Cuckoo Commitments: Registration-Based Encryption and Key-Value Map Commitments for Large Spaces

Dario Fiore

IMDEA Software Institute

Dimitris Kolonelos

IMDEA Software Institute
& Universidad Politecnica de Madrid

Paola de Perthuis

DIENS, CNRS, Inria, université PSL
& Cosmian

ePrint: https://eprint.iacr.org/2023/1389.pdf

Asiacrypt 2023, Guangzhou
5 December 2023

Outline

Motivation: Registration-Based Encryption

Our Contributions

Our RBE compiler

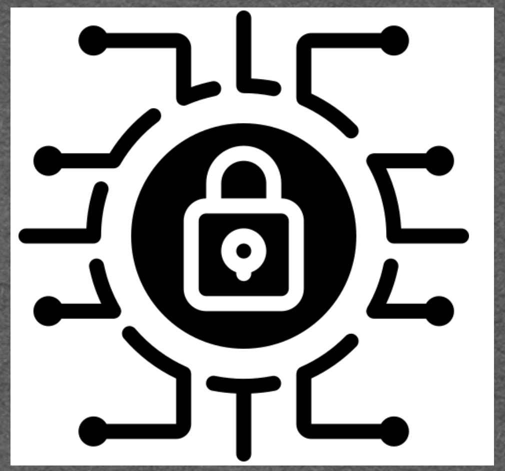
Our KVC compiler

Conclusions and Open Problems

Motivation: Registration-Based Encryption

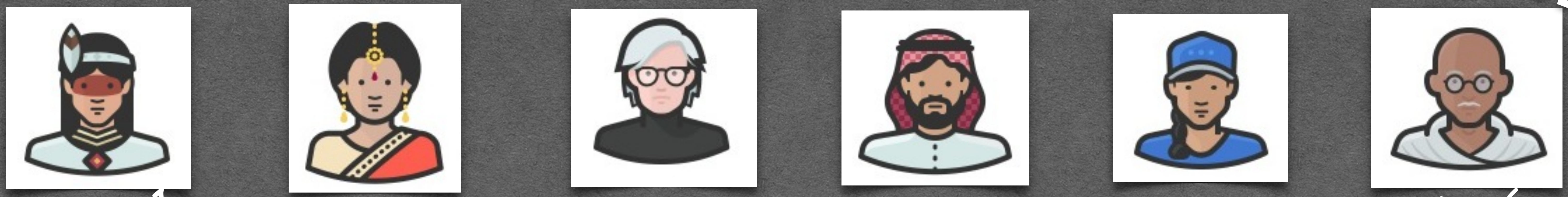
Identity-Based Encryption [Sha84], [BF01]

Private Key Generator



Encryptor

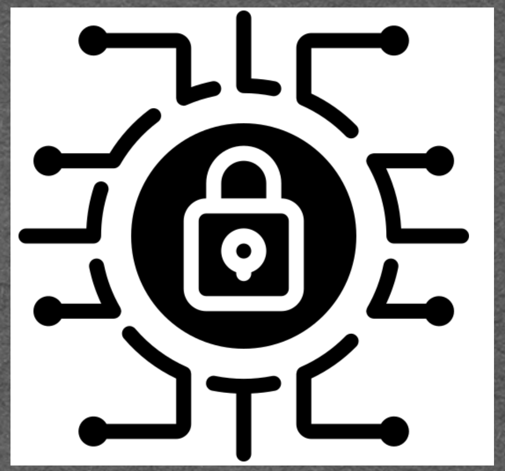
Decryptors



user 1 user 2 user 3 user 4 user 5 user 6

Identity-Based Encryption [Sha84], [BF01]

Private Key Generator

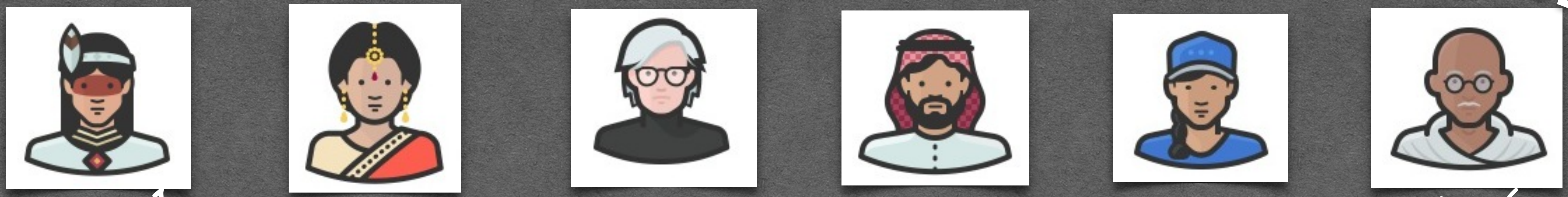


1 Setup$(1^\lambda, n) \rightarrow$ (msk, mpk)

→ mpk

Encryptor

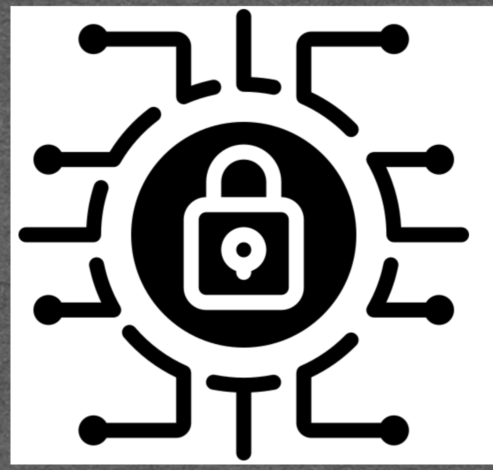
Decryptors



user 1 user 2 user 3 user 4 user 5 user 6

Identity-Based Encryption [Sha84], [BF01]

Private Key Generator



1 $\mathsf{Setup}(1^\lambda, n) \to (\mathsf{msk}, \mathsf{mpk})$

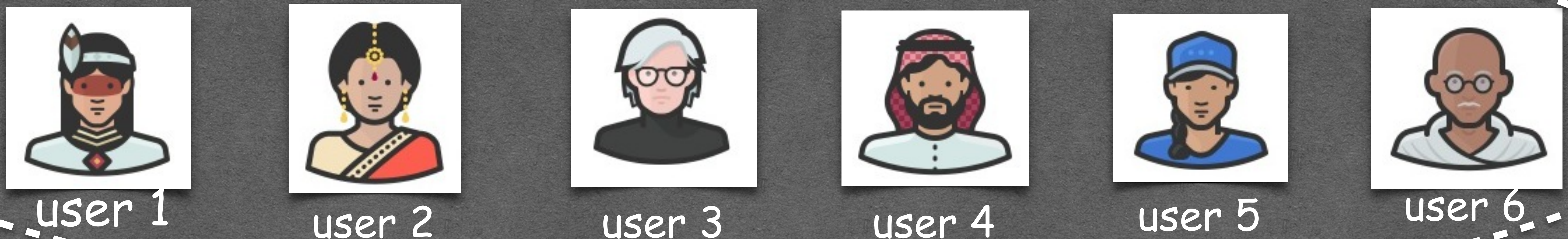
mpk

2 $\mathsf{KeyGen}(\mathsf{msk}, i) \to \mathsf{sk}_i$

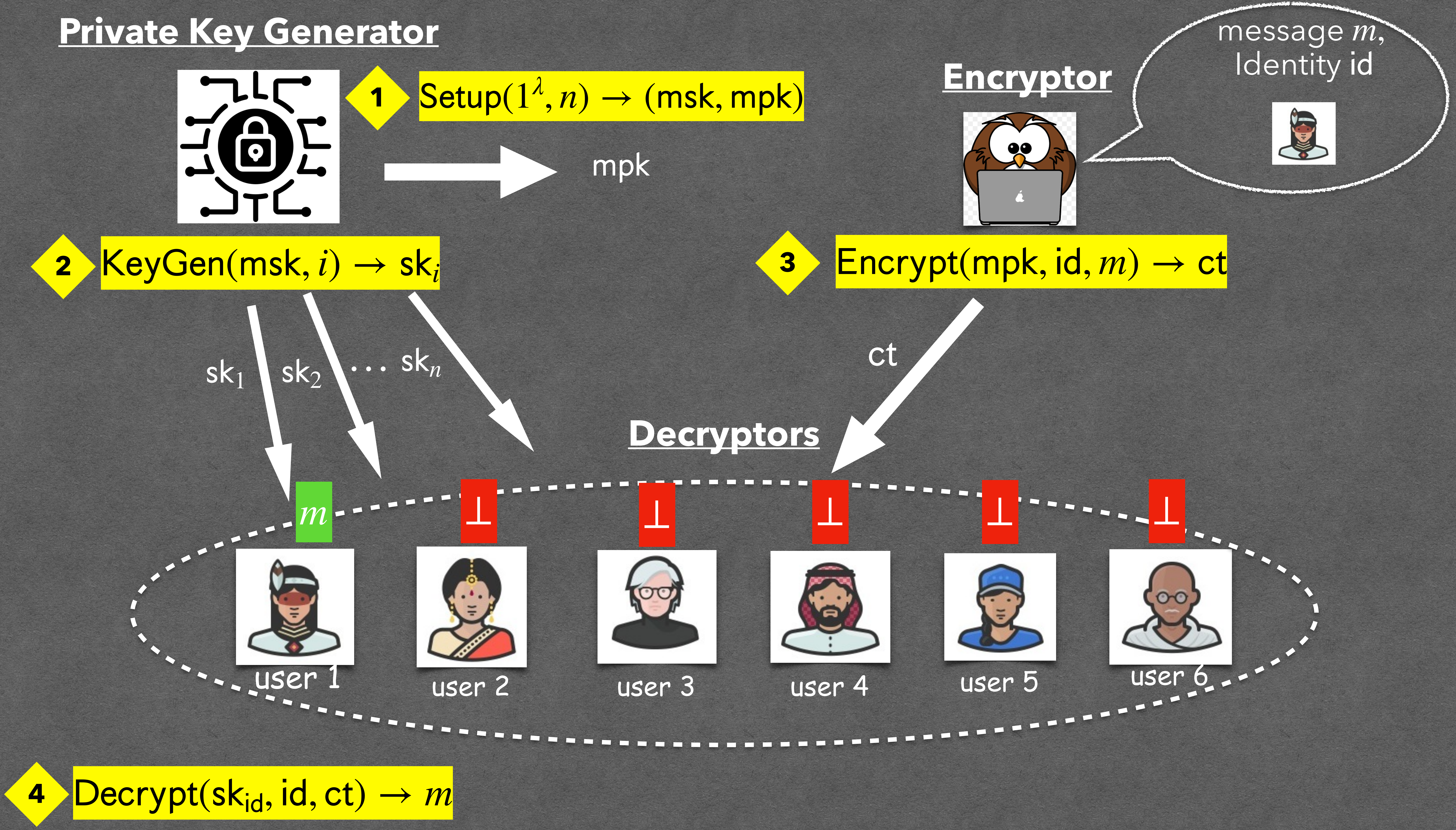
sk_1 sk_2 $\ldots$ sk_n

Encryptor

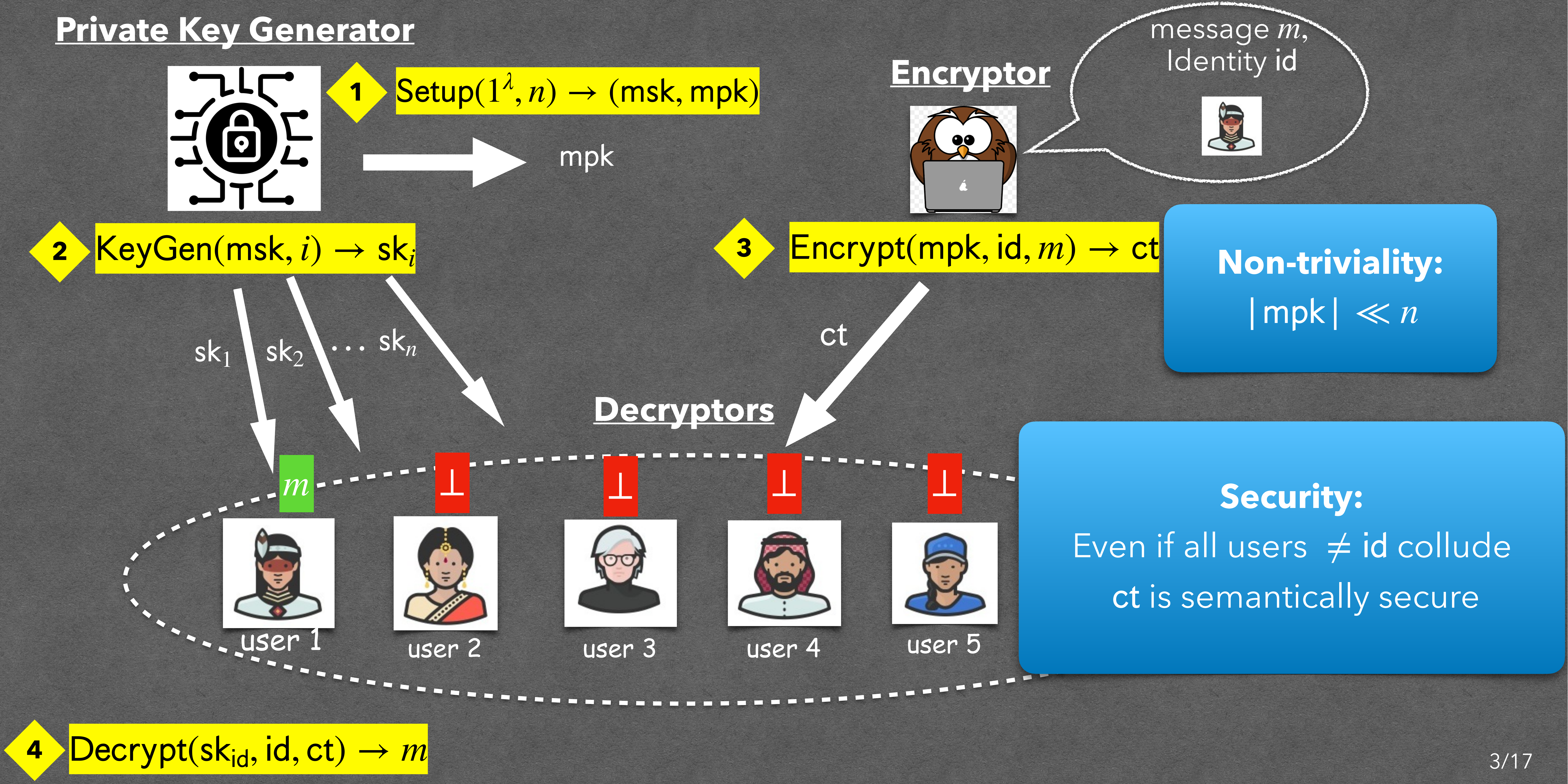
Decryptors



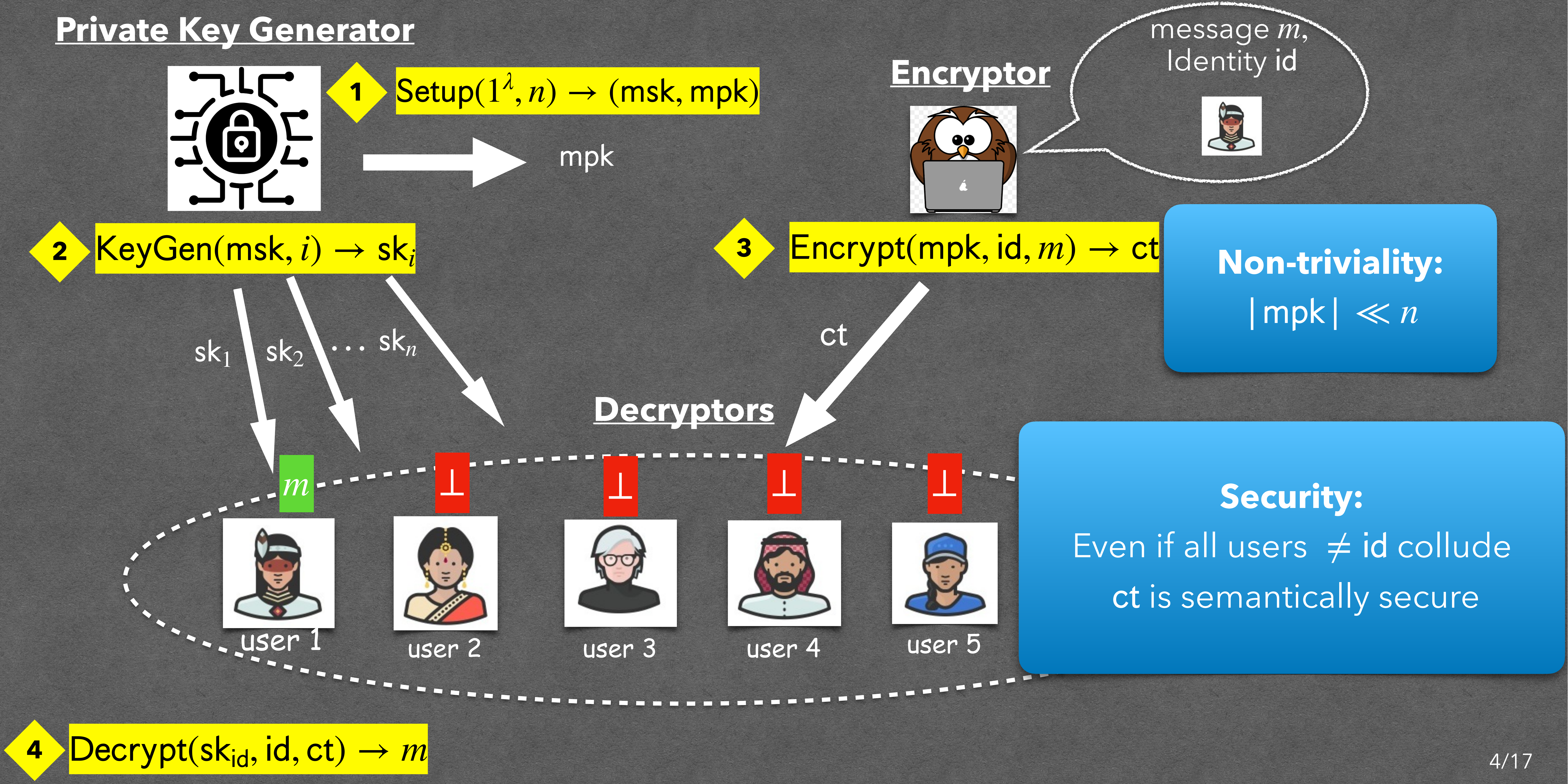
Identity-Based Encryption [Sha84], [BF01]



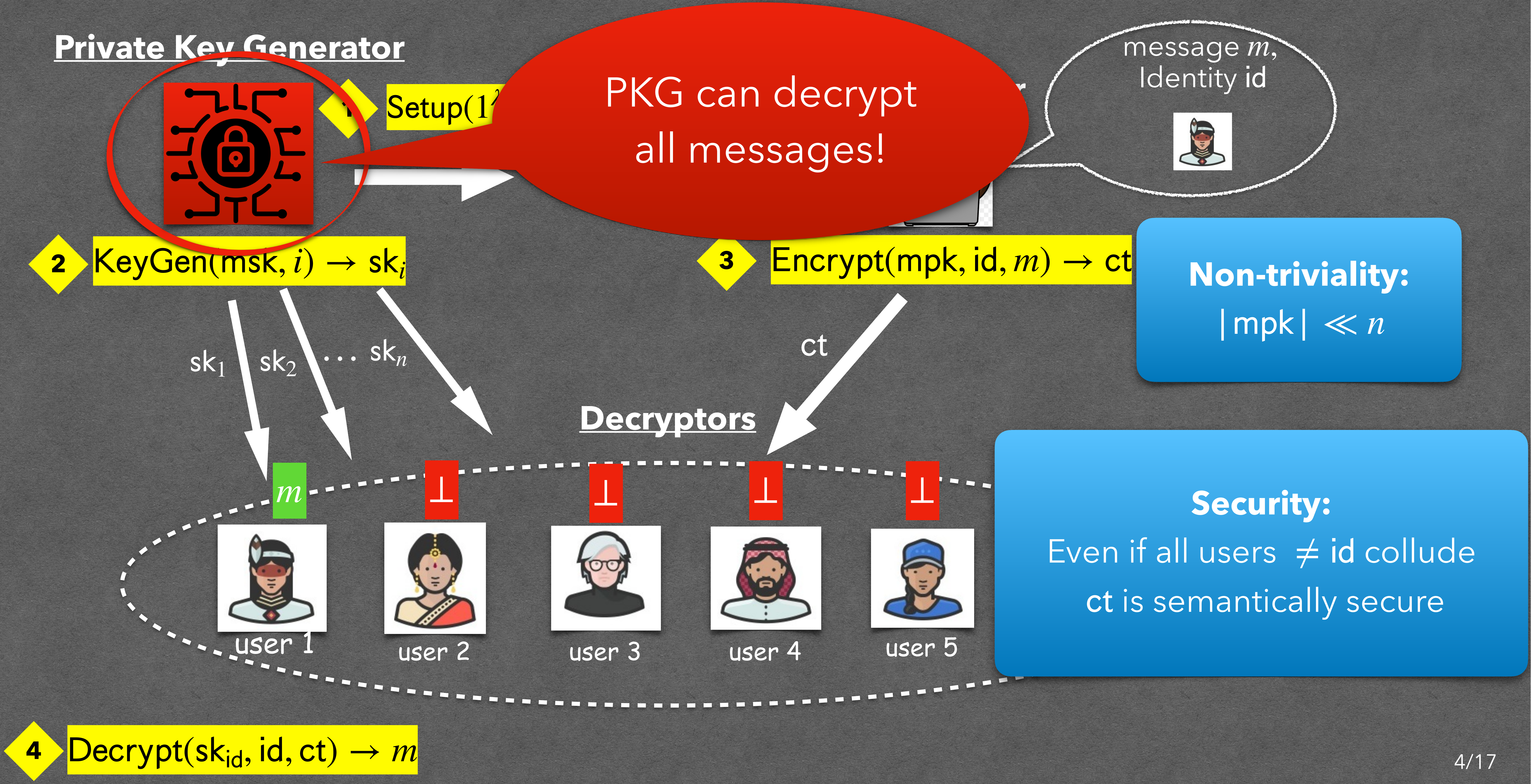
Identity-Based Encryption [Sha84], [BF01]



The key-escrow problem

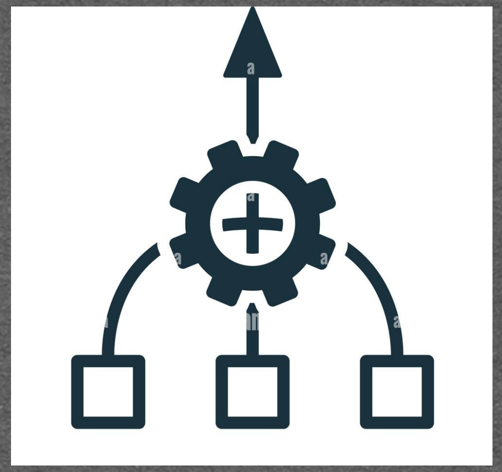


The key-escrow problem



Registration-Based Encryption [GHMR18]

Key Curator



Encryptor

message m,
Identity id

4 $\text{Encrypt}(\text{pp}, \text{id}, m) \to \text{ct}$

Non-triviality:
$|\text{pp}| \ll n$

ct

Decryptors

m $\perp$ $\perp$ $\perp$ $\perp$

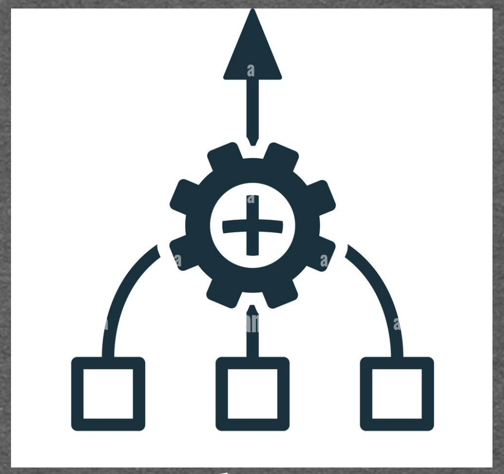
user 1 user 2 user 3 user 4 user 5

Security:
Even if all users $\neq$ id collude
ct is semantically secure

5 $\text{Decrypt}(\text{sk}_{\text{id}}, \text{id}, \text{ct}) \to m$

Registration-Based Encryption [GHMR18]

Key Curator



pk_1 u_1 … pk_n u_n

2 KeyGen(crs, id) $\rightarrow$ (sk_{id}, pk_{id})

Encryptor

message m,
Identity id

4 Encrypt(pp, id, m) $\rightarrow$ ct

ct

Non-triviality:
$|pp| \ll n$

Decryptors

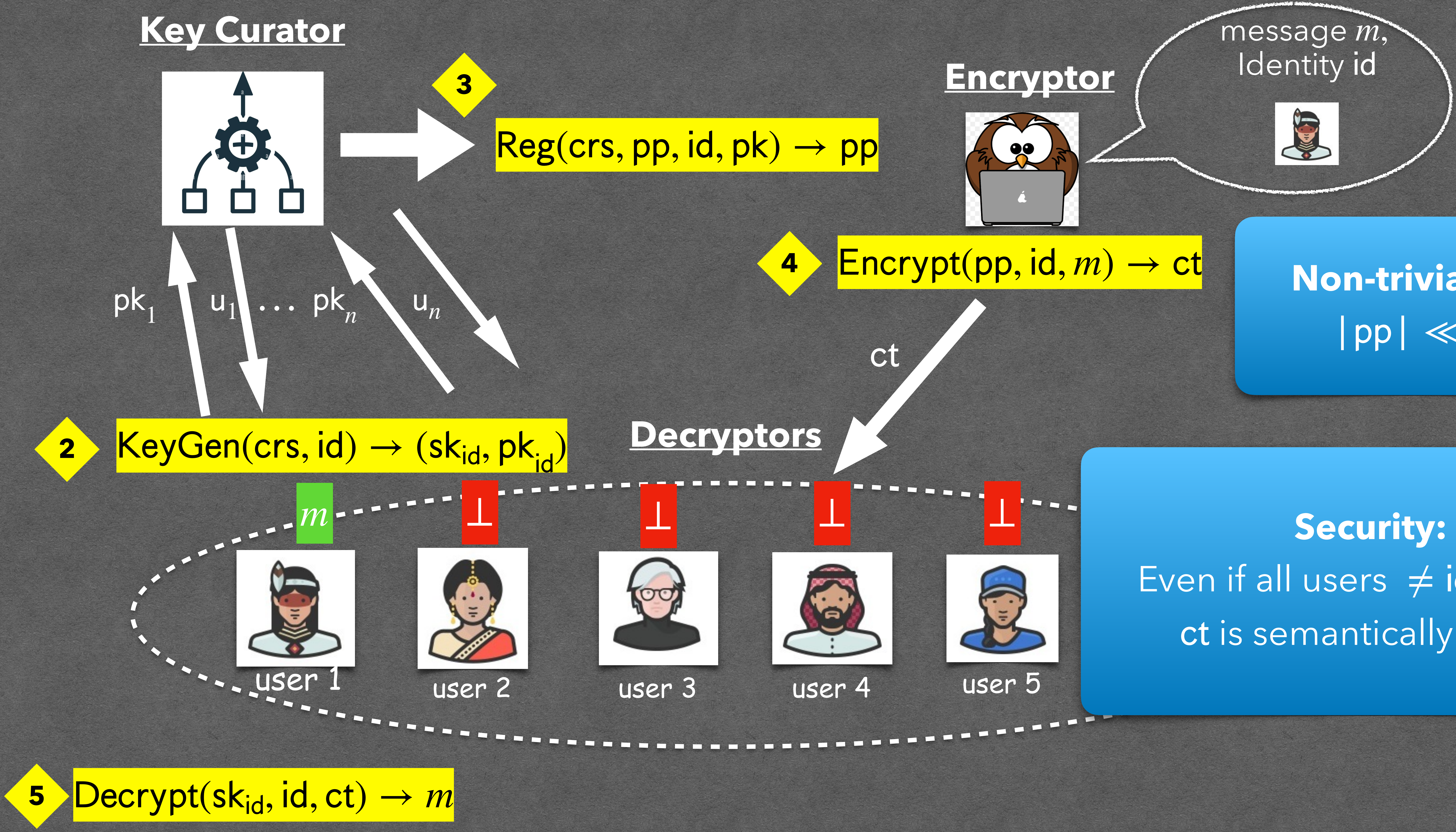
m $\perp$ $\perp$ $\perp$ $\perp$

user 1 user 2 user 3 user 4 user 5

Security:
Even if all users $\neq$ id collude
ct is semantically secure

5 Decrypt(sk_{id}, id, ct) $\rightarrow$ m

Registration-Based Encryption [GHMR18]



Non-triviality:

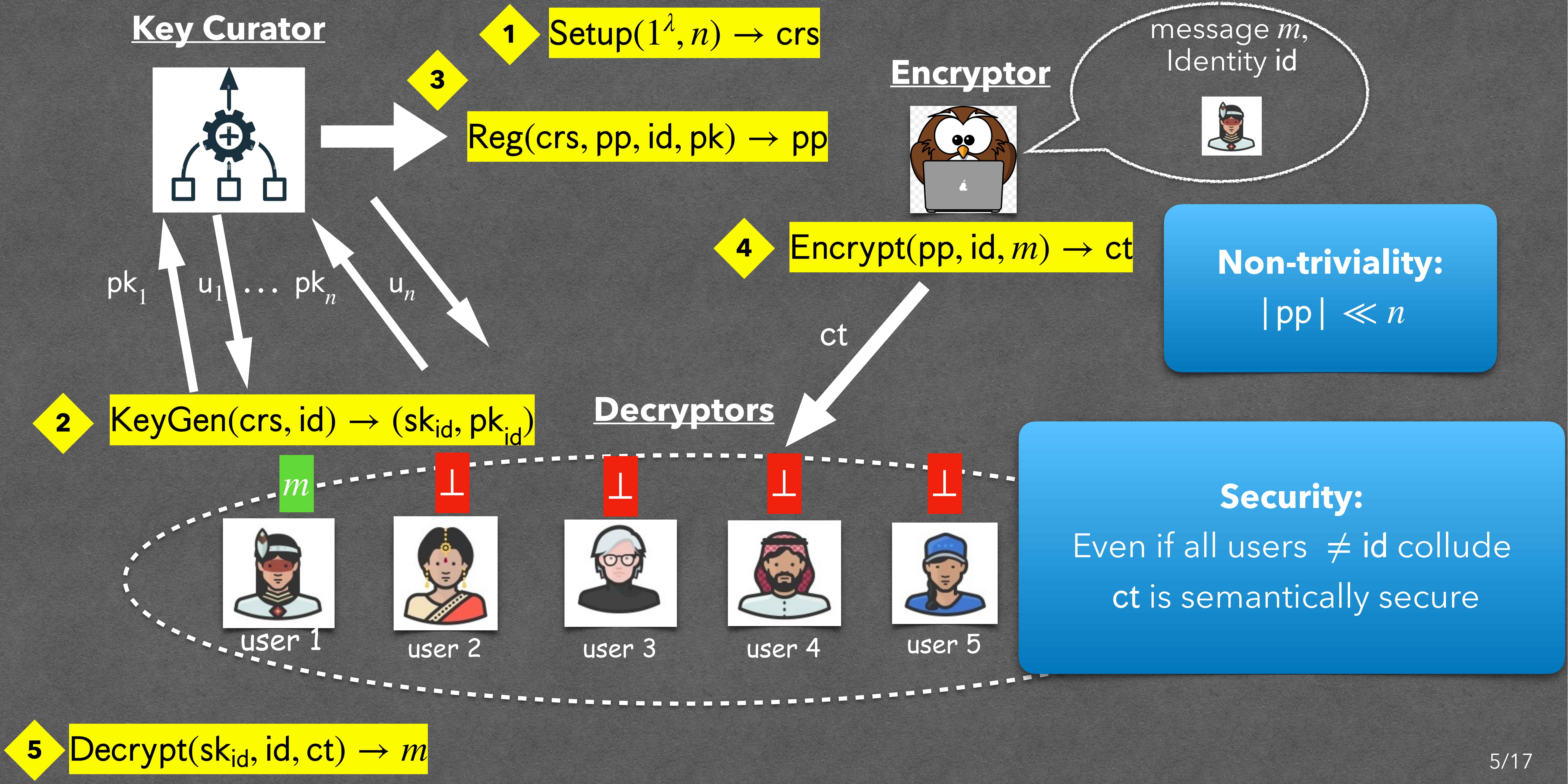
$|\mathsf{pp}| \ll n$

Security:

Even if all users $\neq$ id collude

ct is semantically secure

Registration-Based Encryption [GHMR18]



Registration-Based Encryption: Prior constructions

- ❖ Early (non-black box) constructions [GHMR18], [GHMRS19], [GV20], [CES21]

indistinguishability Obfuscation or Hash-Garbling [CDGGMP17] [DG17]

- ❖ Recent efficient constructions:

	Setting	Efficiency	ID-space
[D**K**LLMR23]	Lattices	×	Unbounded
[HLWW23]	Pairings (composite)	✓	Unbounded
[G**K**MR23]	Pairings (prime)	✓✓	poly(λ)

Registration-Based Encryption: Prior constructions

- Early (non-black box) constructions [GHMR18], [GHMRS19], [GV20], [CES21]

indistinguishability Obfuscation or Hash-Garbling [CDGGMP17] [DG17]

- Recent efficient constructions:

	Setting	Efficiency	ID-space
[DKLLMR23]	Lattices	×	Unbounded
[HLWW23]	Pairings (composite)	✓	Unbounded
[GKMR23]	Pairings (prime)	✓✓	poly(λ)

Our objective: Can we get unbounded identities in [GKMR23]?

Our Contributions

Our Contributions

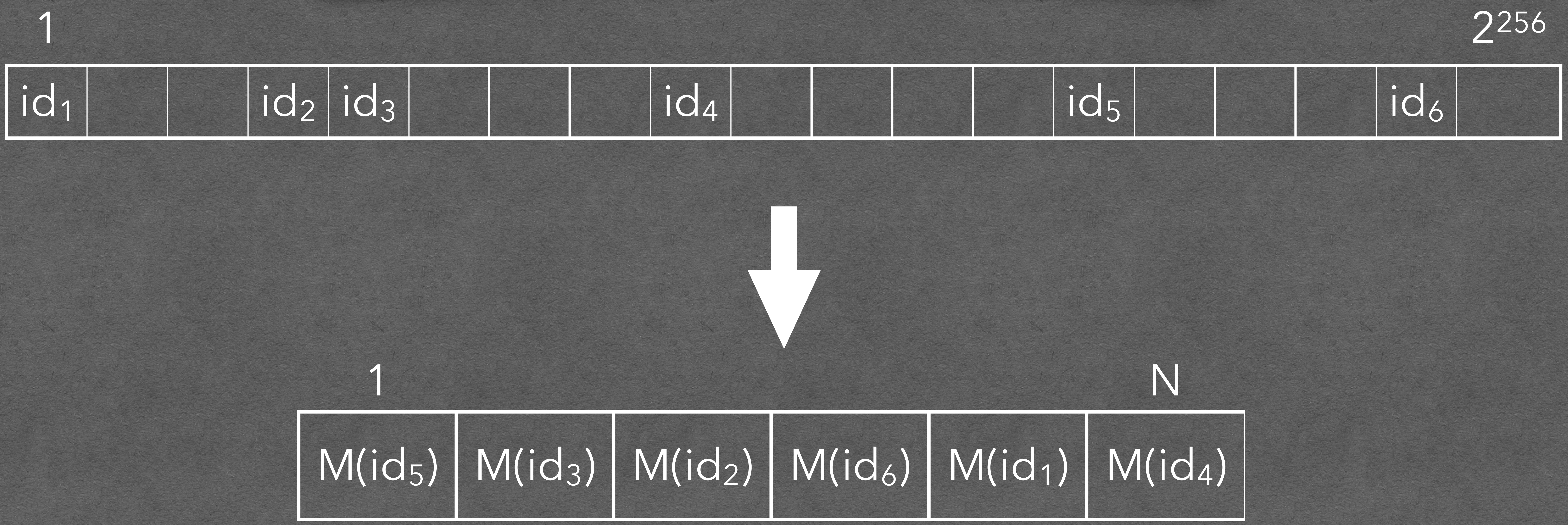
- ★ *Efficient Pairing-based RBE construction with unbounded identities*
- ★ *Generic compiler: Bounded ID-RBE ⇒ Unbounded ID-RBE*
- ★ *Generic compiler: Vector Commitment ⇒ Key-Value Map Commitment*
- ★ *Techniques: Novel use of Cuckoo-Hashing*

Our RBE compiler

Our Core idea

Map the identities to a smaller domain

$$M : \{0,1\}^{2\lambda} \rightarrow [1,n]$$

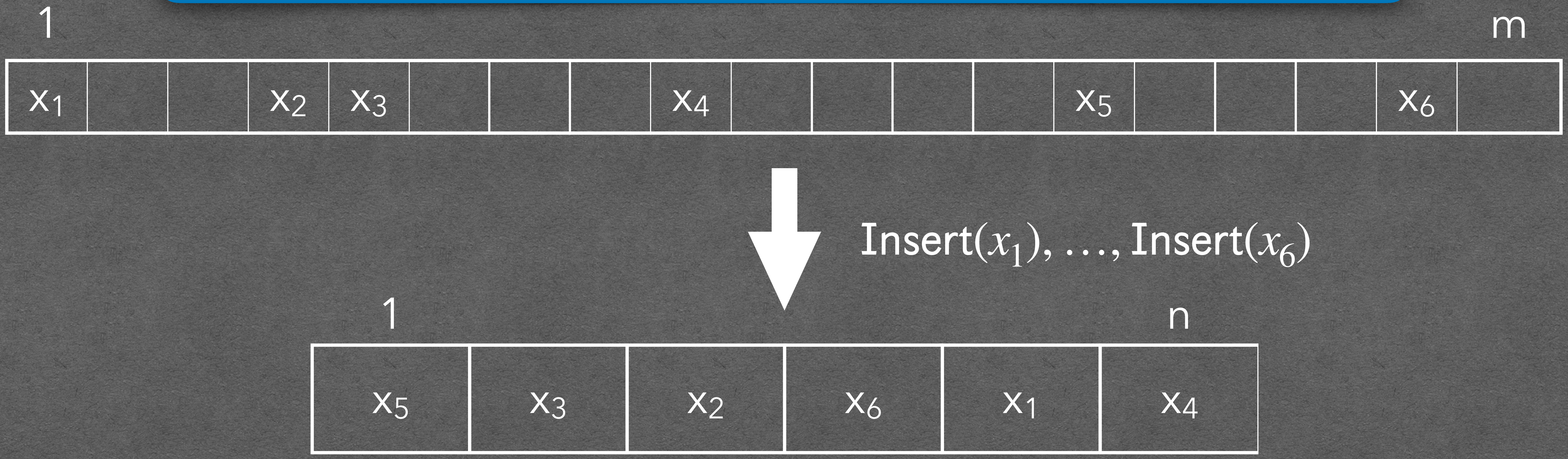


And use an RBE with small identity space

Cuckoo Hashing [PR04]

Cuckoo Hashing

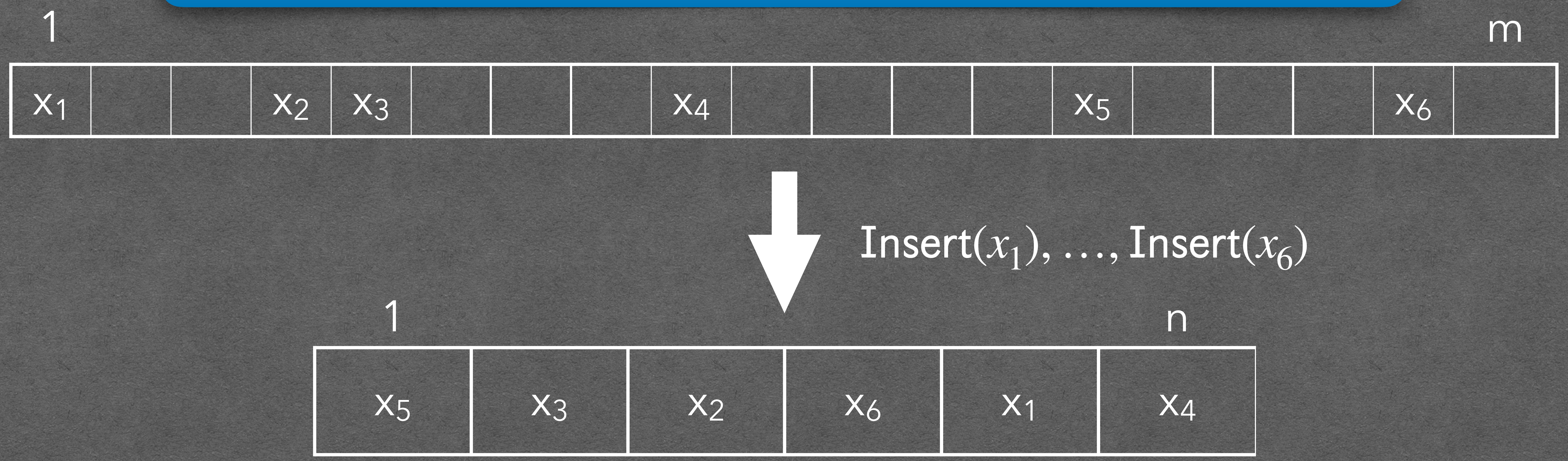
Powerful **probabilistic** data structure to map elements from $[m]$ to elements in $[n]$, where $m > n$ (overcoming collisions)



Cuckoo Hashing [PR04]

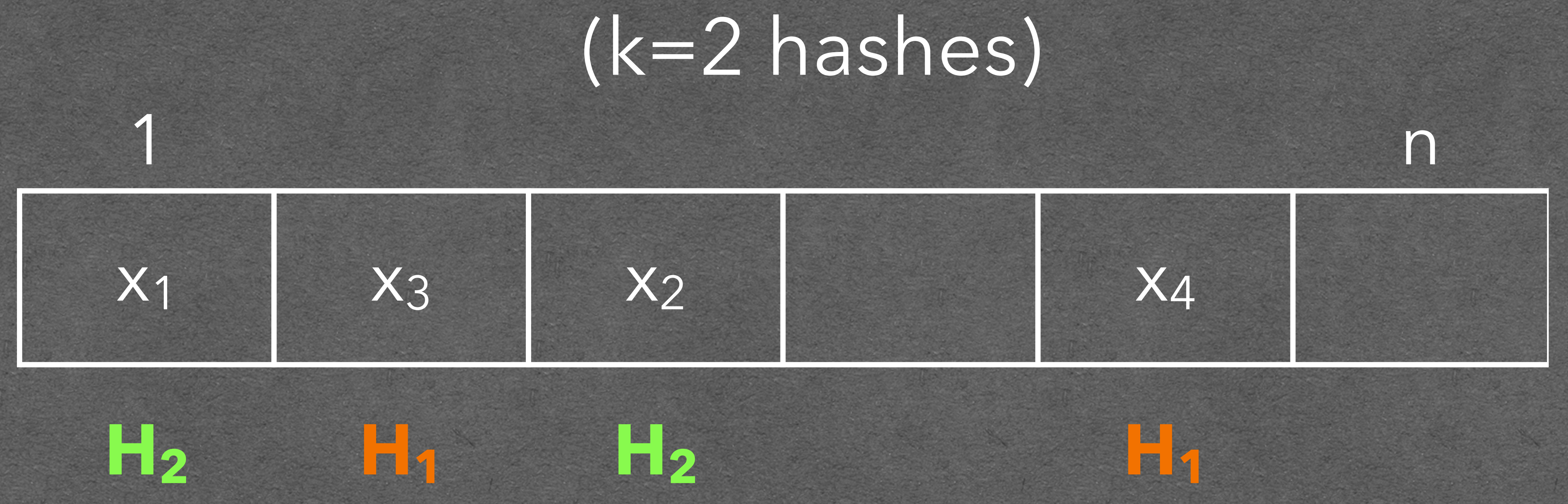
Cuckoo Hashing

Powerful **probabilistic** data structure to map elements from $[m]$ to elements in $[n]$, where $m > n$ (overcoming collisions)



- Pr[*Collision*] $= 0$
- Pr[*Insert*$(x) = \bot$] $= \alpha > 0$
- *Uses k hash functions and evictions.*

Cuckoo Hashing: Example [PR04]



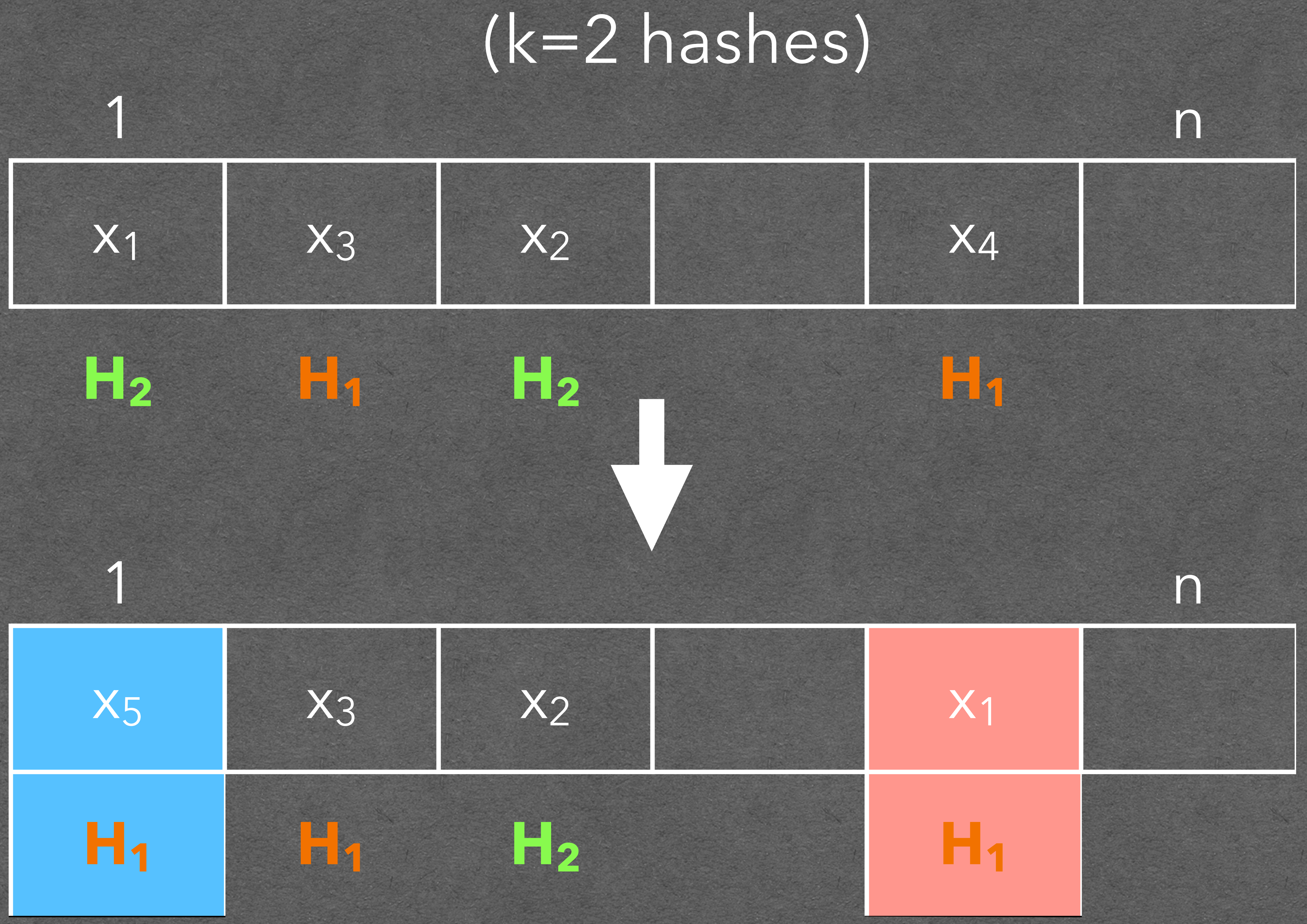
Insert(x_5):

- Puts x_5 in $H_1(x_5)$
- If $H_1(x_5)$ is occupied 'evicts' the element
- Continues until there no eviction

Cuckoo Hashing: Example [PR04]

Insert(x_5):

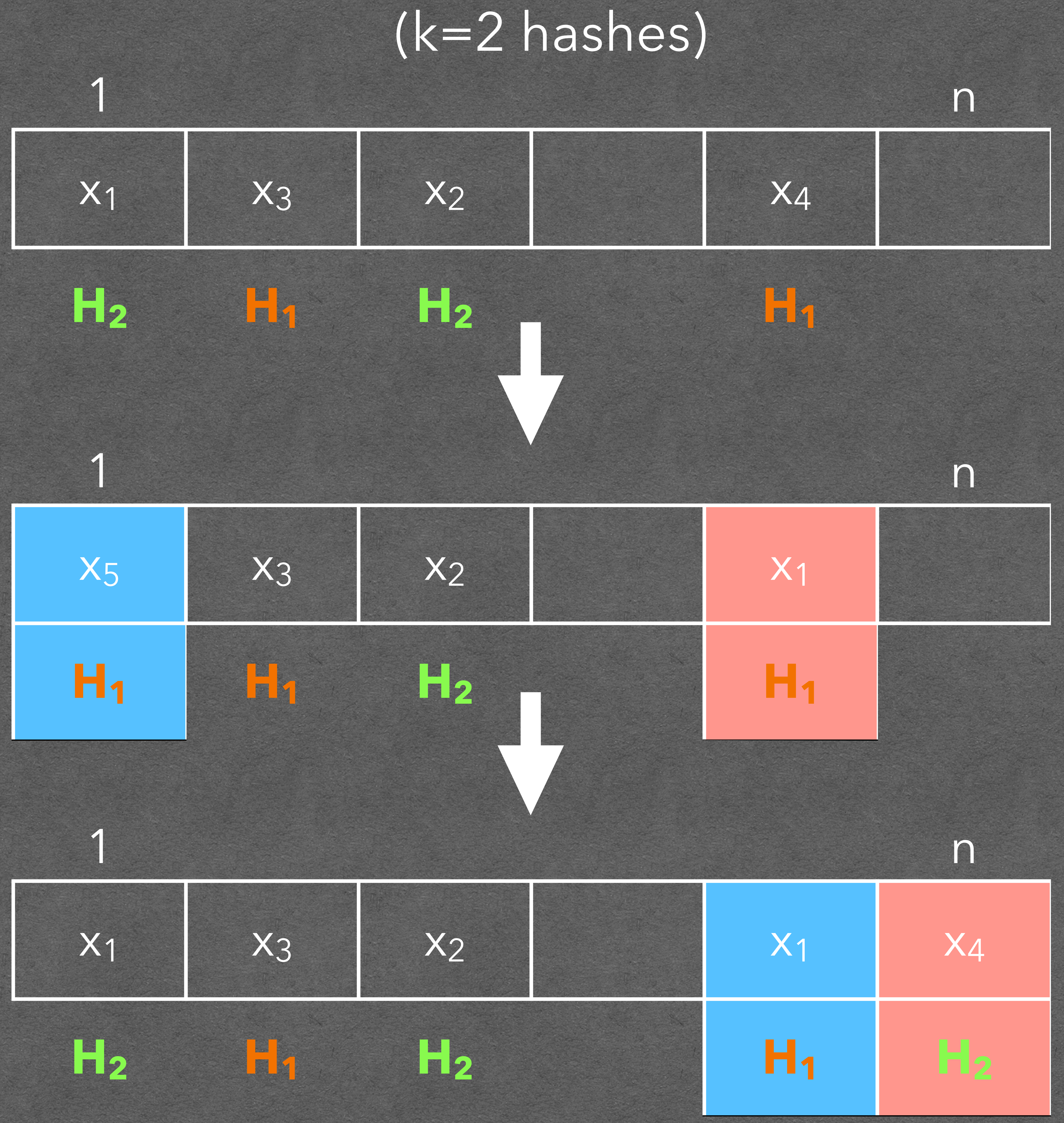
- Puts x_5 in $H_1(x_5)$
- If $H_1(x_5)$ is occupied 'evicts' the element
- Continues until there no eviction



Cuckoo Hashing: Example [PR04]

Insert(x_5):

- Puts x_5 in $H_1(x_5)$
- If $H_1(x_5)$ is occupied 'evicts' the element
- Continues until there no eviction

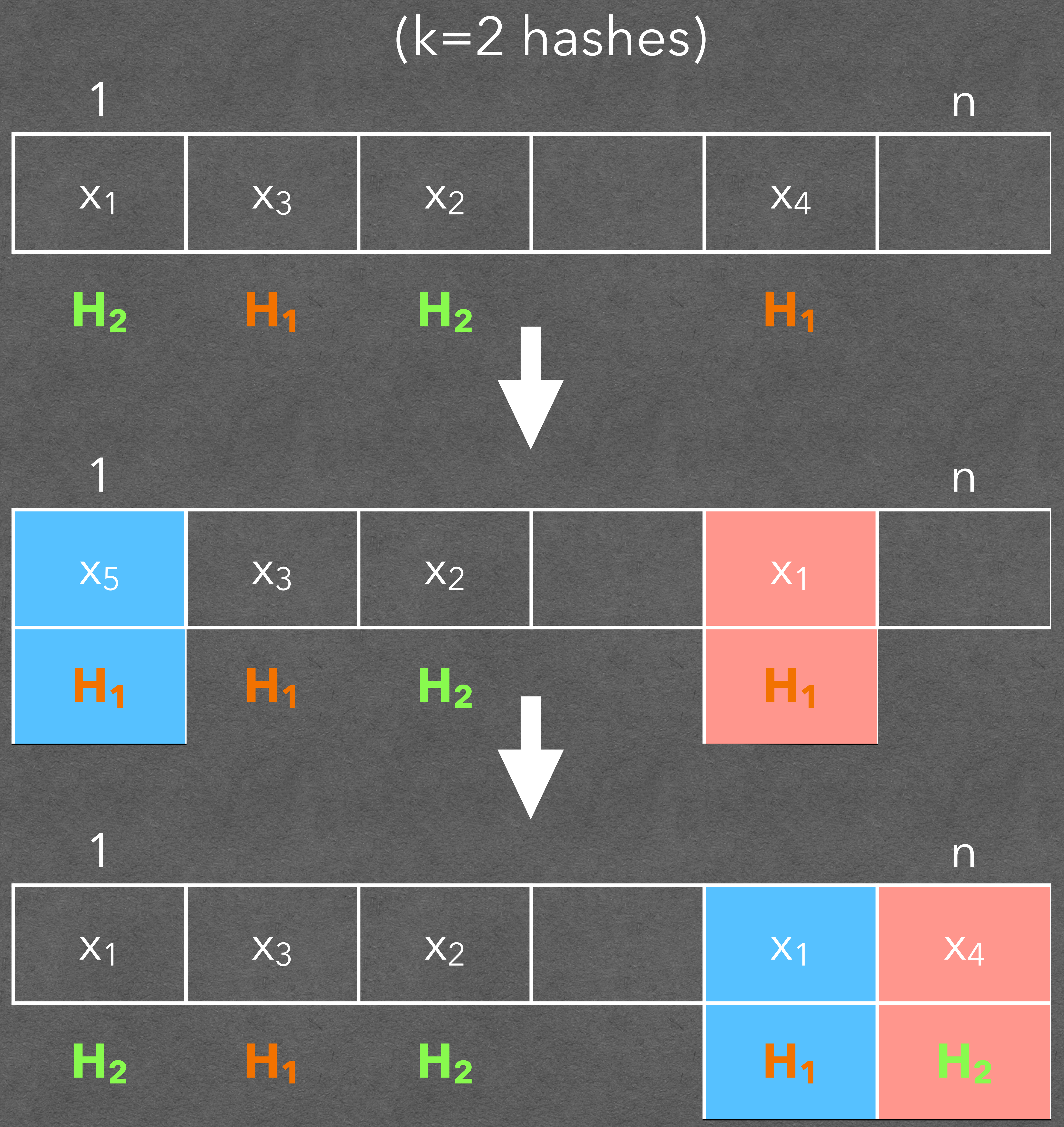


Cuckoo Hashing: Example [PR04]

Insert(x_5):

- Puts x_5 in $H_1(x_5)$
- If $H_1(x_5)$ is occupied 'evicts' the element
- Continues until there no eviction

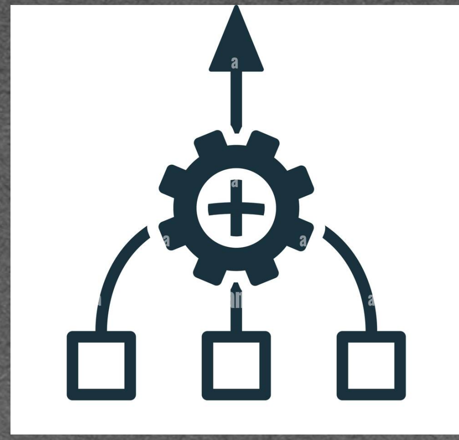
x is either in $H_1(x)$ or $H_2(x)$



Cuckoo Hashing in RBE (with small ID-space)

1 … 2^{256}

id_1			id_2	id_3				id_4					id_5				id_6	

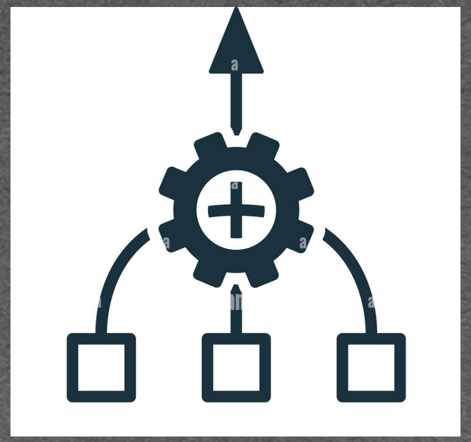


Cuckoo Hashing in RBE (with small ID-space)

1 … 2^{256}

id_1			id_2	id_3				id_4					id_5				id_6	

Cuckoo Hashing



1 … n

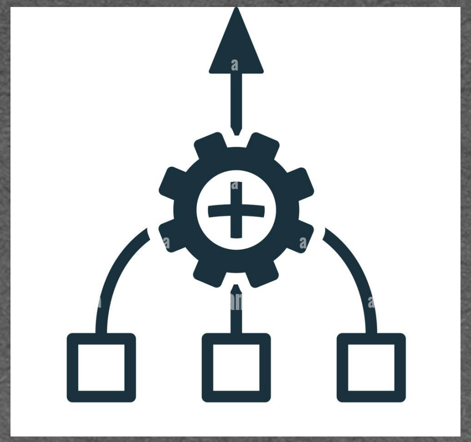
$CH(id_5)$	$CH(id_3)$	$CH(id_2)$	$CH(id_6)$	$CH(id_1)$	$CH(id_4)$

Cuckoo Hashing in RBE (with small ID-space)

1 $\qquad$ 2^{256}

id_1			id_2	id_3				id_4					id_5				id_6	

Cuckoo Hashing



1 $\qquad$ n

CH(id_5)	CH(id_3)	CH(id_2)	CH(id_6)	CH(id_1)	CH(id_4)

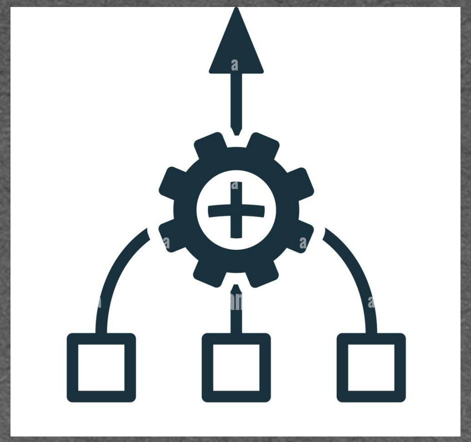
$\widetilde{\text{RBE}}.\text{Reg}(\ldots, \text{CH(id)}, \ldots)$

Cuckoo Hashing in RBE (with small ID-space)

1 … 2^{256}

id_1			id_2	id_3				id_4					id_5			id_6	

Cuckoo Hashing



1 … n

$CH(id_5)$	$CH(id_3)$	$CH(id_2)$	$CH(id_6)$	$CH(id_1)$	$CH(id_4)$

→ $\widetilde{RBE}.Reg(\ldots, CH(id), \ldots)$

$\widetilde{RBE}.Enc(\ldots, H_1(id), \ldots)$

$\widetilde{RBE}.Enc(\ldots, H_2(id), \ldots)$

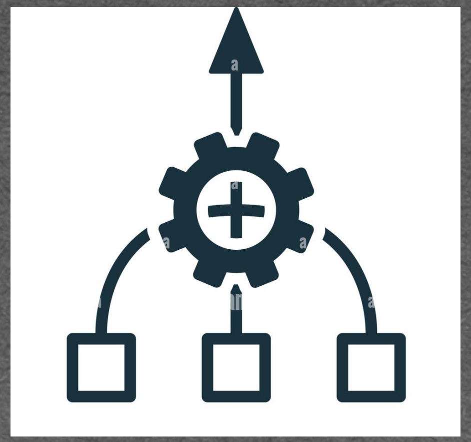
$\vdots$

$\widetilde{RBE}.Enc(\ldots, H_k(id), \ldots)$

Cuckoo Hashing in RBE (with small ID-space)

1																			2^{256}
id_1			id_2	id_3				id_4					id_5				id_6		

Cuckoo Hashing



1					n
$CH(id_5)$	$CH(id_3)$	$CH(id_2)$	$CH(id_6)$	$CH(id_1)$	$CH(id_4)$

→ $\widetilde{\mathsf{RBE}}.\mathsf{Reg}(\ldots, CH(id), \ldots)$

$\widetilde{\mathsf{RBE}}.\mathsf{Enc}(\ldots, H_1(id), \ldots)$

$\widetilde{\mathsf{RBE}}.\mathsf{Enc}(\ldots, H_2(id), \ldots)$

$\vdots$

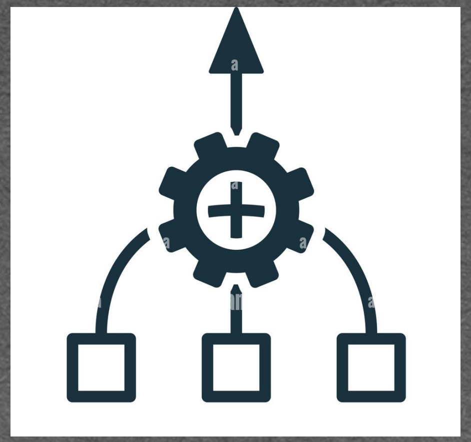
$\widetilde{\mathsf{RBE}}.\mathsf{Enc}(\ldots, H_k(id), \ldots)$

Not secure yet!

Cuckoo Hashing in RBE (with small ID-space)

1																		2^{256}
id_1			id_2	id_3				id_4					id_5				id_6	

↓ Cuckoo Hashing



1					n
$CH(id_5)$	$CH(id_3)$	$CH(id_2)$	$CH(id_6)$	$CH(id_1)$	$CH(id_4)$

→ $\widetilde{\text{RBE}}.\text{Reg}(\ldots, CH(id), \ldots)$

Maybe id′ — $\widetilde{\text{RBE}}.\text{Enc}(\ldots, H_1(id), \ldots)$

Only id — $\widetilde{\text{RBE}}.\text{Enc}(\ldots, H_2(id), \ldots)$

⋮

Maybe id′ — $\widetilde{\text{RBE}}.\text{Enc}(\ldots, H_k(id), \ldots)$

Not secure yet!

The missing piece:
Witness Encryption for Vector Commitments

$\vec{v} =$

v_1	v_2	v_3	v_4	v_5	v_6	...	v_{n-1}	v_n

Vector Commitment

[LY10], [CF13]

Commit

C

Open $\longrightarrow$

v_i ,

Λ_i

Succinctness

 ,  $\ll n$

The missing piece:
Witness Encryption for Vector Commitments

$\vec{v} =$	v_1	v_2	v_3	v_4	v_5	v_6	...	v_{n-1}	v_n

Vector Commitment

[LY10], [CF13]

Commit

C

Open $\longrightarrow$

v_i ,

Λ_i

Succinctness

 ,  $\ll n$

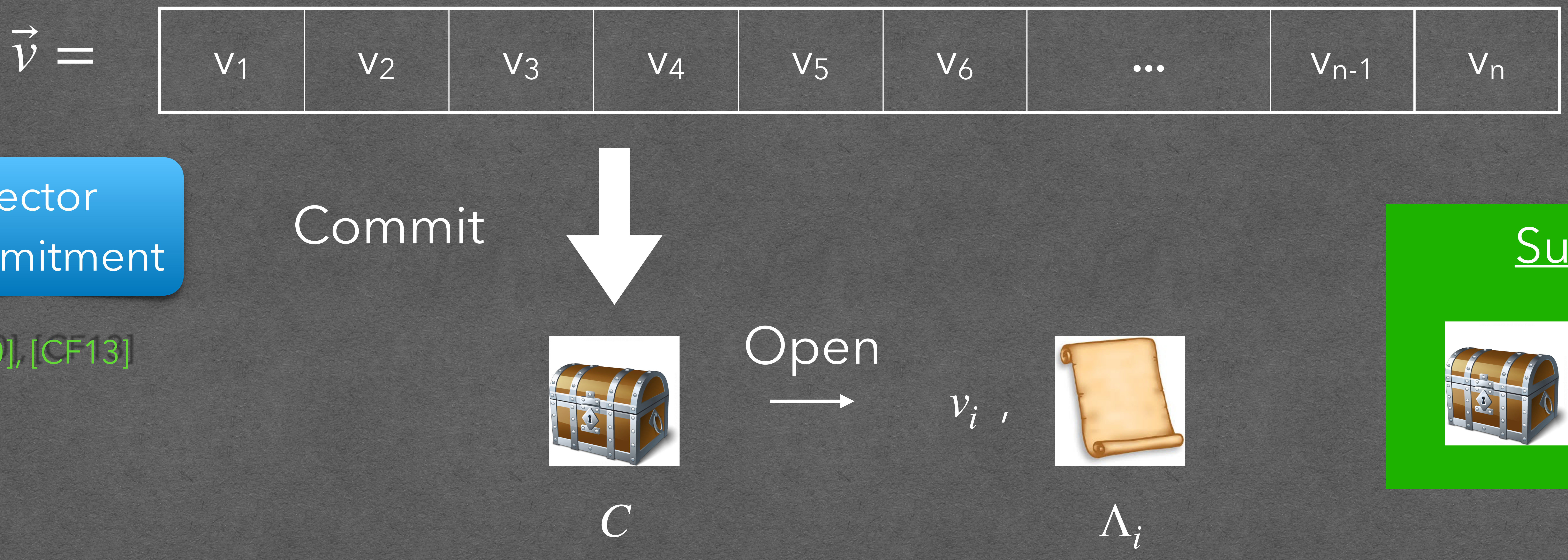
★ *Witness Encryption w.r.t. Vector Commitment (VCWE):*

$\mathsf{Enc}(C, v^*, i, m) \rightarrow \mathsf{ct}$ such that Λ_i decrypts only if $\vec{v}[i] = v^*$

The missing piece:
Witness Encryption for Vector Commitments

Vector Commitment

[LY10], [CF13]



★ *Witness Encryption w.r.t. Vector Commitment (VCWE):*

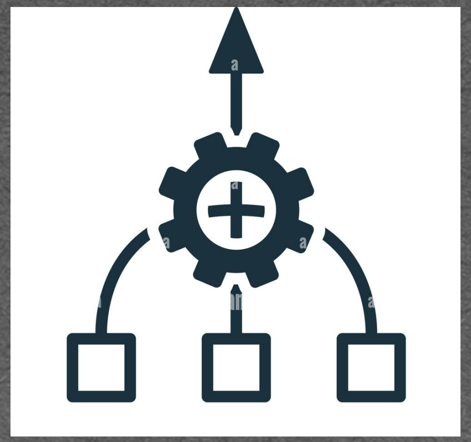
$\mathsf{Enc}(C, v^*, i, m) \rightarrow \mathsf{ct}$ such that Λ_i decrypts only if $\vec{v}[i] = v^*$

★ *In the paper: Efficient constructions from pairings and from LWE.*

The final compiler

1																			2^{256}
id_1			id_2	id_3				id_4					id_5				id_6		

↓ Cuckoo Hashing



1					n
$CH(id_5)$	$CH(id_3)$	$CH(id_2)$	$CH(id_6)$	$CH(id_1)$	$CH(id_4)$

→ $\widetilde{RBE}.\mathrm{Reg}(\ldots, CH(id), \ldots)$

$\widetilde{RBE}.\mathrm{Enc}(\ldots, H_1(id), m_1^{(1)}, \ldots)$

$\widetilde{RBE}.\mathrm{Enc}(\ldots, H_2(id), m_1^{(2)}, \ldots)$

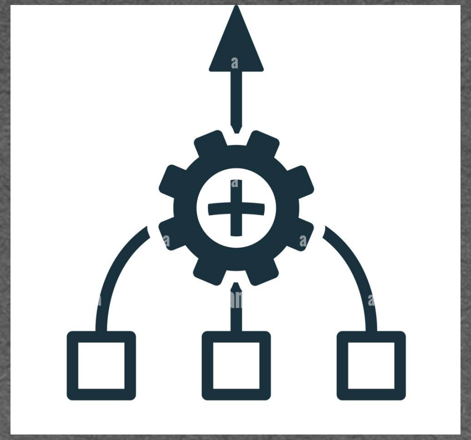
$\vdots$

$\widetilde{RBE}.\mathrm{Enc}(\ldots, H_k(id), m_1^{(k)}, \ldots)$

The final compiler

1																			2^{256}
id_1			id_2	id_3				id_4					id_5				id_6		

Cuckoo Hashing



1					n
$CH(id_5)$	$CH(id_3)$	$CH(id_2)$	$CH(id_6)$	$CH(id_1)$	$CH(id_4)$
id_5	id_3	id_2	id_6	id_1	id_4

→ $\widetilde{\mathsf{RBE}}.\mathsf{Reg}(\ldots, \mathsf{CH}(\mathsf{id}), \ldots)$

→ $D \leftarrow \mathsf{VC}.\mathsf{Commit}(\ldots)$

$\widetilde{\mathsf{RBE}}.\mathsf{Enc}(\ldots, \mathsf{H}_1(\mathsf{id}), m_1^{(1)}, \ldots)$

$\widetilde{\mathsf{RBE}}.\mathsf{Enc}(\ldots, \mathsf{H}_2(\mathsf{id}), m_1^{(2)}, \ldots)$

⋮

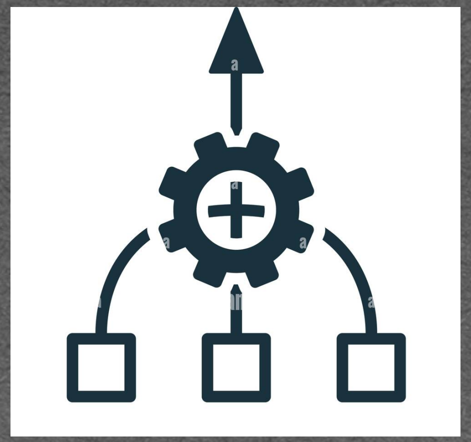
$\widetilde{\mathsf{RBE}}.\mathsf{Enc}(\ldots, \mathsf{H}_k(\mathsf{id}), m_1^{(k)}, \ldots)$

The final compiler

1 ... 2^{256}

id_1			id_2	id_3				id_4					id_5				id_6	

Cuckoo Hashing



1 ... n

$\mathsf{CH}(\mathsf{id}_5)$	$\mathsf{CH}(\mathsf{id}_3)$	$\mathsf{CH}(\mathsf{id}_2)$	$\mathsf{CH}(\mathsf{id}_6)$	$\mathsf{CH}(\mathsf{id}_1)$	$\mathsf{CH}(\mathsf{id}_4)$
id_5	id_3	id_2	id_6	id_1	id_4

$\to \widetilde{\mathsf{RBE}}.\mathsf{Reg}(\ldots, \mathsf{CH}(\mathsf{id}), \ldots)$

$\to D \leftarrow \mathsf{VC}.\mathsf{Commit}(\ldots)$

$\mathsf{SS}.\mathsf{Share}(m) \to (m_1^{(i)}, m_2^{(i)})$

$\widetilde{\mathsf{RBE}}.\mathsf{Enc}(\ldots, \mathsf{H}_1(\mathsf{id}), m_1^{(1)}, \ldots)$ $\quad$ $\mathsf{VCWE}.\mathsf{Enc}(D, \mathsf{H}_1(\mathsf{id}), \mathsf{id}, m_2^{(1)}, \ldots)$

$\widetilde{\mathsf{RBE}}.\mathsf{Enc}(\ldots, \mathsf{H}_2(\mathsf{id}), m_1^{(2)}, \ldots)$ $\quad$ $\mathsf{VCWE}.\mathsf{Enc}(D, \mathsf{H}_2(\mathsf{id}), \mathsf{id}, m_2^{(2)}, \ldots)$

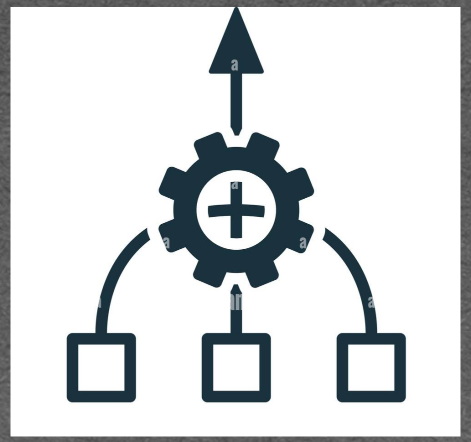
$\vdots$ $\quad$ $\vdots$

$\widetilde{\mathsf{RBE}}.\mathsf{Enc}(\ldots, \mathsf{H}_k(\mathsf{id}), m_1^{(k)}, \ldots)$ $\quad$ $\mathsf{VCWE}.\mathsf{Enc}(D, \mathsf{H}_k(\mathsf{id}), \mathsf{id}, m_2^{(k)}, \ldots)$

The final compiler

1																		2^{256}
id_1			id_2	id_3				id_4					id_5				id_6	

Cuckoo Hashing



1					n
$\mathsf{CH}(\mathsf{id}_5)$	$\mathsf{CH}(\mathsf{id}_3)$	$\mathsf{CH}(\mathsf{id}_2)$	$\mathsf{CH}(\mathsf{id}_6)$	$\mathsf{CH}(\mathsf{id}_1)$	$\mathsf{CH}(\mathsf{id}_4)$
id_5	id_3	id_2	id_6	id_1	id_4

→ $\widetilde{\mathsf{RBE}}.\mathsf{Reg}(\ldots, \mathsf{CH}(\mathsf{id}), \ldots)$

→ $D \leftarrow \mathsf{VC}.\mathsf{Commit}(\ldots)$

$\mathsf{SS}.\mathsf{Share}(m) \rightarrow (m_1^{(i)}, m_2^{(i)})$

Maybe id′	$\widetilde{\mathsf{RBE}}.\mathsf{Enc}(\ldots, \mathsf{H}_1(\mathsf{id}), m_1^{(1)}, \ldots)$	$\mathsf{VCWE}.\mathsf{Enc}(D, \mathsf{H}_1(\mathsf{id}), \mathsf{id}, m_2^{(1)}, \ldots)$	Nobody!
Only id	$\widetilde{\mathsf{RBE}}.\mathsf{Enc}(\ldots, \mathsf{H}_2(\mathsf{id}), m_1^{(2)}, \ldots)$	$\mathsf{VCWE}.\mathsf{Enc}(D, \mathsf{H}_2(\mathsf{id}), \mathsf{id}, m_2^{(2)}, \ldots)$	Everyone
	$\vdots$	$\vdots$	
Maybe id′	$\widetilde{\mathsf{RBE}}.\mathsf{Enc}(\ldots, \mathsf{H}_k(\mathsf{id}), m_1^{(k)}, \ldots)$	$\mathsf{VCWE}.\mathsf{Enc}(D, \mathsf{H}_k(\mathsf{id}), \mathsf{id}, m_2^{(k)}, \ldots)$	Nobody!

Dealing with CH insertion failures

How to deal with $\Pr[\mathsf{Insert}(\mathsf{id}) = \bot\,] = \alpha$?

Dealing with CH insertion failures

How to deal with $\Pr[\mathsf{Insert}(\mathbf{id}) = \bot\,] = \alpha$?

★ *Solution 1: Cuckoo hashing with a stash*

- $k = 2, s = O(\log(n)) \Rightarrow \alpha = \mathsf{negl}(\lambda)$
- Selective adversary (choice of **id** before seeing h_1, h_2)

★ *Solution 2: Robust Cuckoo Hashing [Yeo23]*

- $k = \lambda \Rightarrow \alpha = \mathsf{negl}(\lambda)$
- Adaptive adversary

The resulting RBE schemes

	Setting	$\mathcal{ID}$	Compactness	$\lvert\mathsf{ct}\rvert$	#updates	$\lvert\mathsf{pp}\rvert + \lvert\mathsf{crs}\rvert$
[HLWW23]	Pairings (C)	$\{0,1\}^*$	Adaptive	$O(\lambda \log n)$	$\log n$	$O(\lambda n^{2/3} \log n)$
[GKMR22]	Pairings (P)	$[1,n]$	Adaptive	$4 \log n$	$\log n$	$O(\sqrt{n} \log n)$
Ours P1	Pairings (P)	$\{0,1\}^*$	Adaptive	$6\lambda \log n$	$\log n$	$O(\sqrt{\lambda n} \log n)$
Ours P2	Pairings (P)	$\{0,1\}^*$	Selective	$12 \log n$	$\log n$	$O(\sqrt{n} \log n)$
[DKL+23]	Lattices	$\{0,1\}^*$	Adaptive	$(2\lambda + 1) \log n$	$\log n$	$O(\log n)$
Ours L	Lattices	$\{0,1\}^*$	Selective	$4 \log^2 n$	$\log n$	$O(\log n)$

Pairings: $\lvert\mathsf{ct}\rvert$ measured in group elements

Lattices: $\lvert\mathsf{ct}\rvert$ measured in LWE ciphertexts

Our KVC compiler

Key-Value Map Commitments (KVC) from any Vector Commitment (VC)

1																		2^{256}
v_1			v_2	v_3				v_4					v_5				v_6	
k_1			k_2	k_3				k_4					k_5				k_6	

Cuckoo Hashing

1					n
v_5	v_3	v_2	v_6	v_1	v_4
$CH(k_5)$	$CH(k_3)$	$CH(k_2)$	$CH(k_6)$	$CH(k_1)$	$CH(k_4)$

VC.Commit

Open k $\longrightarrow$

$v[H_1(k)]$,  VC.Open

$v[H_2(k)]$,  VC.Open

⋮

KVC: Commit to $[(k_1, v_1), \dots, (k_n, v_n)]$

Open any key k_i

Conclusions and Open Problems

Conclusions and Open problems

Summary:

- ❖ Efficient pairing-based RBE with unbounded identities
- ❖ Key-Value Map Commitments from any Vector Commitment

Coclusions:

- ❖ Cuckoo-Hashing very powerful and under-explored (in PKE)

Open Problems:

- ❖ Pairing-based RBE with unbounded identity space without $O(\lambda)$ overhead in |ct|
- ❖ Lattice-based RBE with optimal $O(\log n)$ |ct|
- ❖ Pairing-based KVC with $O(1)$ openings

Thank you!

ePrint: https://eprint.iacr.org/2023/1389.pdf