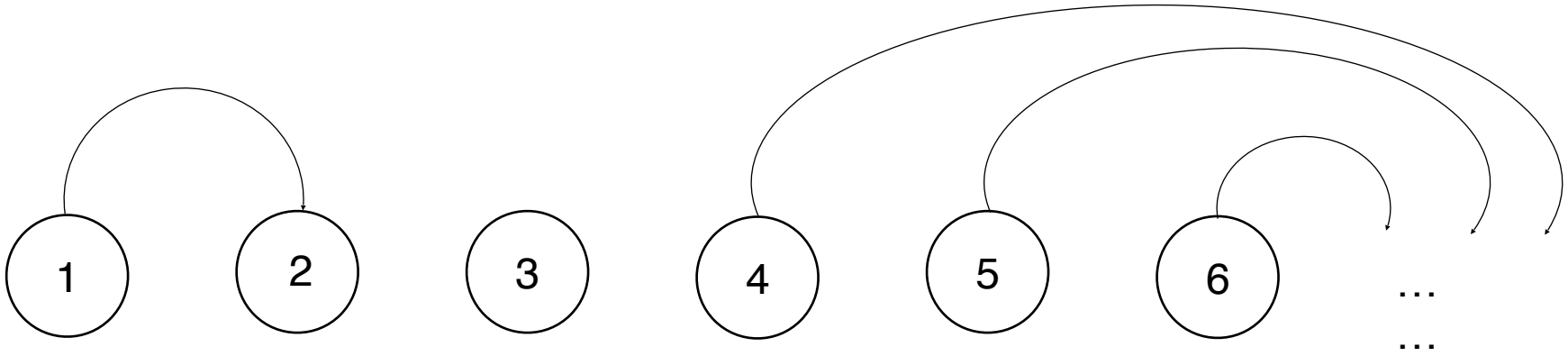
- Focus on degree one.

Proof idea of combinatorial result



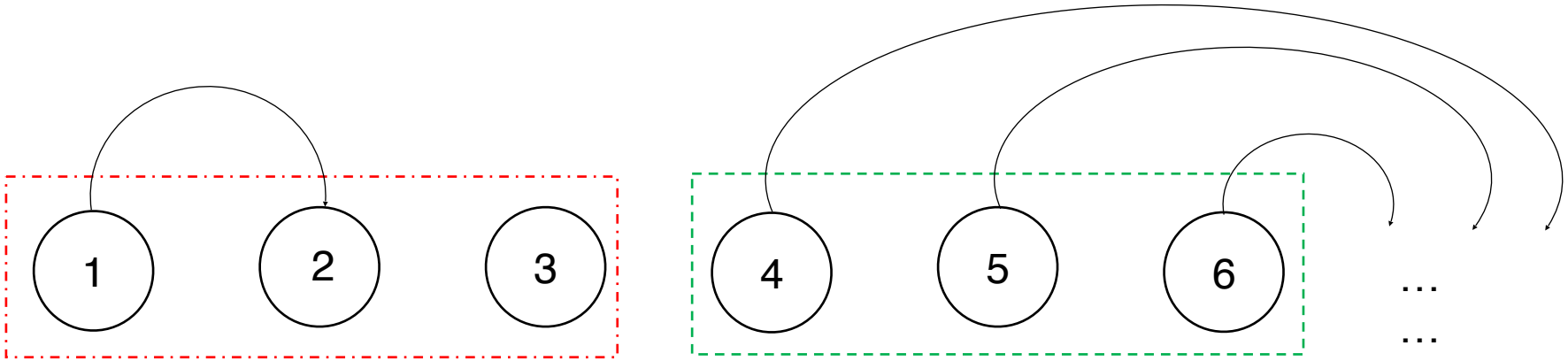
- Focus on degree one.

Proof idea of combinatorial result



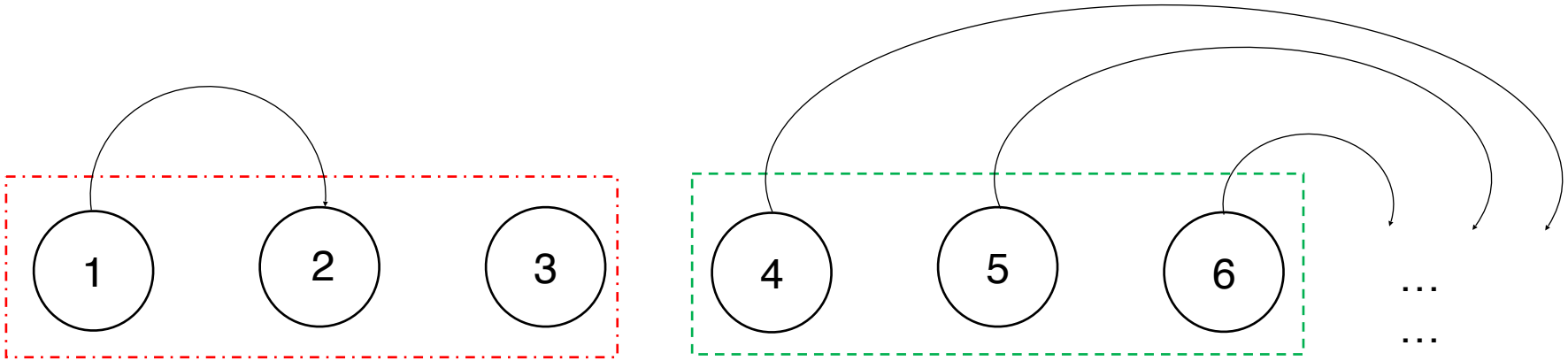
- Focus on degree one.
- Divide DAG into consecutive blocks.

Proof idea of combinatorial result



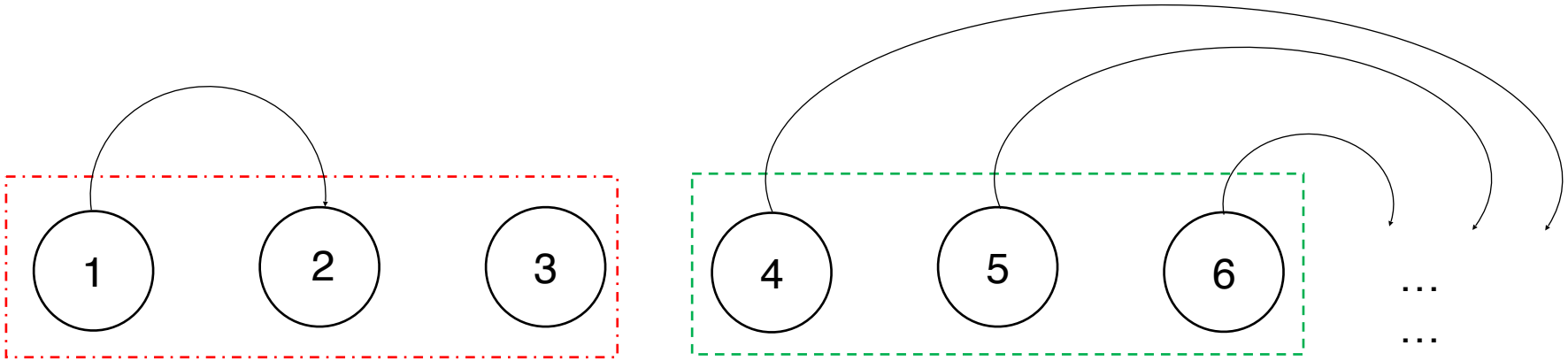
- Focus on degree one.
- Divide DAG into consecutive blocks.

Proof idea of combinatorial result



- Focus on degree one.
- Divide DAG into consecutive blocks.
- Red block: the edges of one of the vertices lie in the block

Proof idea of combinatorial result



- Focus on degree one.
- Divide DAG into consecutive blocks.
- Red block: the edges of one of the vertices lie in the block
- Green block: all vertices have at least one edge going outside the block