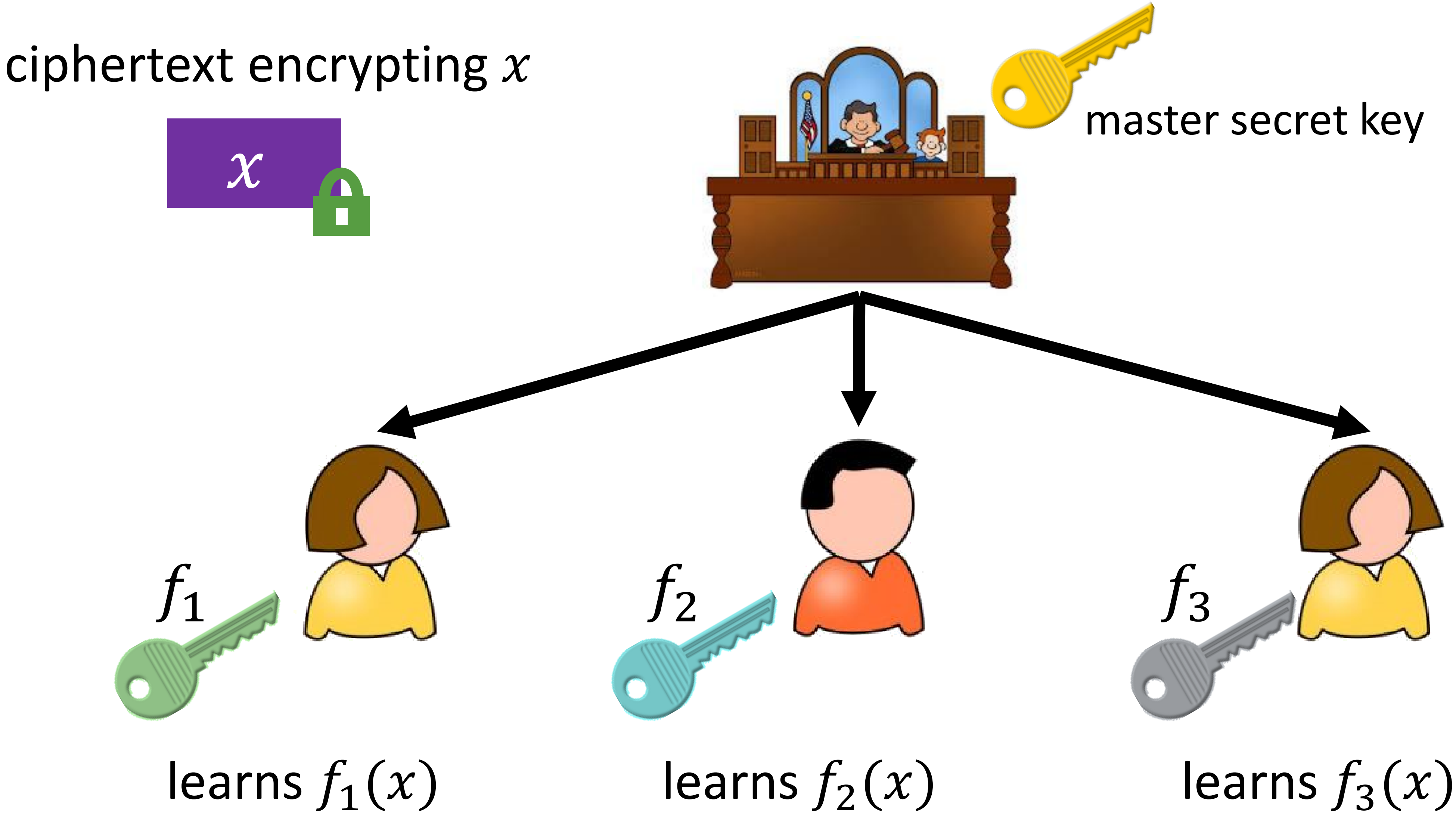
Removing Trust Assumptions from Advanced Encryption Schemes

David Wu

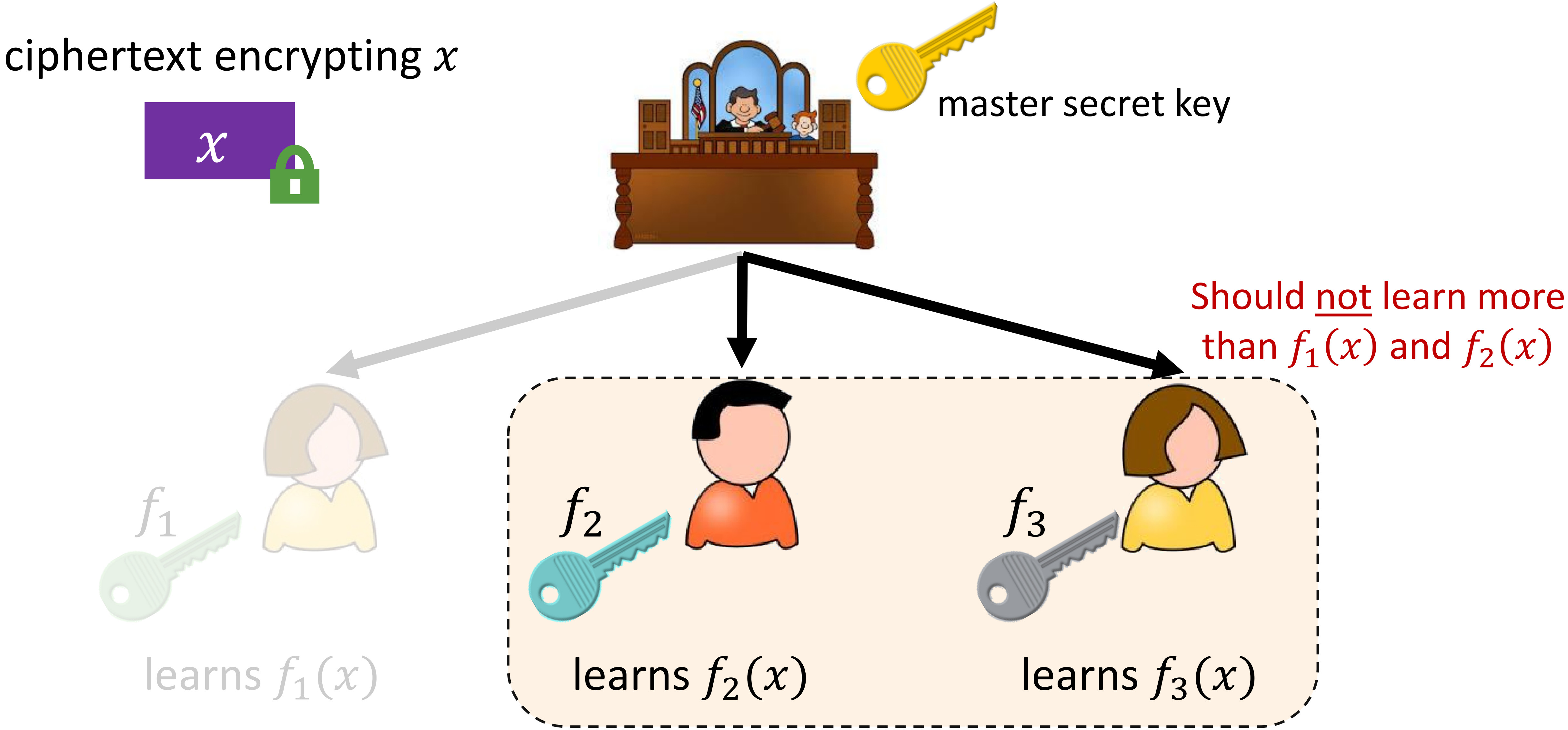
Functional Encryption (FE)

[SS10, O'N10, BSW11]



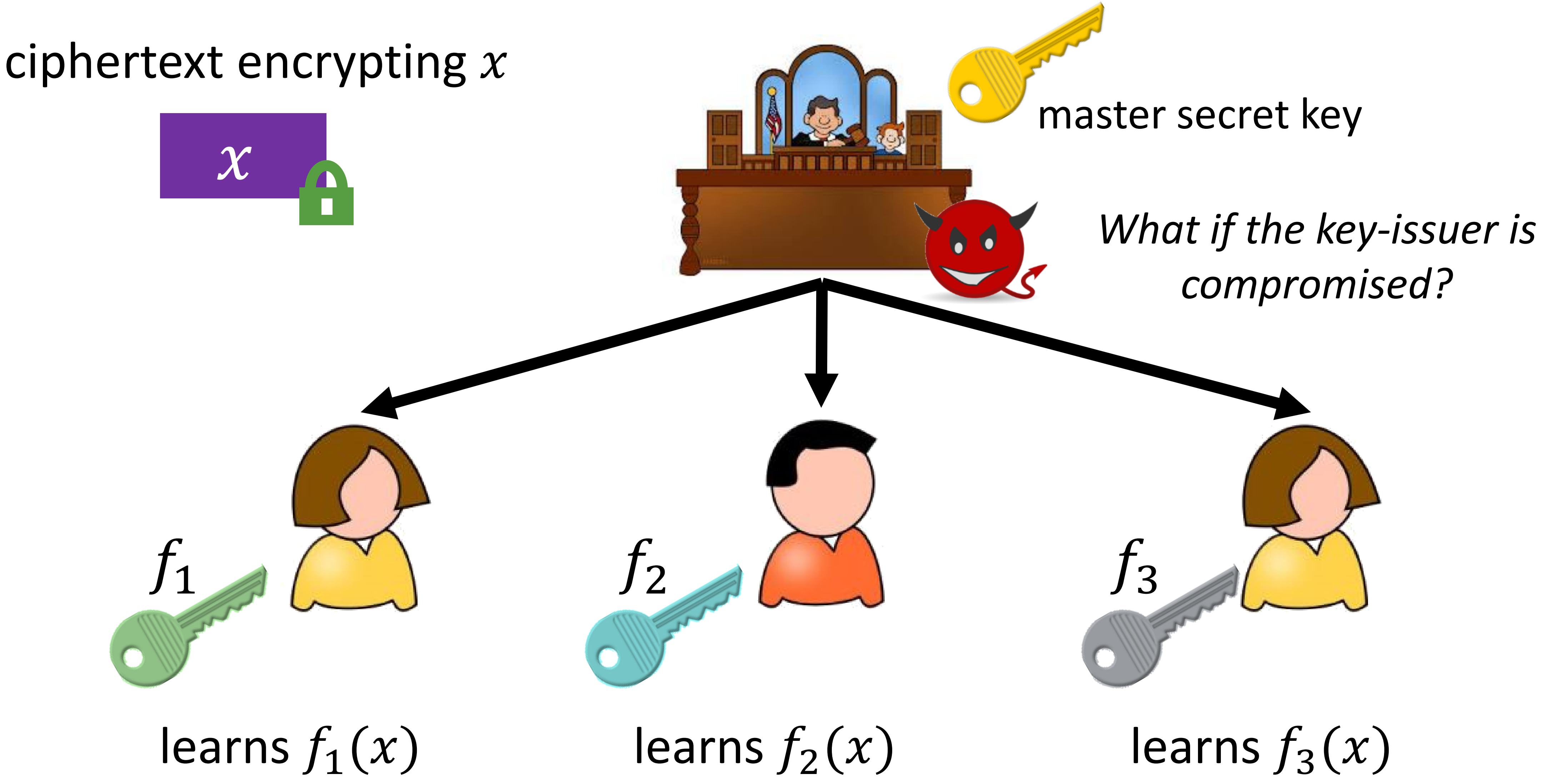
Functional Encryption (FE)

[SS10, O'N10, BSW11]



Functional Encryption (FE)

[SS10, O'N10, BSW11]



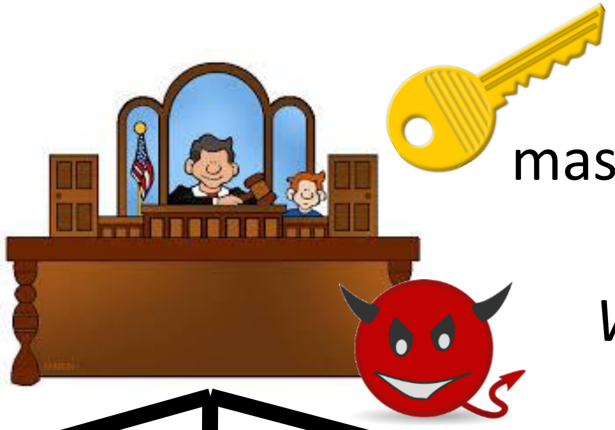
Functional Encryption (FE)

[SS10, O'N10, BSW11]

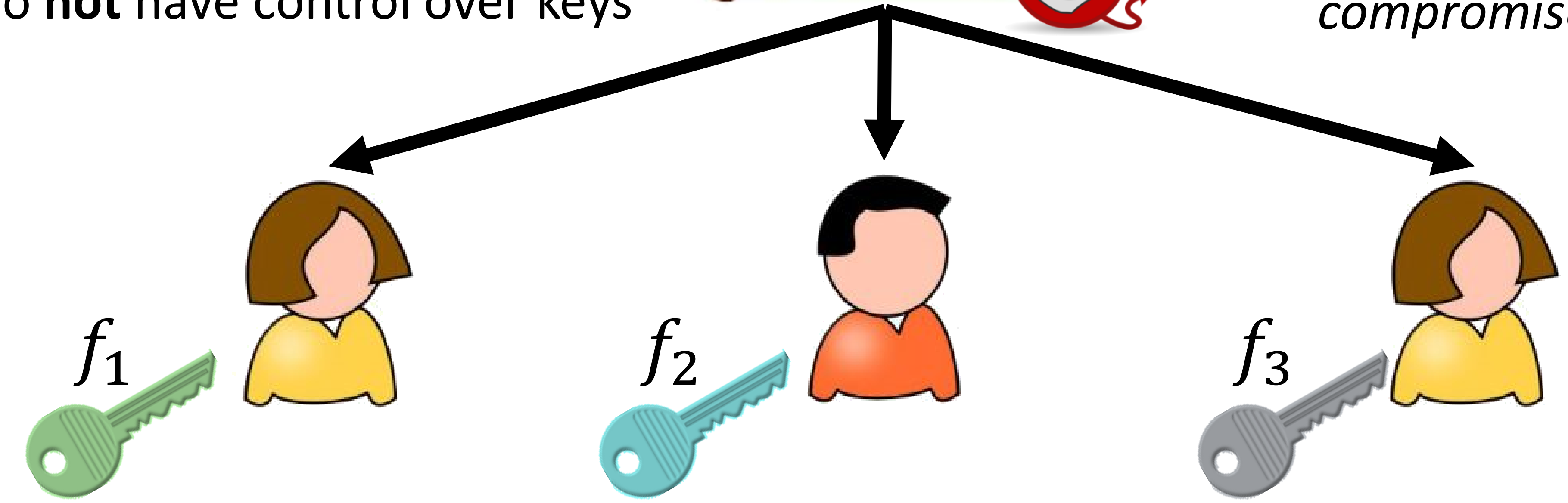
Key issuer can decrypt **all** ciphertexts

Central point of failure

Users do **not** have control over keys



What if the key-issuer is compromised?



learns $f_1(x)$ learns $f_2(x)$ learns $f_3(x)$

Functional Encryption vs. Public-Key Encryption

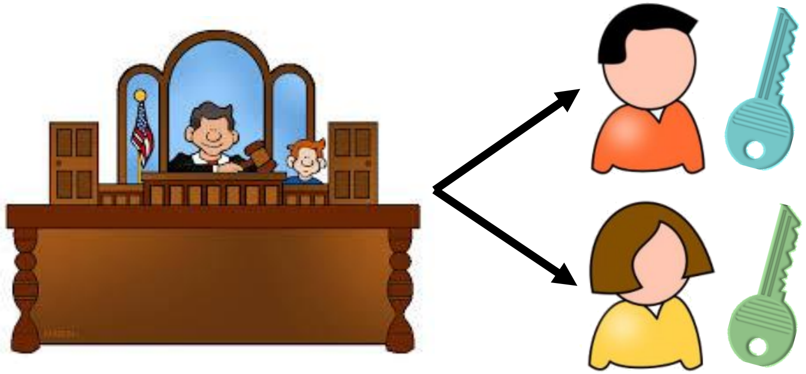
Public-key encryption is **decentralized**

Can we get the best of both worlds?

Every user generates their own key (no coordination or trust needed)

Does **not** support fine-grained decryption

Functional encryption is **centralized**



Central (trusted) authority generates individual keys

Supports **fine-grained** decryption capabilities

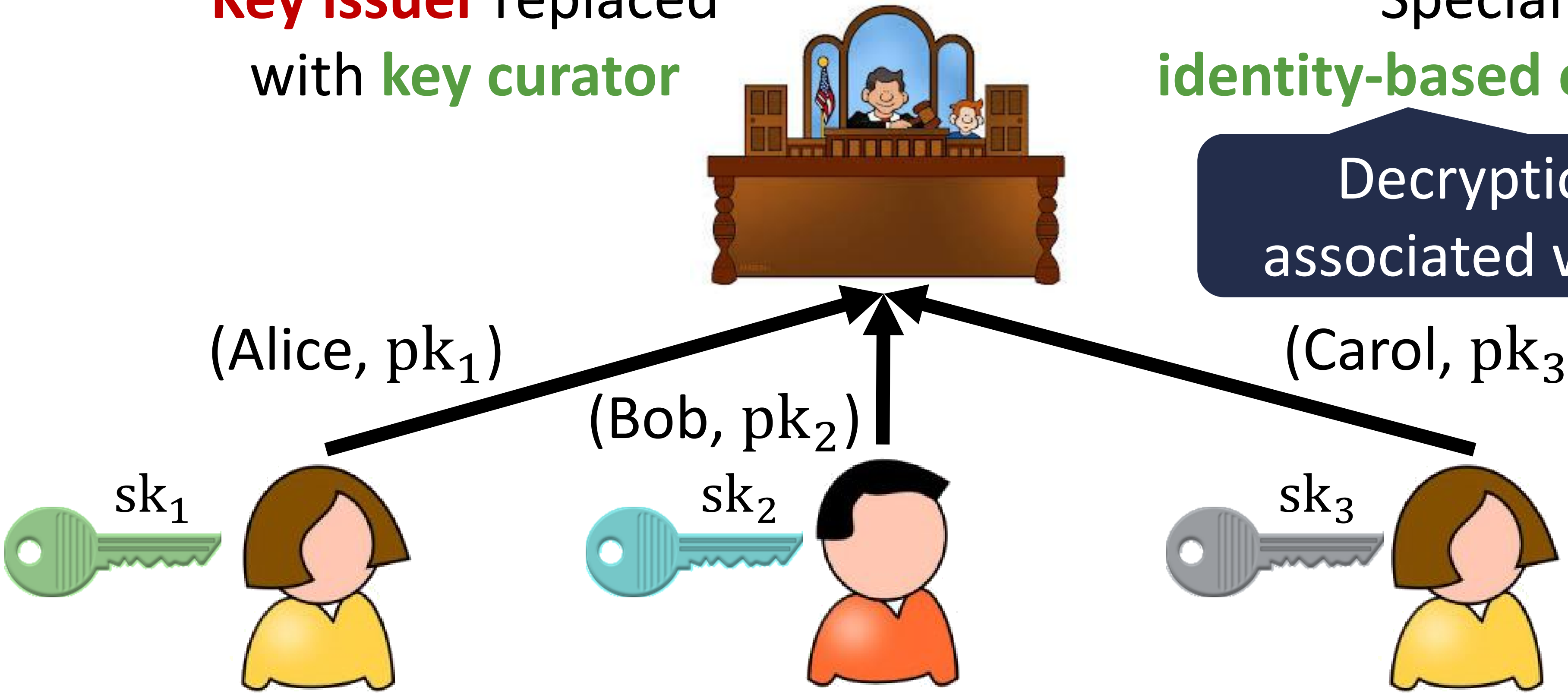
Registration-Based Encryption (RBE)

[GHMR18]

Key issuer replaced with **key curator**

Special case of **identity-based encryption** (IBE)

Decryption keys are associated with **identities**



Users chooses their <u>own</u> public/secret key and **register** their public key with the curator

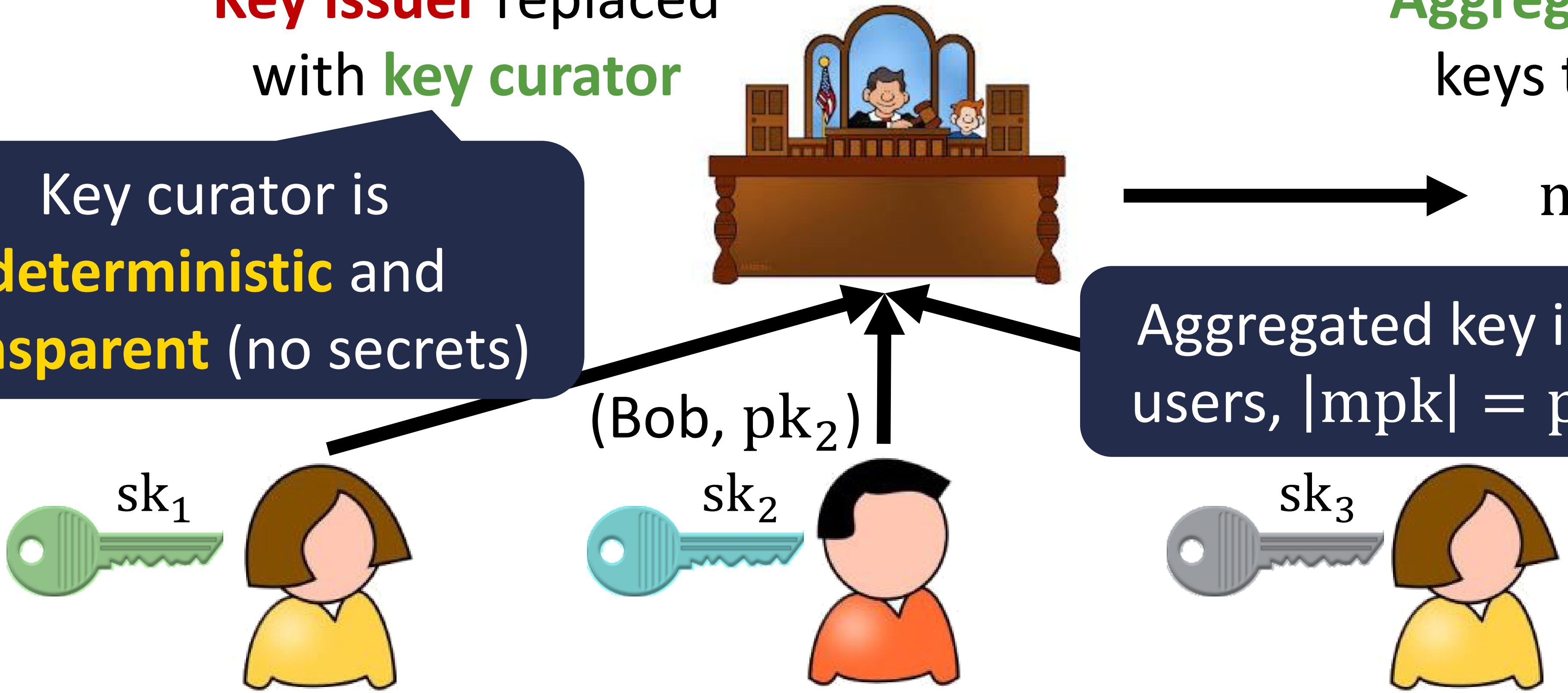
Registration-Based Encryption (RBE)

[GHMR18]

Key issuer replaced with **key curator**

Aggregate public keys together

Key curator is **deterministic** and **transparent** (no secrets)



Aggregated key is **short**: for L users, $|\mathrm{mpk}| = \mathrm{poly}(\lambda, \log L)$

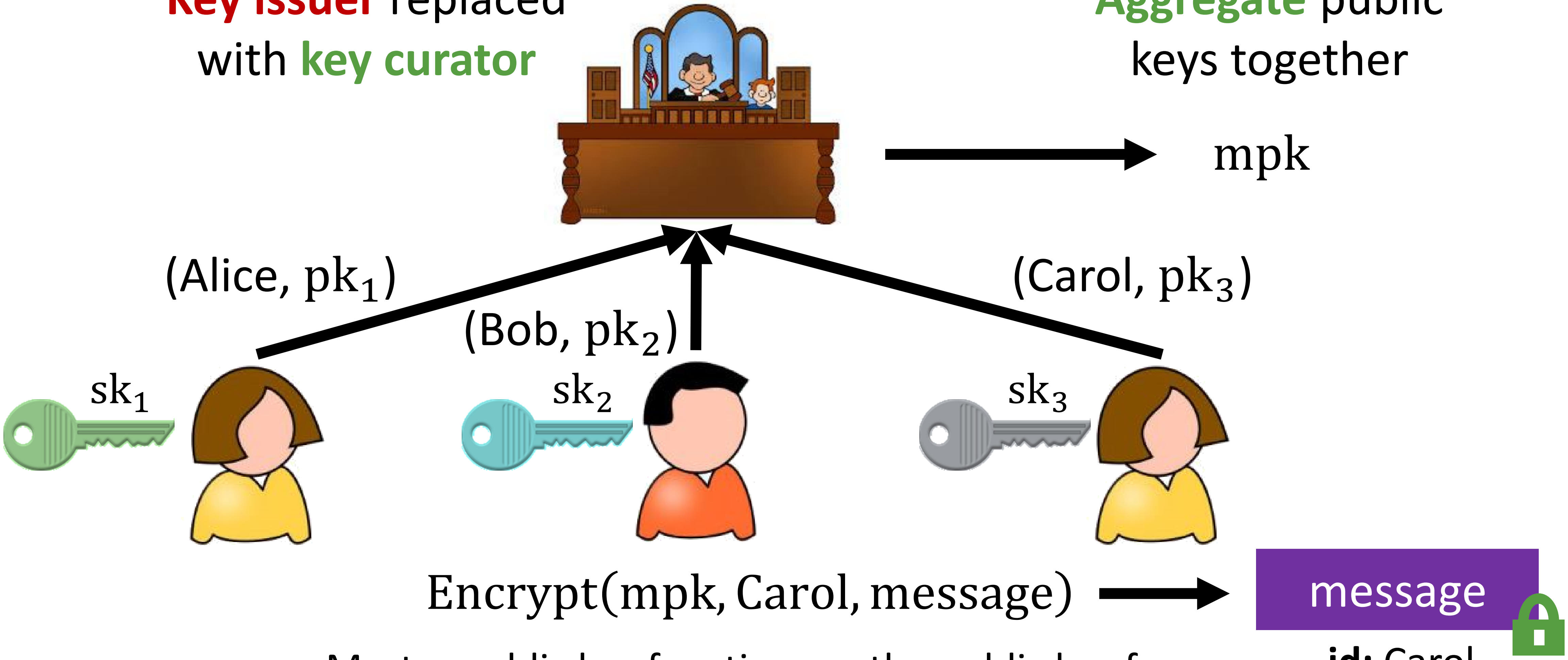
Users chooses their <u>own</u> public/secret key and **register** their public key with the curator

Registration-Based Encryption (RBE)

[GHMR18]

Key issuer replaced with **key curator**

Aggregate public keys together



Master public key functions as the public key for an identity-based encryption scheme

Registration-Based Encryption (RBE)

[GHMR18]

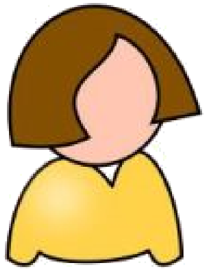
Key issuer replaced with **key curator**

Aggregate public keys together

$\longrightarrow$ mpk

(Alice, pk)

sk



hsk

To decrypt, users periodically retrieve a helper decryption key hsk (function of mpk and user's **public** key pk_1)

$|\mathrm{hsk}| = \mathrm{poly}(\lambda, \log L)$

Note: As users join, the master public key is updated, so users **occasionally** need to retrieve a new helper decryption key

$\#\ \text{key updates per user} = \mathrm{poly}(\lambda, \log L)$

Registration-Based Encryption (RBE)

[GHMR18]

Key issuer replaced with **key curator**

$\longrightarrow$ mpk

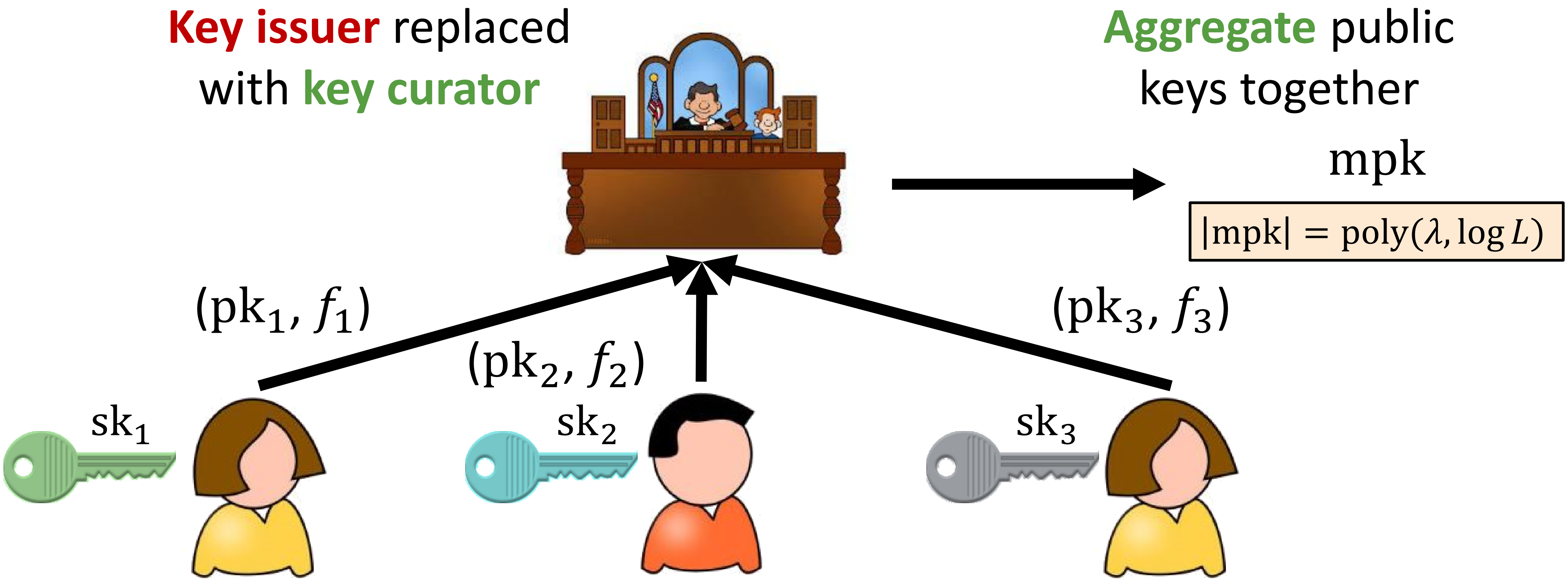
Aggregate public keys together

- Initial constructions based on indistinguishability obfuscation or hash garbling (based on CDH, QR, LWE) – all require **non-black-box** use of cryptography
- **High concrete efficiency costs:** ciphertext is 4.5 TB for supporting 2 billion users [CES21]

Can we construct RBE schemes that only need black-box use of cryptography?

Can we construct support more general policies (beyond identity-based encryption)?

Removing Trust from Functional Encryption



Users chooses their own key and **register** the public key (together with **function f**) with the curator

Note: f could also be chosen by the key curator

Removing Trust from Functional Encryption

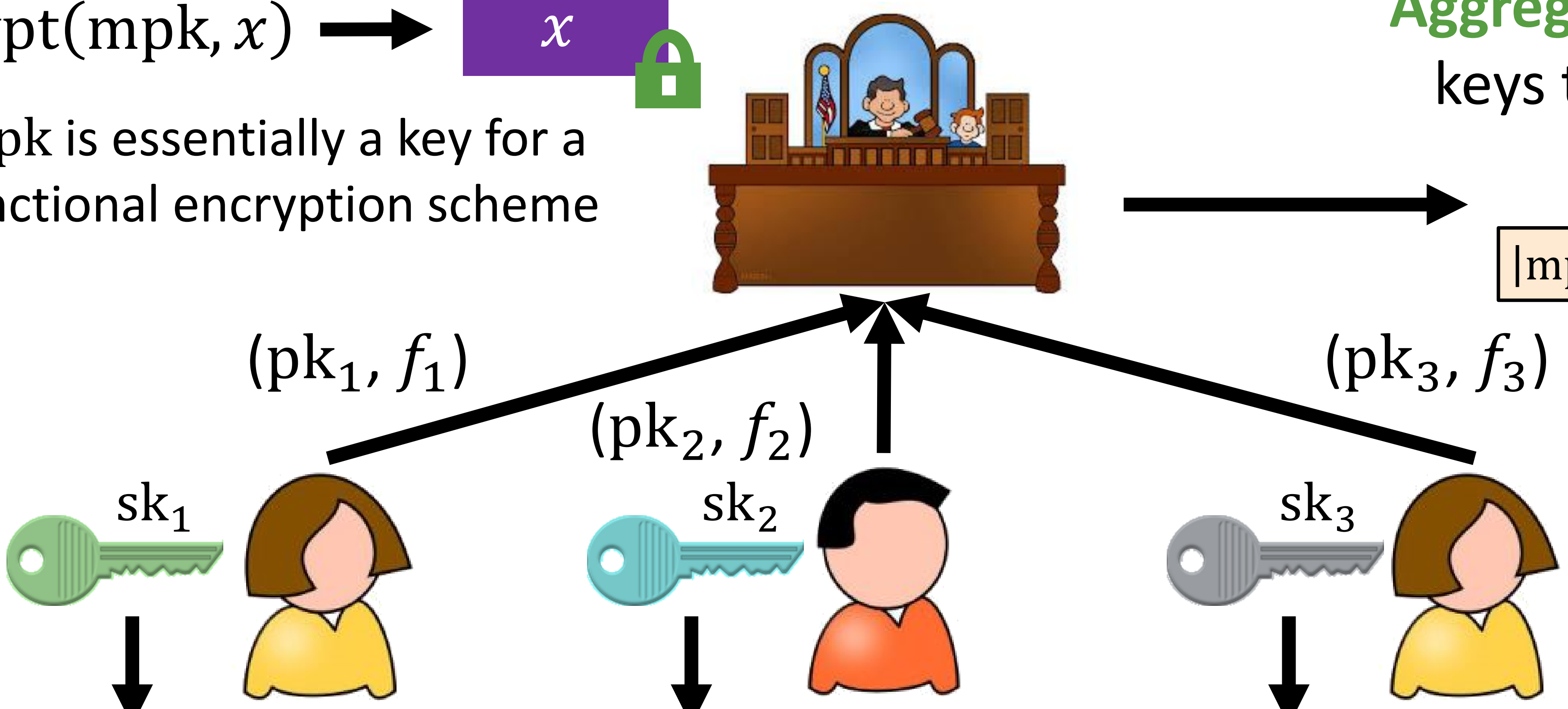
$\text{Encrypt}(\text{mpk}, x) \longrightarrow$ x

mpk is essentially a key for a functional encryption scheme

Aggregate public keys together

mpk

$\boxed{|\text{mpk}| = \text{poly}(\lambda, \log L)}$



$f_1(x)$ $f_2(x)$ $f_3(x)$

Registration-Based Cryptography

Can we construct RBE schemes that only need black-box use of cryptography? **Yes!**

Can we construct support more general policies (beyond identity-based encryption)? **Yes!**

Registration-based encryption [GHMR18, GHMMRS19, GV20, CES21, DKLLMR23, GKMR23, ZZGQ23, FKP23]

Registered attribute-based encryption (ABE)

- Monotone Boolean formulas [HLWW23, ZZGQ23, GLWW24]
- Inner products [FFMMRV23, ZZGQ23]
- Arithmetic branching program [ZZGQ23]
- Boolean circuits [HLWW23, FWW23]

This talk

Distributed/flexible broadcast [BZ14, KMW23, FWW23, GLWW23, GKPW24, CW24]

Lots of progress in this past year!

Registered traitor tracing [BLMMRW24]

Registered functional encryption

- Linear functions [DPY23]
- Quadratic functions [ZLZGQ24]
- Boolean circuits [FFMMRV23, DPY23]

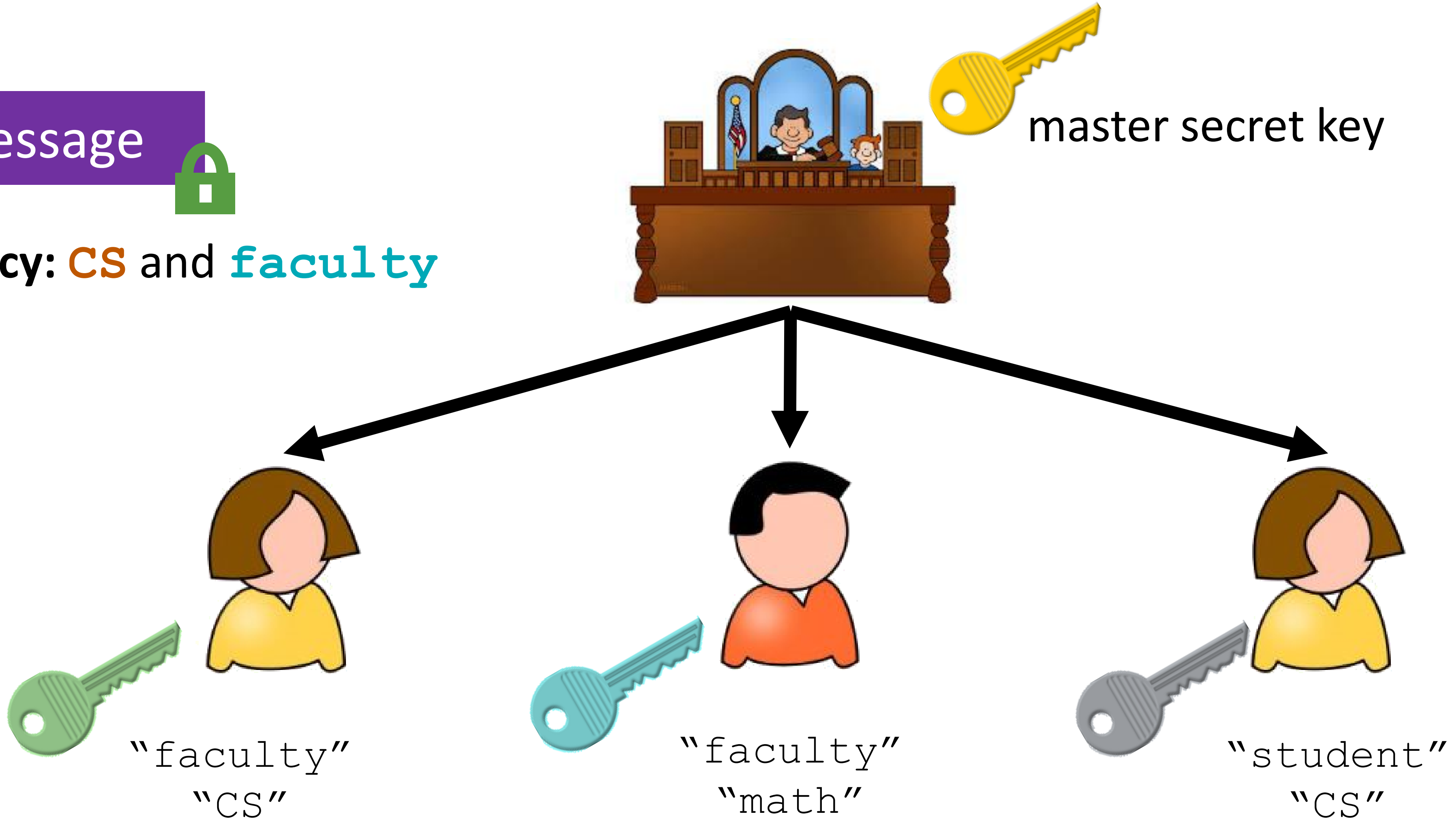
Underlined schemes only need **black-box** use of cryptography

Attribute-Based Encryption

[SW05, GPSW06]

message

policy: `CS` and `faculty`

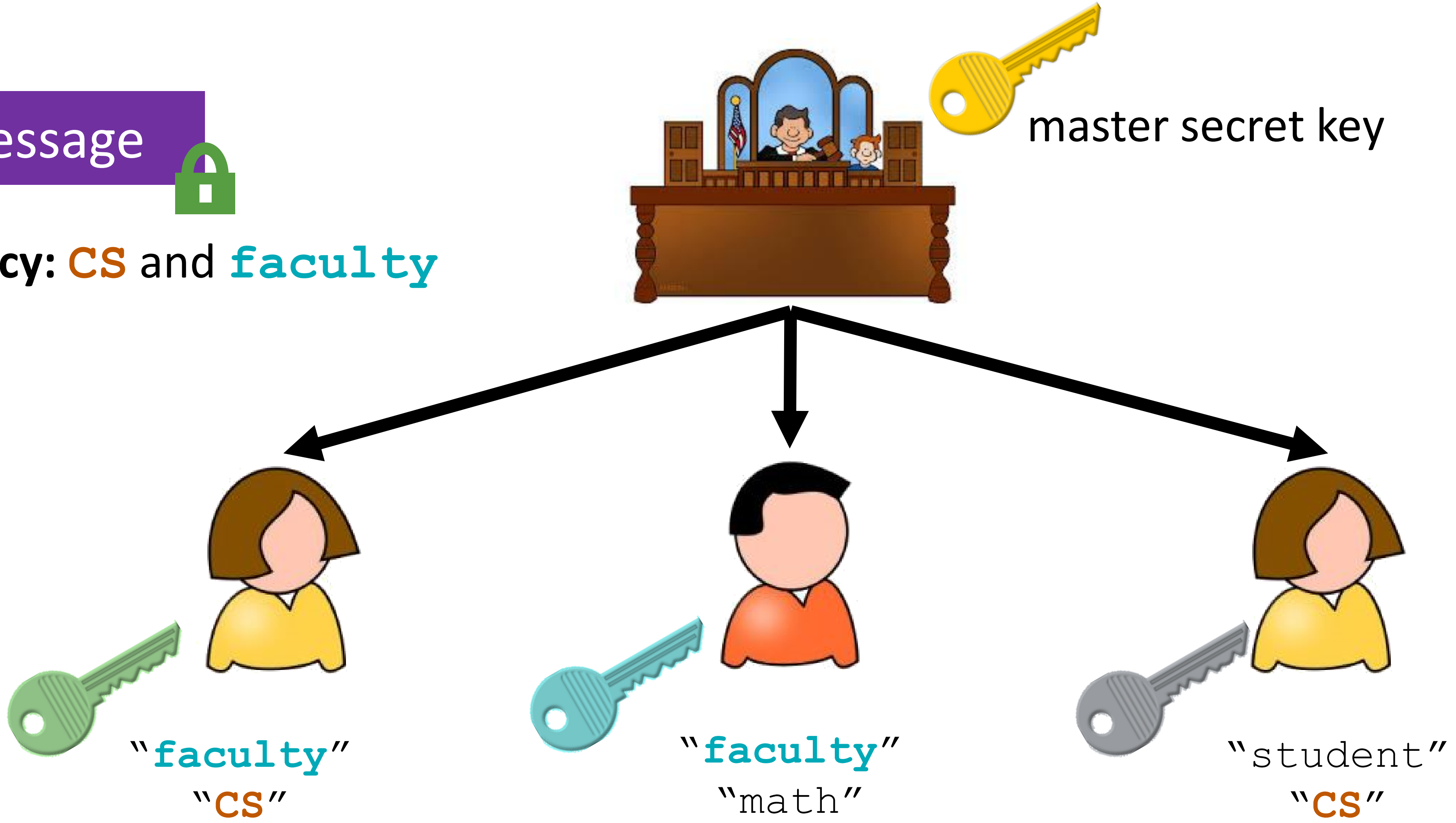


Attribute-Based Encryption

[SW05, GPSW06]

message

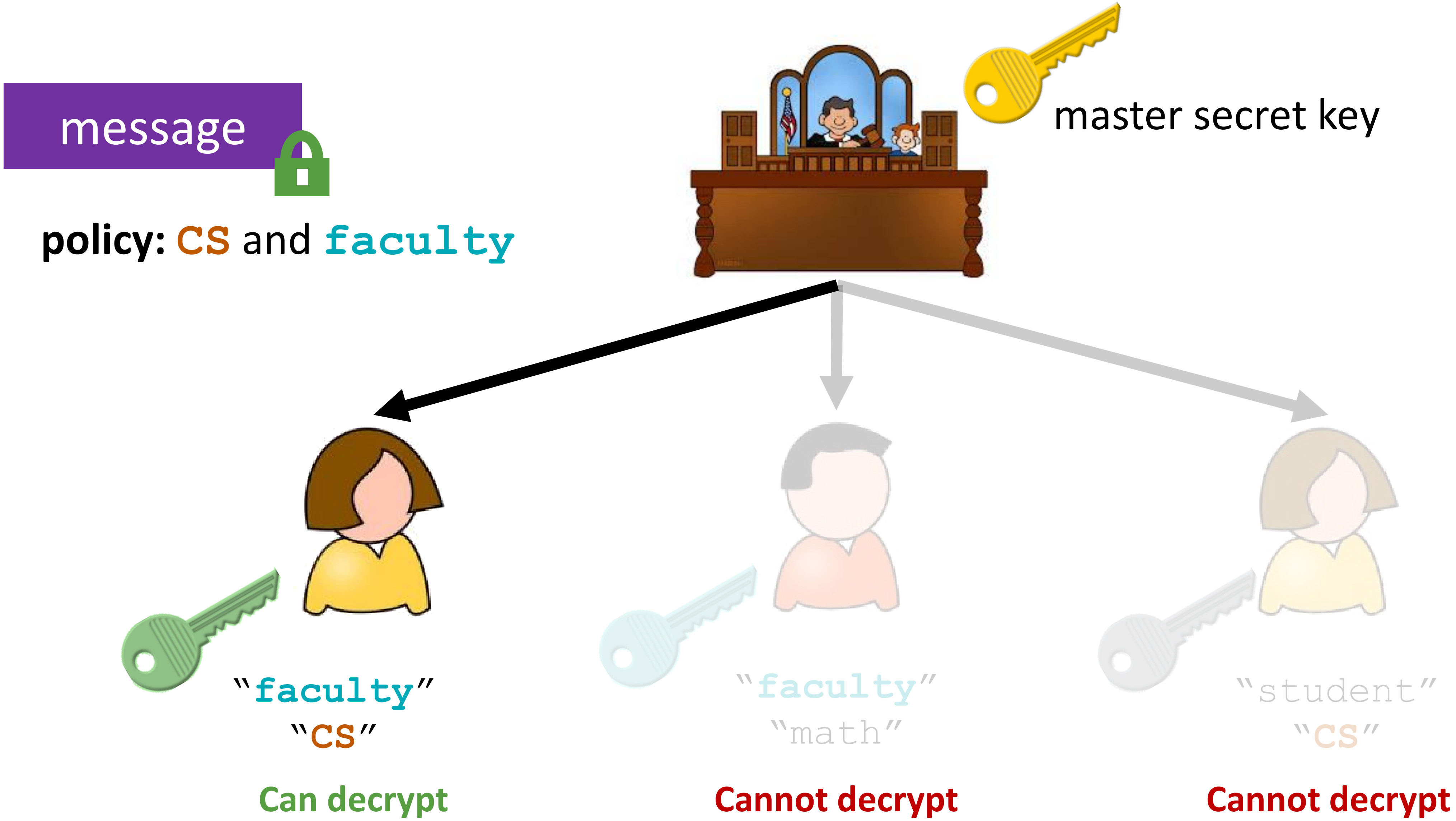
policy: CS and faculty



Can decrypt

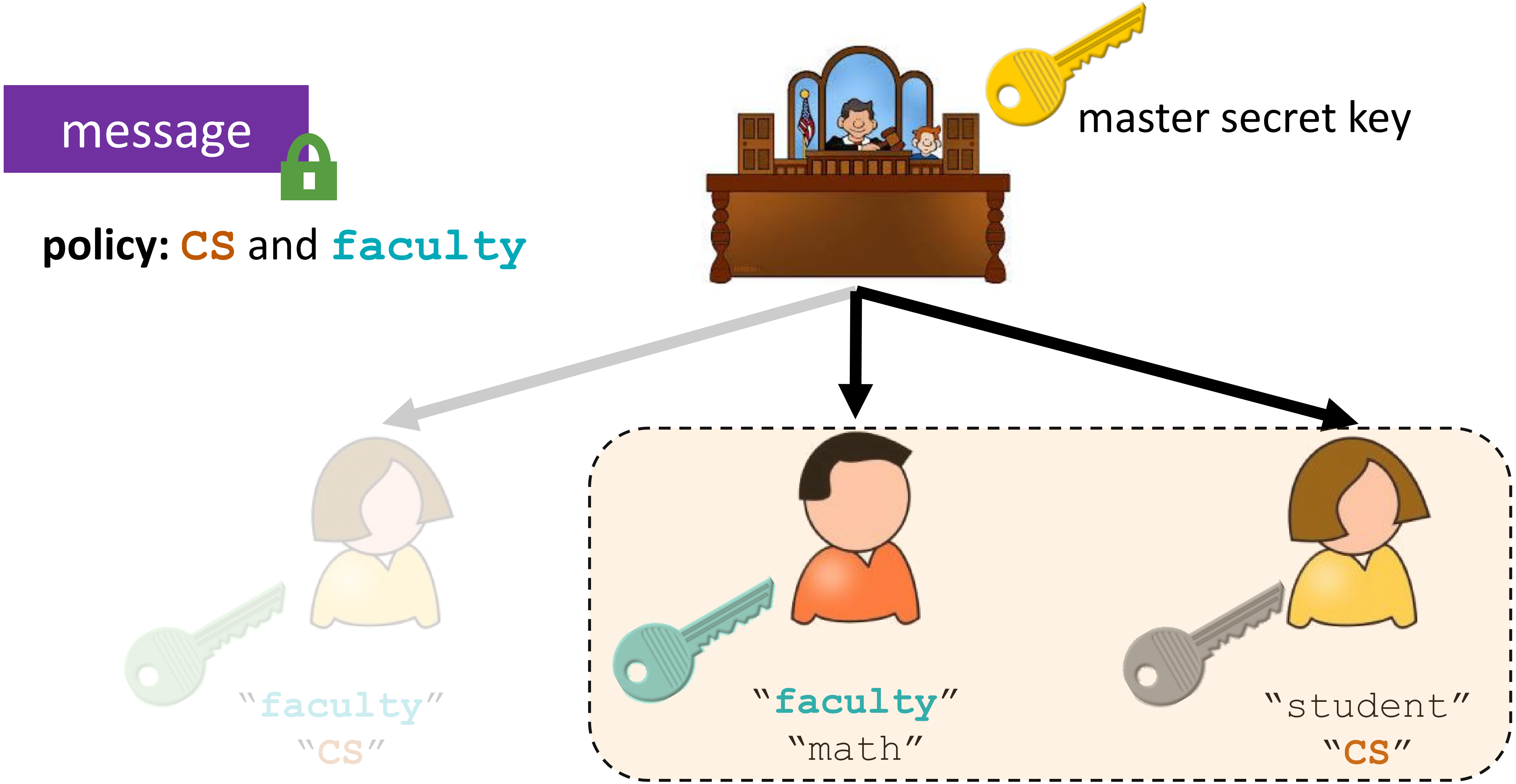
Attribute-Based Encryption

[SW05, GPSW06]



Attribute-Based Encryption

[SW05, GPSW06]



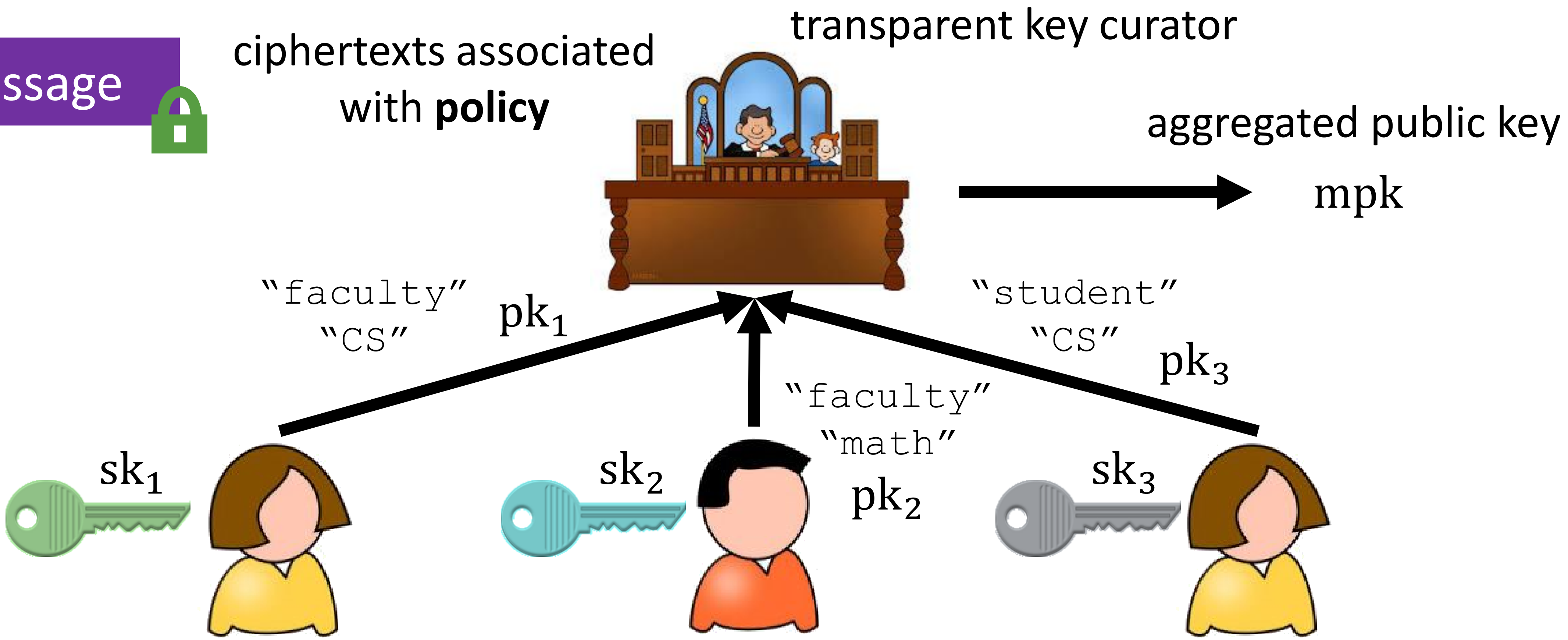
Users cannot collude to decrypt

Registered Attribute-Based Encryption

[HLWW23]

message

ciphertexts associated with **policy**

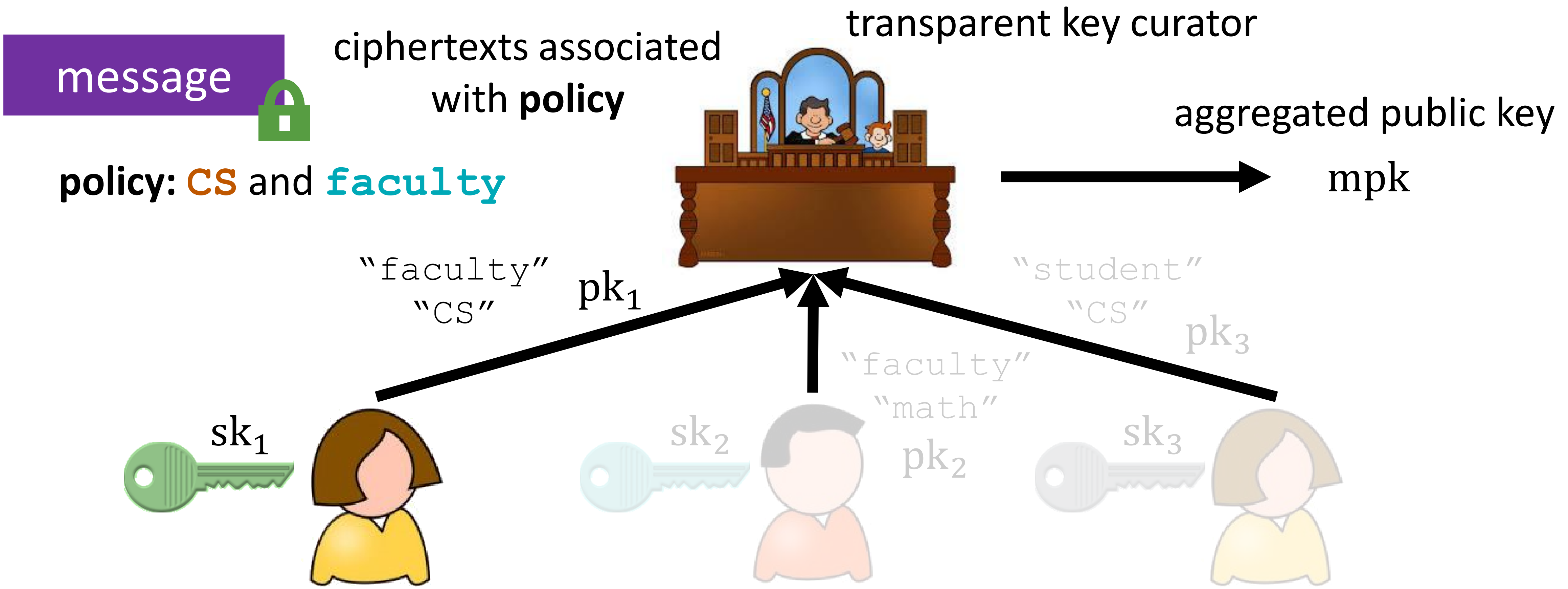


Users chooses their <u>own</u> public/secret key

Users join the system by registering their public key along with a set of attributes

Registered Attribute-Based Encryption

[HLWW23]



Users chooses their <u>own</u> public/secret key

Users join the system by registering their public key along with a set of attributes

A Template for Building Registered ABE

[HLWW23]

Simplification: assume that all of the users register at the **same** time (rather than in an online fashion)

Slotted registered ABE:

Let L be the number of users

pk_1, S_1	pk_2, S_2	pk_3, S_3	pk_4, S_4	$\cdots$	pk_L, S_L

$$\xrightarrow{\text{Aggregate}} \quad \begin{array}{c} \mathrm{mpk} \\ \mathrm{hsk}_1, \dots, \mathrm{hsk}_L \end{array}$$

Each slot associated with a <u>public key</u> pk and a set of attributes S

$$|\mathrm{mpk}| = \mathrm{poly}(\lambda, |\mathcal{U}|, \log L)$$

$$|\mathrm{hsk}_i| = \mathrm{poly}(\lambda, |\mathcal{U}|, \log L)$$

λ: security parameter

$\mathcal{U}$: universe of attributes

A Template for Building Registered ABE

[HLWW23]

Simplification: assume that all of the users register at the **same** time (rather than in an online fashion)

Slotted registered ABE:

Let L be the number of users

pk_1, S_1	pk_2, S_2	pk_3, S_3	pk_4, S_4	$\cdots$	pk_L, S_L

$$\xrightarrow{\text{Aggregate}} \quad \begin{array}{c} \mathrm{mpk} \\ \mathrm{hsk}_1, \ldots, \mathrm{hsk}_L \end{array}$$

Each slot associated with a <u>public key</u> pk and a set of attributes S

$\mathrm{Encrypt}(\mathrm{mpk}, P, m) \to \mathrm{ct}$ — Encryption takes master public key and policy P (<u>no slot</u>)

$\mathrm{Decrypt}(\mathrm{sk}_i, \mathrm{hsk}_i, \mathrm{ct}) \to m$ — Decryption takes secret key sk_i for some slot and the helper key hsk_i for that slot

A Template for Building Registered ABE

[HLWW23]

Simplification: assume that all of the users register at the **same** time (rather than in an online fashion)

Slotted registered ABE:

Let L be the number of users

pk_1, S_1	pk_2, S_2	pk_3, S_3	pk_4, S_4	$\cdots$	pk_L, S_L

$$\xrightarrow{\text{Aggregate}} \quad \mathrm{mpk}, \; \mathrm{hsk}_1, \ldots, \mathrm{hsk}_L$$

Each slot associated with a public key pk and a set of attributes S

$$\mathrm{Encrypt}(\mathrm{mpk}, P, m) \to \mathrm{ct}$$

$$\mathrm{Decrypt}(\mathrm{sk}_i, \mathrm{hsk}_i, \mathrm{ct}) \to m$$

Main difference with registered ABE:
Aggregate takes all L keys simultaneously

Slotted Registered ABE to Registered ABE

[HLWW23]

Let L be the number of users

pk_1, S_1	pk_2, S_2	pk_3, S_3	pk_4, S_4	$\cdots$	pk_L, S_L

Aggregate

$$\xrightarrow{\text{Aggregate}} \quad \begin{matrix} \mathrm{mpk} \\ \mathrm{hsk}_1, \dots, \mathrm{hsk}_L \end{matrix}$$

Slotted scheme does *not* support online registration

Solution: use “powers-of-two” approach (like [GHMR18])

Maintain $\log L$ slotted schemes, where scheme i supports 2^i users

Constructing Slotted Registered ABE

[GLWW24]

Construction will rely on a prime-order pairing group $(\mathbb{G}, \mathbb{G}_T)$

Pairing is an **efficiently-computable** bilinear map $e: \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ from $\mathbb{G}$ to $\mathbb{G}_T$:

$$e(g^x, g^y) = e(g, g)^{xy}$$

Multiplies exponents in the ***target group***

Constructing Slotted Registered ABE

[GLWW24]

Will consider a toy scheme with **two** slots and **two** attributes w_1, w_2

Policy will be "has attribute w_i"

Scheme will rely on a **structured** common reference string (CRS)

General components: $Z = e(g,g)^\alpha$ $h \leftarrow \mathbb{G}$

Slot components: each slot $i \in \{1,2\}$ will have a pair of group elements

(A_1, B_1)	(A_2, B_2)

$A_i = g^{t_i}$ $B_i = g^\alpha h^{t_i}$

Attribute component: for each slot, we have an attribute component $U_i = g^{u_i}$

U_1	U_2

t_i is a slot exponent

u_i is an attribute exponent

Constructing Slotted Registered ABE

[GLWW24]

General components: $Z = e(g, g)^{\alpha}$ $\quad h \leftarrow \mathbb{G}$

Slot components: (A_1, B_1) and (A_2, B_2) $\quad A_i = g^{t_i} \quad B_i = g^{\alpha} h^{t_i}$

Attribute component: U_1, U_2 $\quad U_i = g^{u_i}$

To decrypt a ciphertext, **two** properties should hold:

- User should have the secret key for slot i — *Enforced by the <u>slot</u> components*
- Attributes associated with slot i should satisfy the challenge policy

Enforced by the <u>attribute</u> components

Constructing Slotted Registered ABE

[GLWW24]

General components: $Z = e(g,g)^{\alpha}$ $\quad h \leftarrow \mathbb{G}$

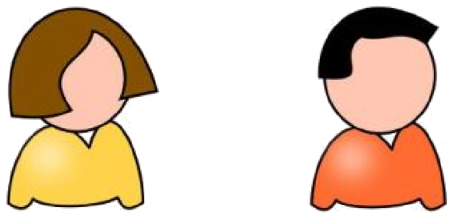
Slot components: (A_1, B_1) and (A_2, B_2) $\qquad A_i = g^{t_i} \quad B_i = g^{\alpha} h^{t_i}$

Attribute component: U_1, U_2 $\qquad U_i = g^{u_i}$

User's individual public/secret key is an ElGamal key-pair

$$\mathrm{sk} = r, \quad \mathrm{pk} = g^{r}$$ *(and some auxiliary information)*

Aggregating public keys $(\mathrm{pk}_1, \mathrm{pk}_2)$ with attribute sets S_1, S_2



$\mathrm{pk}_1 = g^{r_1}$ $\quad S_1 = \{1\}$

$\mathrm{pk}_2 = g^{r_2}$ $\quad S_2 = \{2\}$

aggregation $\longrightarrow$

Aggregated public key: $\hat{T} = \mathrm{pk}_1 \cdot \mathrm{pk}_2 = g^{r_1 + r_2}$

product of public keys

Key for attribute 1: $\widehat{U}_1 = g^{u_2}$

Key for attribute 2: $\widehat{U}_2 = g^{u_1}$

product of attribute components for slots that do not contain the attribute

Constructing Slotted Registered ABE

[GLWW24]

General components: $Z = e(g,g)^{\alpha}$ $\quad h \leftarrow \mathbb{G}$

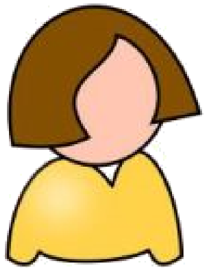
Slot components: $A_i = g^{t_i},\ B_i = g^{\alpha} h^{t_i}$

Attribute component: $U_1 = g^{u_1},\ U_2 = g^{u_2}$

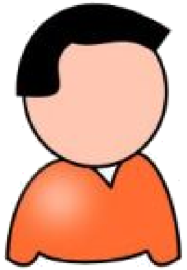
<u>Aggregated master public key</u>

$\hat{T} = g^{r_1 + r_2}$

$\hat{U}_1 = g^{u_2},\ \hat{U}_2 = g^{u_1}$



$\mathrm{pk}_1 = g^{r_1}$
$S_1 = \{1\}$



$\mathrm{pk}_1 = g^{r_2}$
$S_1 = \{2\}$

Ciphertext: $s \leftarrow \mathbb{Z}_p, h_1, h_2 \leftarrow \mathbb{G}$ such that $h_1 h_2 = h$

Suppose we encrypt μ to the policy "has attribute 1"

General components: $\mu \cdot Z^s, g^s$

Slot component: $h_1^s \hat{T}^s$

Attribute component: $h_2^s\, \hat{U}_1^s$

Constructing Slotted Registered ABE

[GLWW24]

General components: $Z = e(g,g)^{\alpha}$ $\quad h \leftarrow \mathbb{G}$

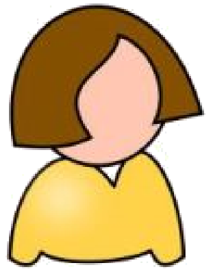
Slot components: $A_i = g^{t_i},\ B_i = g^{\alpha} h^{t_i}$

Attribute component: $U_1 = g^{u_1},\ U_2 = g^{u_2}$

Aggregated master public key

$\hat{T} = g^{r_1 + r_2}$

$\hat{U}_1 = g^{u_2},\ \hat{U}_2 = g^{u_1}$



$\mathrm{pk}_1 = g^{r_1}$
$S_1 = \{1\}$

General components: $\mu \cdot Z^s, g^s$

Slot component: $h_1^s \hat{T}^s$

Attribute component: $h_2^s\, \hat{U}_1^s$

Goal: recover μ

Step 1: Compute $e(g^s, B_1) = e(g,g)^{\alpha s} e(g,h)^{s t_i} = Z^s \cdot e(g,h)^{s t_i}$

Need to cancel out this component

Observe: ciphertext contains a secret share of $h^s = (h_1 h_2)^s$, but blinded by slot component $\hat{T}$ and attribute component $\hat{U}$

Constructing Slotted Registered ABE

[GLWW24]

General components: $Z = e(g,g)^{\alpha}$ $\quad h \leftarrow \mathbb{G}$

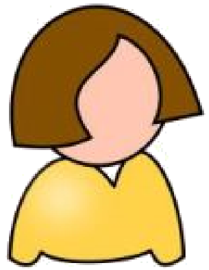
Slot components: $A_i = g^{t_i},\ B_i = g^{\alpha} h^{t_i}$

Attribute component: $U_1 = g^{u_1},\ U_2 = g^{u_2}$

Aggregated master public key

$\hat{T} = g^{r_1 + r_2}$

$\hat{U}_1 = g^{u_2},\ \hat{U}_2 = g^{u_1}$



$\mathrm{pk}_1 = g^{r_1}$
$S_1 = \{1\}$

General components: $\mu \cdot Z^s,\ g^s$

Slot component: $h_1^s \hat{T}^s$

Attribute component: $h_2^s\ \hat{U}_1^s$

Goal: recover μ

Can compute using secret key r_1

Step 1: Compute $e(g^s, B_1) = e(g,g)^{\alpha s} e(g,h)^{s t_i} = Z^s \cdot e(g,h)^{s t_1}$

Step 2 (Slot Check): Compute $e\left(A_1, h_1^s \hat{T}^s\right) = e\left(g^{t_1}, h_1^s \hat{T}^s\right) = e(g,h_1)^{s t_1} e(g,g)^{s r_1 t_1} e(g,g)^{s r_2 t_1}$

Given cross-term $e(g,g)^{r_2 t_1}$, can recover $e(g,h_1)^{s t_1}$

Share of $e(g,h)^{s t_1}$

Cross term from Party 2

Constructing Slotted Registered ABE

[GLWW24]

General components: $Z = e(g,g)^{\alpha}$ $h \leftarrow \mathbb{G}$

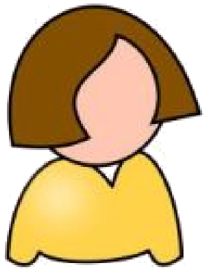
Slot components: $A_i = g^{t_i},\ B_i = g^{\alpha} h^{t_i}$

Attribute component: $U_1 = g^{u_1},\ U_2 = g^{u_2}$

Aggregated master public key

$\hat{T} = g^{r_1 + r_2}$

$\widehat{U}_1 = g^{u_2},\ \widehat{U}_2 = g^{u_1}$



$\mathrm{pk}_1 = g^{r_1}$
$S_1 = \{1\}$

General components: $\mu \cdot Z^s,\ g^s$

Slot component: $h_1^s \hat{T}^s$

Attribute component: $h_2^s\, \widehat{U}_1^s$

Goal: recover μ

Concretely: User in slot j would compute $A_i^{r_j} = g^{t_i r_j}$ for all $i \neq j$

$^{st_i} = Z^s \cdot e(g,h)^{st_1}$

Can compute using secret key r_1

$(g^{t_1}, h_1^s \hat{T}^s) = e(g,h_1)^{st_1} e(g,g)^{sr_1 t_1} e(g,g)^{sr_2 t_1}$

Given cross-term $e(g,g)^{r_2 t_1}$, can recover $e(g,h_1)^{st_1}$

Share of $e(g,h)^{st_1}$

Cross term from Party 2

Constructing Slotted Registered ABE

[GLWW24]

General components: $Z = e(g,g)^{\alpha}$ $h \leftarrow \mathbb{G}$

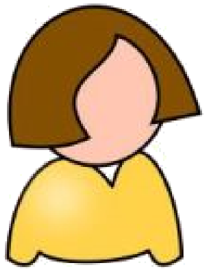
Slot components: $A_i = g^{t_i},\ B_i = g^{\alpha} h^{t_i}$

Attribute component: $U_1 = g^{u_1},\ U_2 = g^{u_2}$

Aggregated master public key

$\hat{T} = g^{r_1 + r_2}$

$\hat{U}_1 = g^{u_2},\ \hat{U}_2 = g^{u_1}$



$\mathrm{pk}_1 = g^{r_1}$
$S_1 = \{1\}$

General components: $\mu \cdot Z^s,\ g^s$

Slot component: $h_1^s \hat{T}^s$

Attribute component: $h_2^s\ \hat{U}_1^s$

Goal: recover μ

Step 1: Compute $e(g^s, B_1) = e(g,g)^{\alpha s} e(g,h)^{s t_i} = Z^s \cdot e(g,h)^{s t_1}$

Step 2 (Slot Check): Using cross-terms and secret key r_1, compute $e(g, h_1)^{s t_1}$

Constructing Slotted Registered ABE

[GLWW24]

General components: $Z = e(g,g)^{\alpha}$ $\quad h \leftarrow \mathbb{G}$

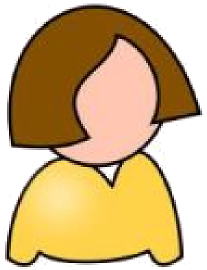
Slot components: $A_i = g^{t_i},\ B_i = g^{\alpha} h^{t_i}$

Attribute component: $U_1 = g^{u_1},\ U_2 = g^{u_2}$

Aggregated master public key

$\hat{T} = g^{r_1 + r_2}$

$\hat{U}_1 = g^{u_2},\ \hat{U}_2 = g^{u_1}$



$\mathrm{pk}_1 = g^{r_1}$
$S_1 = \{1\}$

General components: $\mu \cdot Z^s,\ g^s$

Slot component: $h_1^s \hat{T}^s$

Attribute component: $h_2^s\, \hat{U}_1^s$

Goal: recover μ

Step 1: Compute $e(g^s, B_1) = e(g,g)^{\alpha s} e(g,h)^{s t_i} = Z^s \cdot e(g,h)^{s t_1}$

Step 2 (Slot Check): Using cross-terms and secret ke

Share of $e(g,h)^{s t_1}$

Cross-term between slot and attribute components (available only if user has attribute)

Step 3 (Policy Check): Compute $e\left(A_1, h_2^s \hat{U}_1^s\right) = e\left(g^{t_1}, h_2^s \hat{U}_1^s\right) = e(g, h_2)^{s t_1} e(g,g)^{s t_1 u_2}$

Constructing Slotted Registered ABE

[GLWW24]

General components: $Z = e(g,g)^{\alpha}$ $h \leftarrow \mathbb{G}$

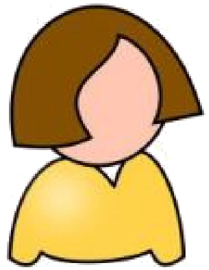
Slot components: $A_i = g^{t_i},\ B_i = g^{\alpha} h^{t_i}$

Attribute component: $U_1 = g^{u_1},\ U_2 = g^{u_2}$

Aggregated master public key

$\hat{T} = g^{r_1 + r_2}$

$\hat{U}_1 = g^{u_2},\ \hat{U}_2 = g^{u_1}$



$\mathrm{pk}_1 = g^{r_1}$
$S_1 = \{1\}$

General components: $\mu \cdot Z^s, g^s$

Slot component: $h_1^s \hat{T}^s$

Attribute component: $h_2^s \hat{U}_1^s$

Step 1: Compute $e(g^s, B_1) = e(g,g)^{\alpha s} e(g,h)^{st_i} = Z^s \cdot e(g,h)^{st_1}$

Step 2 (Slot Check): Using cross-terms and secret key r_1, compute $e(g, h_1)^{st_1}$

Step 3 (Policy Check): Using cross-terms, compute $e(g, h_2)^{st_1}$

Constructing Slotted Registered ABE

[GLWW24]

General components: $Z = e(g,g)^{\alpha}$ $\quad h \leftarrow \mathbb{G}$

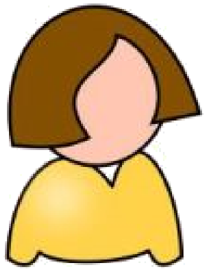
Slot components: $A_i = g^{t_i},\ B_i = g^{\alpha} h^{t_i}$

Attribute component: $U_1 = g^{u_1},\ U_2 = g^{u_2}$

Aggregated master public key

$$\hat{T} = g^{r_1 + r_2}$$

$$\hat{U}_1 = g^{u_2},\ \hat{U}_2 = g^{u_1}$$



$\mathrm{pk}_1 = g^{r_1}$
$S_1 = \{1\}$

General components: $\mu \cdot Z^s,\ g^s$

Slot component: $h_1^s \hat{T}^s$

Attribute component: $h_2^s\ \hat{U}_1^s$

Summary of approach:

- Aggregated key is the product of each user's individual public key (one per slot)
- Decryption will produce cross terms between slot i and user j's secret key
- Each user includes a cross-term to cancel out these effects (part of the user's helper decryption key); CRS will contain cross-terms for attribute-slot components

Constructing Slotted Registered ABE

[GLWW24]

General components: $Z = e(g, g)^{\alpha}$ $h \leftarrow \mathbb{G}$

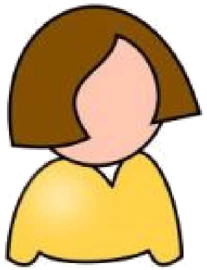
Aggregated master public key

Slot components: $A_i = g^{t_i},\ B_i = g^{\alpha} h^{t_i}$

$\hat{T} = g^{r_1 + r_2}$

Attribute component: $U_1 = g^{u_1},\ U_2 = g^{u_2}$

$\hat{U}_1 = g^{u_2},\ \hat{U}_2 = g^{u_1}$



$\mathrm{pk}_1 = g^{r_1}$
$S_1 = \{1\}$

General components: $\mu \cdot Z^s,\ g^s$

Slot component: $h_1^s \hat{T}^s$

Attribute component: $h_2^s\ \hat{U}_1^s$

To decrypt a ciphertext, **two** properties should hold:

- User should have the secret key for slot i *Enforced by the slot components*
- Attributes associated with slot i should satisfy the challenge policy

Enforced by the attribute components

Constructing Slotted Registered ABE

[GLWW24]

General components: $Z = e(g,g)^\alpha \quad h \leftarrow \mathbb{G}$

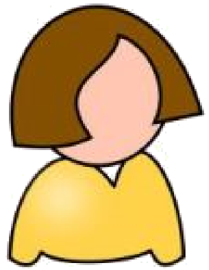
Slot components: $A_i = g^{t_i},\ B_i = g^\alpha h^{t_i}$

Attribute component: $U_1 = g^{u_1},\ U_2 = g^{u_2}$

Aggregated master public key

$\hat{T} = g^{r_1+r_2}$

$\widehat{U}_1 = g^{u_2},\ \widehat{U}_2 = g^{u_1}$



$\mathrm{pk}_1 = g^{r_1}$
$S_1 = \{1\}$

General components: $\mu \cdot Z^s, g^s$

Slot component: $h_1^s \hat{T}^s$

Attribute component: $h_2^s\, \widehat{U}_1^s$

Key technical approach: cancelling out cross-terms

- Technique leveraged in many pairing-based constructions of registration-based primitives
- **Recently:** lattice-based instantiation (in the setting of broadcast encryption) [CW24]
- But… seems to require a *long* and *structured* common reference string

Constructing Slotted Registered ABE

[GLWW24]

General components: $Z = e(g,g)^{\alpha}$ $\quad h \leftarrow \mathbb{G}$

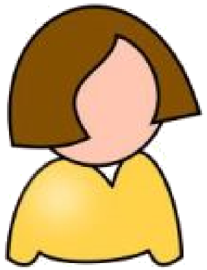
Aggregated master public key

Slot components: $A_i = g^{t_i},\ B_i = g^{\alpha} h^{t_i}$

$\hat{T} = g^{r_1 + r_2}$

Attribute component: $U_1 = g^{u_1},\ U_2 = g^{u_2}$

$\hat{U}_1 = g^{u_2},\ \hat{U}_2 = g^{u_1}$



$\mathrm{pk}_1 = g^{r_1}$
$S_1 = \{1\}$

General components: $\mu \cdot Z^s,\ g^s$

Slot component: $h_1^s \hat{T}^s$

Attribute component: $h_2^s\ \hat{U}_1^s$

Key technical app

- Technique leve
- **Recently:** lattic
- But… seems to

Replace attribute components with **linear secret sharing** of s to support policies with a linear secret sharing scheme

Reducing the CRS Size

[GLWW24]

As described, size of CRS is **quadratic** in number of slots

Reason: Each slot is associated with a slot exponent t_i and an attribute exponent u_i

Policy checking mechanism produces **extraneous** terms of the form $g^{s t_i u_j}$ for $i \neq j$ and where g^s is from the challenge ciphertext

CRS will need to contain $g^{t_i u_j}$ for each $i \neq j$ for correctness

Can we publish fewer cross terms and still have correctness?

Approach: Choose t_i, u_i to be structured so there is redundancy in cross terms

Reducing the CRS Size

[GLWW24]

Given $g^{t_1}, \ldots, g^{t_L}$ and $g^{u_1}, \ldots, g^{u_L}$

Goal: give out $g^{t_i u_j}$ for all $i \neq j$, but without ability to compute $g^{t_i u_i}$

Set $t_i = \alpha^{d_i}$ for some $\alpha \leftarrow \mathbb{Z}_p$

Set $u_i = \beta \cdot \alpha^{d_i}$ where $\beta \leftarrow \mathbb{Z}_p$

$\Rightarrow$ $t_i u_j = \alpha^{d_i} \cdot \beta \alpha^{d_j} = \beta \alpha^{d_i + d_j}$

for *some* choice of $d_1, \ldots, d_L \in \mathbb{N}$

Observe: if many pairs i, j share a common value $d_i + d_j$, then all such pairs can share a single cross term $g^{\beta \alpha^{d_i + d_j}}$

Reducing the CRS Size

[GLWW24]

Observe: if many pairs i, j share a common value $d_i + d_j$, then all such pairs can share a single cross term $g^{\beta\alpha^{d_i+d_j}}$

How to choose $d_1, \dots, d_L$?

Requirement: For all k, there should not exist $i \neq j$ where $d_i + d_j = d_k + d_k$

Cross-term for (i, j) *must* ***not*** *collide with non-cross-term for* k

If $d_i + d_j = 2d_k$ (with $d_i < d_j$), then (d_i, d_k, d_j) form an **arithmetic progression**

Suffices to come up with a *progression-free* set of integers $\mathcal{D} \subset \mathbb{N}$ of size L and set $\{d_1, \dots, d_L\} = \mathcal{D}$; number of cross terms is then at most $2 \max \mathcal{D}$

Reducing the CRS Size

[GLWW24]

Observe: if many pairs i, j share a common value $d_i + d_j$, then all such pairs can share a single cross term $g^{\beta\alpha^{d_i+d_j}}$

How to choose $d_1, \dots, d_L$?

Previously used to reduce the CRS size in the context of pairing-based SNARKs [Lip12]

j where $d_i + d_j = d_k + d_k$

non-cross-term for k

(d_i, d_k, d_j) form an **arithmetic progression**

Suffices to come up with a *progression-free* set of integers $\mathcal{D} \subset \mathbb{N}$ of size L and set $\{d_1, \dots, d_L\} = \mathcal{D}$; number of cross terms is then at most $2 \max \mathcal{D}$

Progression-Free Sets

[GLWW24]

Simple construction due to Erdös and Turán [ET36]

Let $\mathcal{D} \subset \mathbb{N}$ be the numbers whose ternary representation only use the digits 0 and 1

1	=	001
3	=	010
4	=	011
9	=	100
10	=	101
12	=	110
13	=	111

Progression-free:

$2d_k$ is a number that only uses 0 and 2 in ternary

If $d_i \neq d_j$, then $d_i + d_j$ must contain a 1 somewhere in ternary

Thus $d_i + d_j \neq 2d_k$ for all $i \neq j$

To get a progression-free set with L values, maximum entry has size $L^{\log_2 3}$

Implies registered ABE scheme with CRS of size $O\left(L^{\log_2 3}\right)$

State-of-the-art [Beh46, Elk10]**:** For every $L \in \mathbb{N}$, there exists a progression-free set of L integers with maximum value bounded by $L^{1+o(1)} \Longrightarrow$ registered ABE with CRS size $L^{1+o(1)}$

Progression-Free Sets

[GLWW24]

Simple construction due to Erdös and Turán [ET36]

Let $\mathcal{D} \subset \mathbb{N}$ be the numbers whose ternary representation only use the digits 0 and 1

1	=	001
3	=	010
4	=	011
9	=	100
10	=	101
12	=	110
13	=	111

Progression-free:

$2d_k$ is a number that only uses 0 and 2 in ternary

If $d_i \neq d_j$, then $d_i + d_j$ must contain a 1 somewhere in ternary

Thus $d_i + d_j \neq 2d_k$ for all $i \neq j$

To get a progression … $\log_2 3$

Implies registered A…

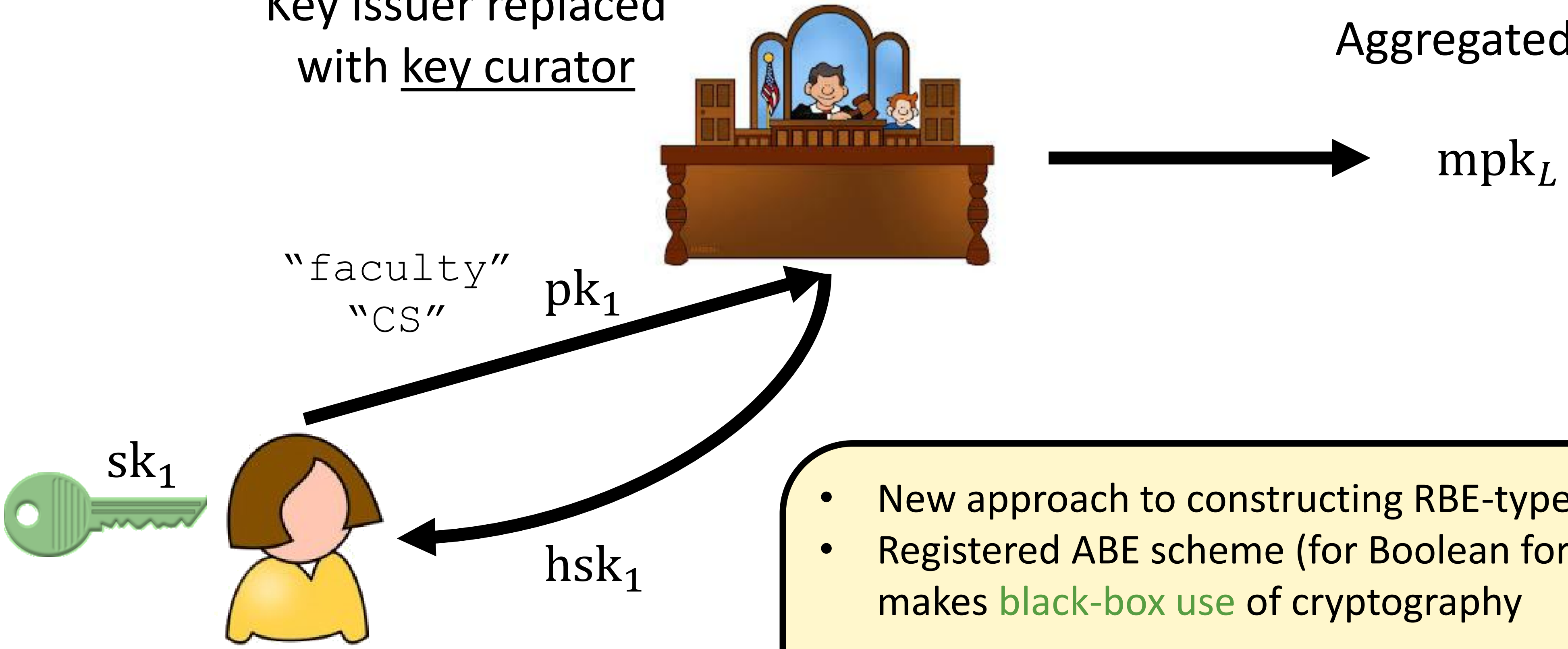
Achieves **nearly linear CRS**, but this approach **cannot** get to linear-size CRS

State-of-the-art [Beh46, Elk10]**:** For every $L \in \mathbb{N}$, there exists a progression-free set of L integers with maximum value bounded by $L^{1+o(1)} \Longrightarrow$ registered ABE with CRS size $L^{1+o(1)}$

Registered ABE Summary

Key issuer replaced with key curator

Aggregated key



Users chooses their own public/secret key

- New approach to constructing RBE-type of primitives
- Registered ABE scheme (for Boolean formulas) only makes black-box use of cryptography

- Construction will need a (trusted) common reference string (CRS) and supports bounded number of users

Lots to Explore for Registered ABE!

Pairing-based constructions require a **long** and **structured** CRS

- [HLWW23, ZZGQ23]: quadratic-size CRS
- [GLWW24]: nearly-linear size CRS ($L^{1+o(1)}$) using progression-free sets

Pairing-based constructions with linear-size CRS? Sublinear-size CRS? Transparent CRS?

- Possible using indistinguishability obfuscation [HLWW23] or witness encryption [FWW23]

Lower bounds on CRS size for constructions that make black-box use of cryptography?

Registered ABE from LWE (or falsifiable lattice assumptions)?

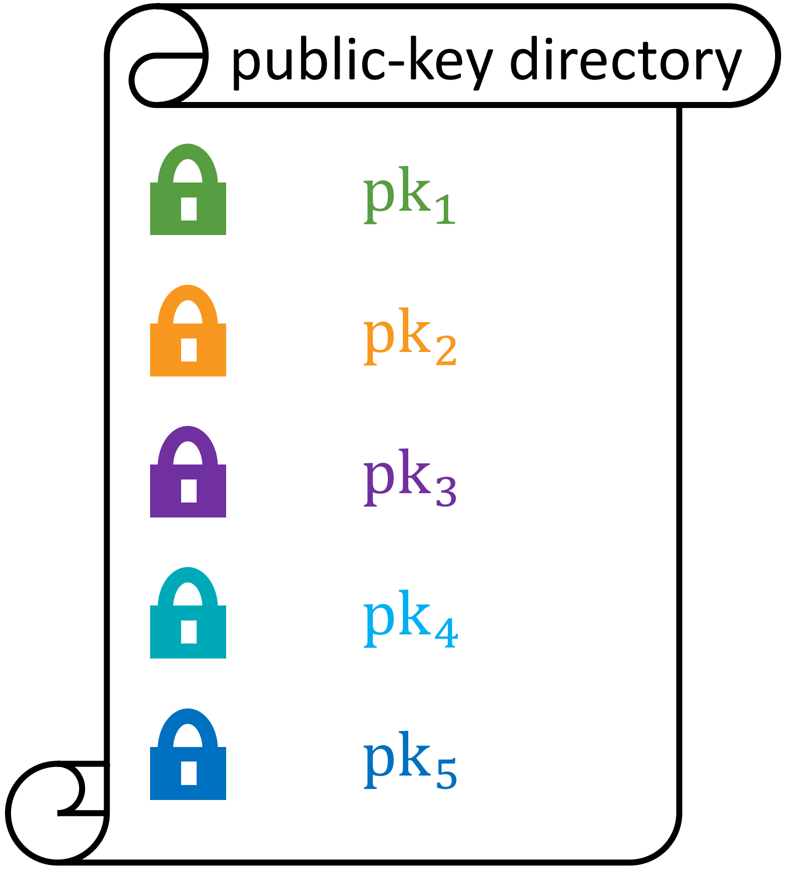
Registered ABE for Boolean circuits?

- Known from indistinguishability obfuscation or witness encryption
- [ZZGQ23]: registered ABE for arithmetic branching programs and inner products

An Application to Broadcast Encryption

[FWW23]

Registered ABE is a useful building block for other **trustless** cryptographic systems



Independent, user-generated keys

Suppose we want to encrypt a message to $\{\mathrm{pk}_1, \mathrm{pk}_3, \mathrm{pk}_4\}$

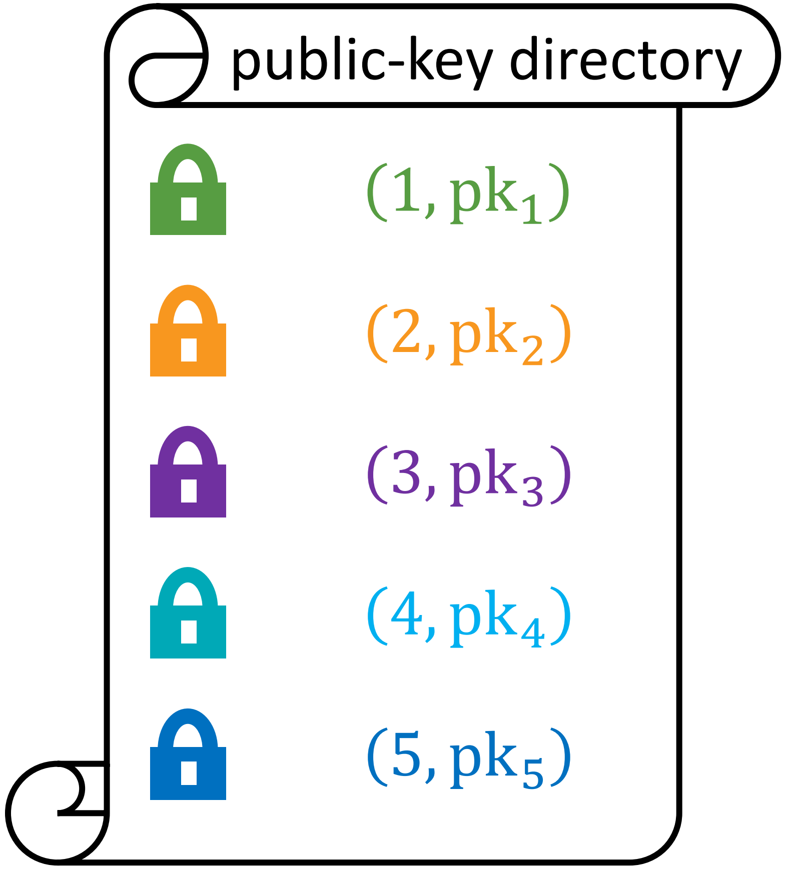
Public-key encryption: ciphertext size grows with the size of the set

Broadcast encryption: achieve *sublinear* ciphertext size, but requires central authority

An Application to Broadcast Encryption

[FWW23]

Distributed broadcast encryption [BZ14]



Each user chooses its own public key, and each key has a **unique** index

$$\mathrm{Encrypt}(\mathrm{pp}, \{\mathrm{pk}_i\}_{i \in S}, m) \to \mathrm{ct}$$

Can encrypt a message m to any set of public keys

Efficiency: $|\mathrm{ct}| = |m| + \mathrm{poly}(\lambda, \log|S|)$

$$\mathrm{Decrypt}(\mathrm{pp}, \{\mathrm{pk}_i\}_{i \in S}, \mathrm{sk}, \mathrm{ct}) \to m$$

Any secret key associated with broadcast set can decrypt

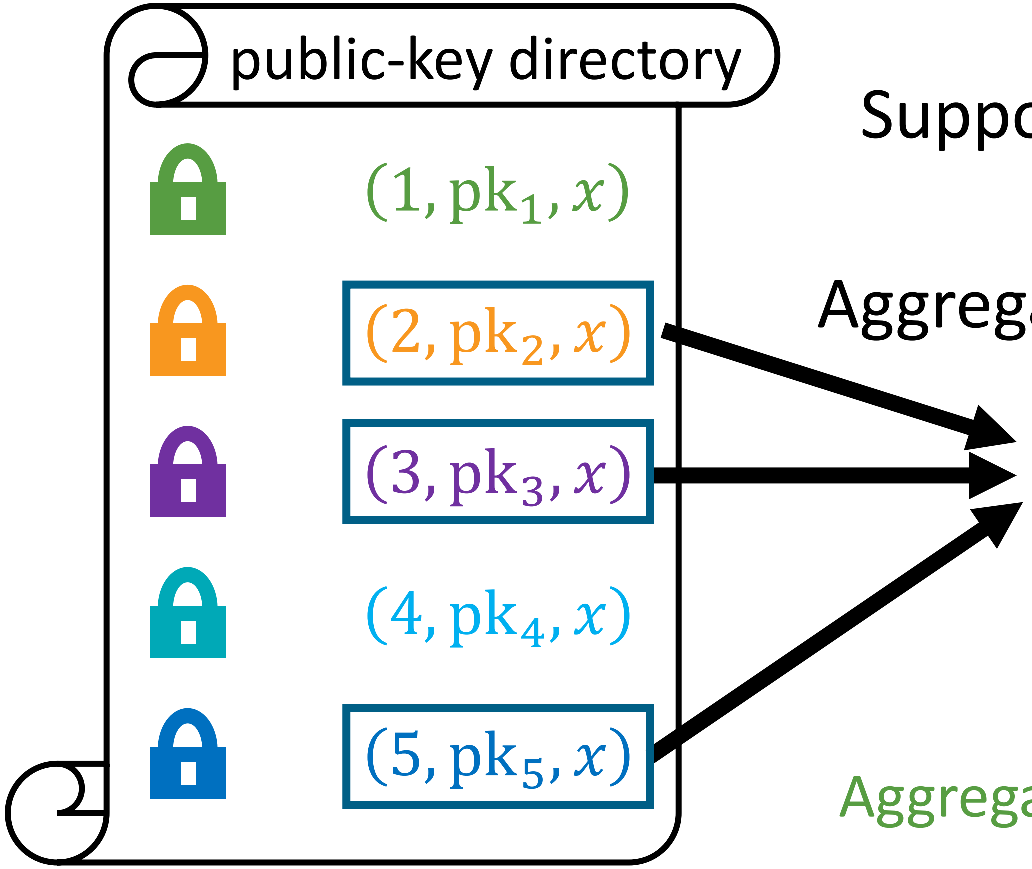
Decryption does requires knowledge of public keys in broadcast set

Distributed Broadcast from Slotted Registered ABE

[FWW23]

Consider a registered ABE scheme with a single dummy attribute x

Public key for an index i is a key for **slot i** with **attribute** x



Suppose we want to encrypt to a set $S = \{2,3,5\}$

Aggregate

$\mathrm{mpk} \longrightarrow \mathrm{Encrypt}(\mathrm{mpk}, x, P)$

Encrypt with respect to mpk to policy P that accepts x

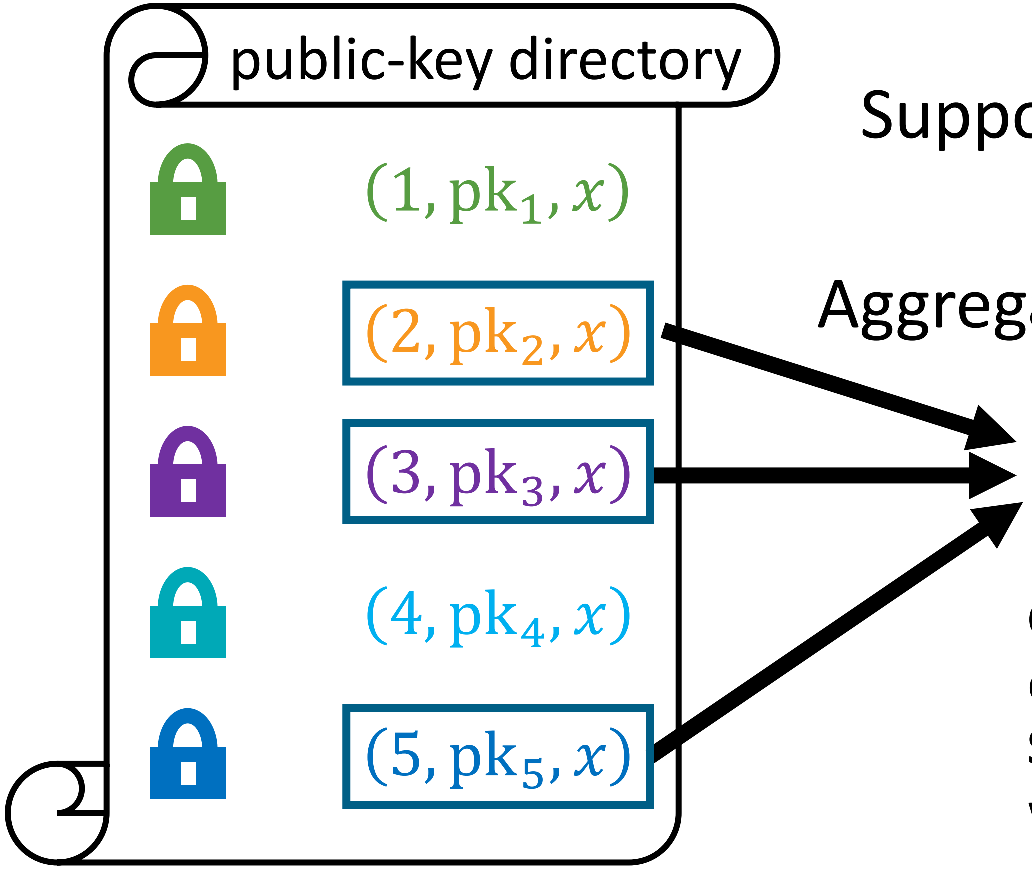
Aggregate public keys using slotted registered ABE scheme

Distributed Broadcast from Slotted Registered ABE

[FWW23]

Consider a registered ABE scheme with a single dummy attribute x

Public key for an index i is a key for **slot i** with **attribute** x



Suppose we want to encrypt to a set $S = \{2,3,5\}$

Aggregate

Encrypt with respect to mpk to policy P that accepts x

$\mathrm{mpk} \longrightarrow \mathrm{Encrypt}(\mathrm{mpk}, x, P)$

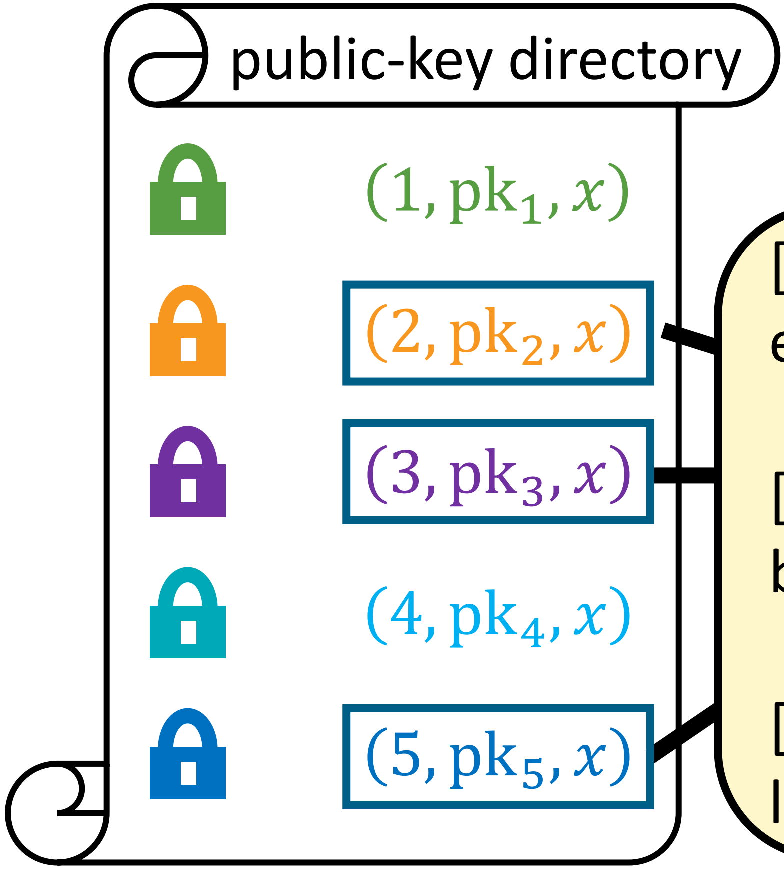
Correctness: If $i \in S$, then (i, pk_i, x) was aggregated in mpk so decryption is possible using sk_i

Security: If $i \notin S$, then (i, pk_i, x) was **not** aggregated in mpk so we can appeal to security of registered ABE

Distributed Broadcast from Slotted Registered ABE

Consider a registered ABE scheme with a single dummy attribute x

Public key for an index i is a key for **slot i** with **attribute** x



Suppose we want to encrypt to a set $S = \{2,3,5\}$

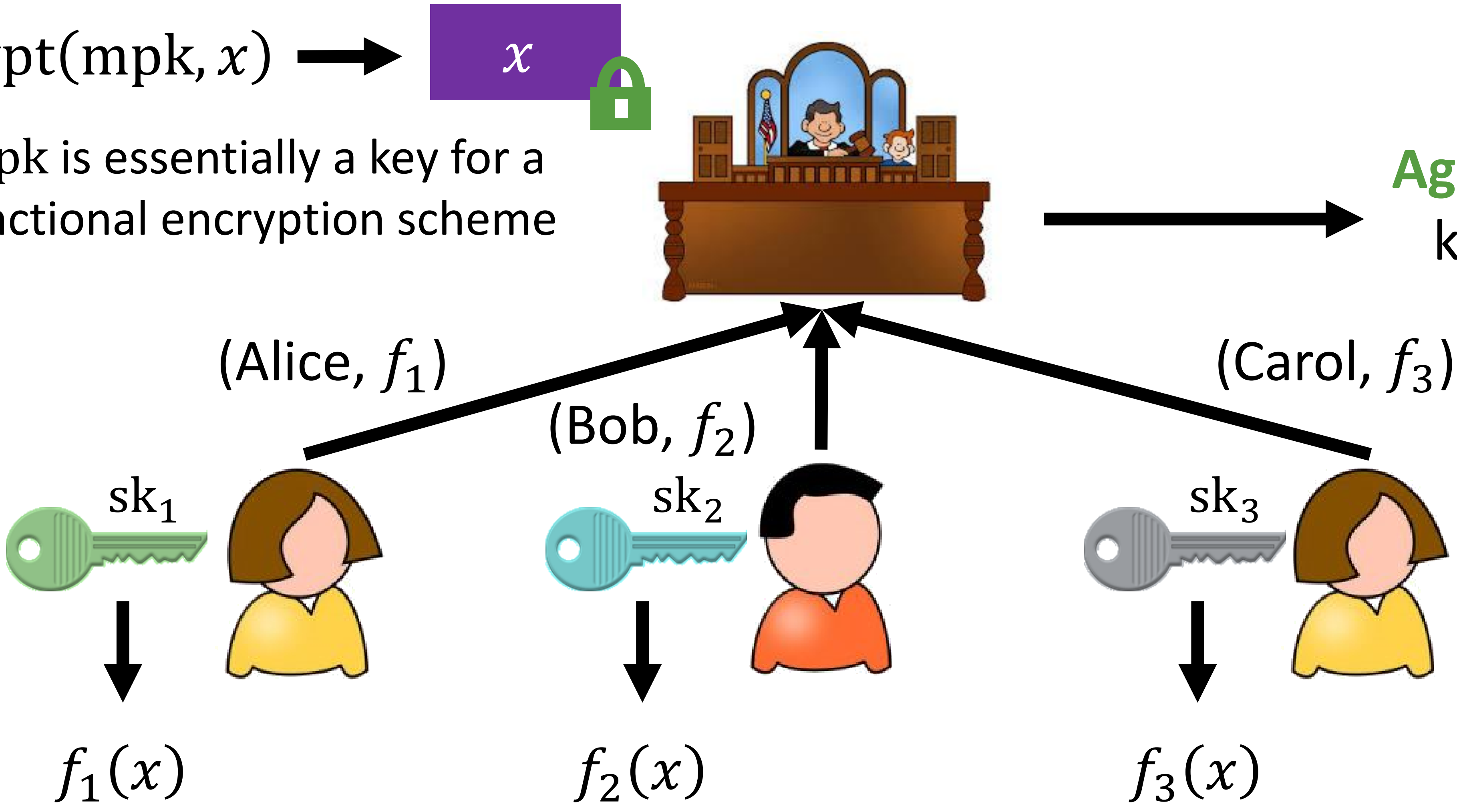
[FWW23]: Registered ABE + compiler ⇒ distributed broadcast encryption from pairings

[KMW23, GKPW24]: direct constructions of distributed broadcast encryption (and more) from pairings

[CW24]: distributed broadcast encryption from falsifiable lattice assumptions (ℓ-succinct LWE)

Removing Trust from Functional Encryption

$\mathrm{Encrypt}(\mathrm{mpk}, x)$ →



mpk is essentially a key for a functional encryption scheme

Aggregate public keys together

Goal: Support capabilities of functional encryption **without** a trusted authority

Open Problems

Schemes with **short** CRS or **unstructured** CRS without non-black-box use of cryptography

Existing constructions have long structured CRS (typically quadratic in the number of users)

Lattice-based constructions of registration-based primitives

Registration-based encryption known from LWE [DKLLMR23]

Registered ABE for circuits known from evasive LWE (via witness encryption) [FWW23]

Distributed broadcast encryption from ℓ-succinct LWE [CW24]

Key revocation and verifiability

Defending against possibly malicious adversaries

Improve concrete efficiency for registration-based primitives

Current bottlenecks include large CRS and large public keys

Thank you!

References

[BLMMRW24] Pedro Branco, Russell W. F. Lai, Monosij Maitra, Giulio Malavolta, Ahmadreza Rahimi, and Ivy K. Y. Woo. Traitor Tracing without Trusted Authority from Registered Functional Encryption. 2024.

[BSW11] Dan Boneh, Amit Sahai, and Brent Waters. Functional Encryption: Definitions and Challenges. TCC 2011.

[BZ14] Dan Boneh and Mark Zhandry. Multiparty Key Exchange, Efficient Traitor Tracing, and More from Indistinguishability Obfuscation. CRYPTO 2014.

[CES21] Kelong Cong, Karim Eldefrawy, and Nigel P. Smart. Optimizing Registration Based Encryption. IMACC 2021.

[CW24] Jeffrey Champion and David J. Wu. Distributed Broadcast Encryption from Lattices. 2024.

[DKLLMR23] Nico Döttling, Dimitris Kolonelos, Russell W. F. Lai, Chuanwei Lin, Giulio Malavolta, and Ahmadreza Rahimi. Efficient Laconic Cryptography from Learning with Errors. EUROCRYPT 2023.

[DPY23] Pratish Datta, Tapas Pal, and Shota Yamada. Registered FE Beyond Predicates: (Attribute-Based) Linear Functions and More. 2023.

[FFMMRV23] Danilo Francati, Daniele Friolo, Monosij Maitra, Giulio Malavolta, Ahmadreza Rahimi, and Daniele Venturi. Registered (Inner-Product) Functional Encryption. ASIACRYPT 2023.

[FKP23] Dario Fiore, Dimitris Kolonelos, and Paola de Perthuis. Cuckoo Commitments: Registration-Based Encryption and Key-Value Map Commitments for Large Spaces. ASIACRYPT 2023.

References

[FWW23] Cody Freitag, Brent Waters, and David J. Wu. How to Use (Plain) Witness Encryption: Registered ABE, Flexible Broadcast, and More. CRYPTO 2023.

[GHMMRS19] Sanjam Garg, Mohammad Hajiabadi, Mohammad Mahmoody, Ahmadreza Rahimi, and Sruthi Sekar. Registration-Based Encryption from Standard Assumptions. PKC 2019.

[GHMR18] Sanjam Garg, Mohammad Hajiabadi, Mohammad Mahmoody, and Ahmadreza Rahimi. Registration-Based Encryption: Removing Private-Key Generator from IBE. TCC 2018.

[GKMR23] Noemi Glaeser, Dimitris Kolonelos, Giulio Malavolta, and Ahmadreza Rahimi. Efficient Registration-Based Encryption. ACM CCS 2023.

[GKPW24] Sanjam Garg, Dimitris Kolonelos, Guru-Vamsi Policharla, and Mingyuan Wang. Threshold Encryption with Silent Setup. CRYPTO 2024.

[GLWW23] Rachit Garg, George Lu, Brent Waters, and David J. Wu. Realizing Flexible Broadcast Encryption: How to Broadcast to a Public-Key Directory. ACM CCS 2023.

[GLWW24] Rachit Garg, George Lu, Brent Waters, and David J. Wu. Reducing the CRS Size in Registered ABE Systems. CRYPTO 2024.

[GPSW06] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-Based Encryption for Fine-Grained Access Control of Encrypted Data. ACM CCS 2006

References

[GV20] Rishab Goyal and Satyanarayana Vusirikala. Verifiable Registration-Based Encryption. CRYPTO 2020.

[HLWW23] Susan Hohenberger, George Lu, Brent Waters, and David J. Wu. Registered Attribute-Based Encryption. EUROCRYPT 2023.

[KMW23] Dimitris Kolonelos, Giulio Malavolta, and Hoeteck Wee. Distributed Broadcast Encryption from Bilinear Groups. ASIACRYPT 2023.

[O'N10] Adam O'Neill. Definitional Issues in Functional Encryption. 2010.

[SS10] Amit Sahai and Hakan Seyalioglu. Worry-Free Encryption: Functional Encryption with Public Keys. ACM CCS 2010.

[SW05] Amit Sahai and Brent Waters. Fuzzy Identity-Based Encryption. EUROCRYPT 2005.

[ZLZGQ24] Ziqi Zhu, Jiangtao Li, Kai Zhang, Junqing Gong, and Haifeng Qian. Registered Functional Encryptions from Pairings. EUROCRYPT 2024.

[ZZGQ23] Ziqi Zhu, Kai Zhang, Junqing Gong, and Haifeng Qian. Registered ABE via Predicate Encodings. ASIACRYPT 2023.