Removing Trust Assumptions from Advanced Encryption Schemes

David Wu
Functional Encryption (FE)

[SS10, O’N10, BSW11]

ciphertext encrypting $x$

master secret key

$f_1$ learns $f_1(x)$

$f_2$ learns $f_2(x)$

$f_3$ learns $f_3(x)$
Functional Encryption (FE)

$ciphertext$ encrypting $x$

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

- Should not learn more than $f_1(x)$ and $f_2(x)$

master secret key

[SS10, O’N10, BSW11]
Functional Encryption (FE)

If the key-issuer is compromised?

ciphertext encrypting $x$

$f_1$ learns $f_1(x)$

$f_2$ learns $f_2(x)$

$f_3$ learns $f_3(x)$
Functional Encryption (FE)

- Key issuer can decrypt all ciphertexts
- Central point of failure
- Users do not have control over keys

What if the key-issuer is compromised?

- $f_1$ learns $f_1(x)$
- $f_2$ learns $f_2(x)$
- $f_3$ learns $f_3(x)$
**Functional Encryption vs. Public-Key Encryption**

### Public-key encryption is **decentralized**

- 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

Can we get the best of both worlds?
Registration-Based Encryption (RBE)

Key issuer replaced with key curator

Special case of identity-based encryption (IBE)

Decryption keys are associated with identities

Users choose their own public/secret key and register their public key with the curator

[Alice, \( pk_1 \)]

\( sk_1 \)

\( (Alice, pk_1) \)

\( (Bob, pk_2) \)

\( sk_2 \)

\( (Carol, pk_3) \)

\( sk_3 \)
Registration-Based Encryption (RBE)

Key issuer replaced with key curator

Key curator is deterministic and transparent (no secrets)

Aggregated public keys together

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

Users choose their own public/secret key and register their public key with the curator

sk_1

sk_2

sk_3

(Bob, pk_2)
Registration-Based Encryption (RBE)

Key issuer replaced with key curator

(Alice, pk₁)

(Bob, pk₂)

(Carol, pk₃)

Aggregate public keys together

mpk

Encrypt(mpk, Carol, message)

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

id: Carol

[GHMR18]
Registration-Based Encryption (RBE)

Key issuer replaced with key curator

Aggregate public keys together

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

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

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

#### Key Updates per User

$$\# \text{ key updates per user} = \text{poly}(\lambda, \log L)$$
Registration-Based Encryption (RBE) [GHMR18]

- Key issuer replaced with key curator
- 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

Key issuer replaced with key curator

Aggregate public keys together

\[
|\text{mpk}| = \text{poly}(\lambda, \log L)
\]

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

Encrypt(mpk, x) → x

mpk is essentially a key for a functional encryption scheme

Aggregate public keys together

mpk

|mpk| = poly(\lambda, \log L)

Encrypt(mpk, x) → x

mpk is essentially a key for a functional encryption scheme

Aggregate public keys together

mpk

|mpk| = poly(\lambda, \log L)

Encrypt(mpk, x) → x

mpk is essentially a key for a functional encryption scheme

Aggregate public keys together

mpk

|mpk| = poly(\lambda, \log L)
# 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, GKM23, ZZGQ23, FKP23]

<table>
<thead>
<tr>
<th>Registered attribute-based encryption (ABE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Monotone Boolean formulas [HLW23, ZZGQ23, GLW24]</td>
</tr>
<tr>
<td>• Inner products [FFMMRV23, ZZGQ23]</td>
</tr>
<tr>
<td>• Arithmetic branching program [ZZGQ23]</td>
</tr>
<tr>
<td>• Boolean circuits [HLW23, FW23]</td>
</tr>
</tbody>
</table>

This talk: Lots of progress in this past year!

Distributed/flexible broadcast [BZ14, KMW23, FW23, GLW23, GKP24, CW24]

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

message

policy: CS and faculty

master secret key

“faculty” “CS”

“faculty” “math”

“student” “CS”
Attribute-Based Encryption

policy: CS and faculty

message

master secret key

“faculty”
“CS”

Can decrypt

“faculty”
“math”

“student”
“CS”

[SW05, GPSW06]
Attribute-Based Encryption

**Policy:** CS and faculty

- **faculty**
  - Can decrypt
- **CS**
  - Cannot decrypt
- **faculty**
  - Cannot decrypt
- **math**
  - Cannot decrypt
- **student**
  - Cannot decrypt

**Message:**

[SW05, GSW06]
Attribute-Based Encryption

message

policy: CS and faculty

Users cannot collude to decrypt

“faculty”
“CS”

“faculty”
“math”

“student”
“CS”

master secret key

[SW05, GSW06]
Registered Attribute-Based Encryption

Users choose their own public/secret key.

message

ciphertexts associated with policy

transparent key curator

aggregated public key

mpk

"faculty" "CS"

pk₁

"student" "CS"

pk₃

"faculty" "math"

pk₂

"student"

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

Registered Attribute-Based Encryption

[HLWW23]
Registered Attribute-Based Encryption

Users choose their own public/secret key

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

message

policy: CS and faculty

ciphertexts associated with policy

transparent key curator

aggregated public key

users

sk_1

pk_1

sk_2

pk_2

sk_3

pk_3

“student”

“CS”

“faculty”

“math”

“faculty”

[policy: CS and faculty]

[HLW23]
**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

$$\begin{align*}
\text{pk}_1, S_1 & \quad \text{pk}_2, S_2 & \quad \text{pk}_3, S_3 & \quad \text{pk}_4, S_4 & \quad \cdots & \quad \text{pk}_L, S_L
\end{align*}$$

Each slot associated with a public key $\text{pk}$ and a set of attributes $S$

$$\begin{align*}
|\text{mpk}| & = \text{poly}(\lambda, |\mathcal{U}|, \log L) & \lambda: \text{security parameter} \\
|hsk_i| & = \text{poly}(\lambda, |\mathcal{U}|, \log L) & \mathcal{U}: \text{universe of attributes}
\end{align*}$$
A Template for Building Registered ABE

**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

\[
\begin{array}{c|c|c|c|c|c}
pk_1, S_1 & pk_2, S_2 & pk_3, S_3 & pk_4, S_4 & \ldots & pk_L, S_L \\
\end{array}
\]

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

Encrypt($mpk, P, m$) $\rightarrow$ ct

Decrypt($sk_i, hsk_i, ct$) $\rightarrow$ $m$

Encryption takes master public key and policy $P$ (no slot)

Decryption takes secret key $sk_i$ for some slot and the helper key $hsk_i$ for that slot
**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.

![Diagram showing slots with public keys and attributes](image)

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

- **Encrypt**($mpk, P, m$) → $ct$
- **Decrypt**($sk_i, hsk_i, ct$) → $m$

Main difference with registered ABE: Aggregate takes all $L$ keys simultaneously.
Let $L$ be the number of users

$$
\begin{align*}
pk_1, S_1 & \quad pk_2, S_2 & \quad pk_3, S_3 & \quad pk_4, S_4 & \quad \cdots & \quad pk_L, S_L
\end{align*}
$$

Aggregate

$$
\text{mpk} \quad \text{hsk}_1, \ldots, \text{hsk}_L
$$

Slotted scheme does \textit{not} support online registration

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

Maintain $\log L$ slotted schemes, where scheme $i$ supports $2^i$ users
Constructing Slotted Registered ABE

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

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

\[
e(g^x, g^y) = e(g, g)^{xy}
\]

\textit{Multiplies exponents in the target group}
Constructing Slotted Registered ABE

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

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

\[
(A_1, B_1) \quad (A_2, B_2)
\]

\[
A_i = g^{t_i} \quad B_i = g^\alpha h^{t_i}
\]

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

\[
U_1 \quad U_2
\]

$t_i$ is a slot exponent $u_i$ is an attribute exponent
Constructing Slotted Registered ABE

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

Slot components: \((A_1, B_1)\) and \((A_2, B_2)\)

\[ A_i = g^{t_i} \quad B_i = g^\alpha h^{t_i} \]

Attribute component: \(U_1, U_2\)

\[ U_i = g^{u_i} \]

To decrypt a ciphertext, two properties should hold:

- User should have the secret key for slot \(i\)
  \(\text{Enforced by the slot components}\)
- Attributes associated with slot \(i\) should satisfy the challenge policy
  \(\text{Enforced by the attribute components}\)
Constructing Slotted Registered ABE

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

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

Attribute component: \(U_1, U_2\) \( U_i = g^{u_i} \)

User’s individual public/secret key is an ElGamal key-pair
\( sk = r, \ pk = g^r \) (and some auxiliary information)

Aggregating public keys \((pk_1, pk_2)\) with attribute sets \(S_1, S_2\)

\[ \hat{T} = pk_1 \cdot pk_2 = g^{r_1+r_2} \]

Aggregated public key: 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

pk\( _1 = g^{r_1}\) \( S_1 = \{1\} \)

pk\( _2 = g^{r_2}\) \( S_2 = \{2\} \)
Constructing Slotted Registered ABE

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

\[ \hat{T} = g^{r_1+r_2} \]
\[ \hat{U}_1 = g^{u_2}, \hat{U}_2 = g^{u_1} \]

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

Suppose we encrypt \( \mu \) to the policy “has attribute 1”

General components: \( \mu \cdot Z^s, g^s \)

Slot component: \( h_1^s \hat{T}^s \)

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

pk_1 = g^{r_1}
\( S_1 = \{1\} \)

pk_1 = g^{r_2}
\( S_1 = \{2\} \)
Constructing Slotted Registered ABE

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

\[ \hat{T} = g^{r_1+r_2} \]
\[ \hat{U}_1 = g^{u_2}, \; \hat{U}_2 = g^{u_1} \]

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 \( \mu \)

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

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

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
\( \hat{T} = g^{r_1+r_2} \)
\( \hat{U}_1 = g^{u_2}, \hat{U}_2 = g^{u_1} \)

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 \( \mu \)

Can compute using secret key \( r_1 \)

Can compute using secret key \( r_1 \)

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

Cross term from Party 2

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

Step 2 (Slot Check): Compute \( e(A_1, h_1^s \hat{T}^s) = e(g^{t_1}, h_1^s \hat{T}^s) = e(g, h_1)^{st_1} e(g, g)^{sr_{1t_1}} e(g, g)^{sr_{2t_1}} \)

Given cross-term \( e(g,g)^{r_2t_1} \), can recover \( e(g, h_1)^{st_1} \)
Constructing Slotted Registered ABE

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

**Slot components:** \( A_i = g^{t_i}, B_i = g^α h^{t_i} \)

**Attribute component:** \( U_1 = g^{u_1}, U_2 = g^{u_2} \)

**Aggregated master public key**

\[ \hat{T} = g^{r_1+r_2} \]
\[ \hat{U}_1 = g^{u_2}, \hat{U}_2 = g^{u_1} \]

\( pk_1 = g^{r_1} \)
\( S_1 = \{1\} \)

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

Given cross-term \( e(g, g)^{r_2t_1} \), can recover \( e(g, h_1)^{st_1} \)

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

Can compute using secret key \( r_1 \)

Cross term from Party 2

\( \mu \cdot Z^{s}, g^{s} \)

\( h_1^{s} \hat{T}^{s} \)

\( h_2^{s} \hat{U}^{s}_1 \)

**Goal:** recover \( \mu \)
Constructing Slotted Registered ABE

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

\[ \hat{T} = g^{r_1 + r_2} \]
\[ \hat{U}_1 = g^{u_2}, \hat{U}_2 = g^{u_1} \]

Goal: recover \( \mu \)

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 \( e(g^s, B_1) = e(g, g)^{as} e(g, h)^{st_i} = Z^s \cdot e(g, h)^{st_1} \)

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

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

Slot components:  \[ A_i = g^{t_i}, \quad B_i = g^{\alpha h^{t_i}} \]

Attribute component:  \[ U_1 = g^{u_1}, \quad U_2 = g^{u_2} \]

Aggregated master public key

\[ \widehat{T} = g^{r_1 + r_2} \]
\[ \widehat{U}_1 = g^{u_2}, \quad \widehat{U}_2 = g^{u_1} \]

\[ \text{pk}_1 = g^{r_1} \]
\[ S_1 = \{1\} \]

Goal: recover \( \mu \)

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

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

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

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

Step 3 (Policy Check): Compute \( e(A_1, h_2^s \widehat{U}_1^s) = e(g^{t_1}, h_2^s \widehat{U}_1^s) = e(g, h_2)^{st_1} e(g, g)^{st_1 u_2} \)
Constructing Slotted Registered ABE

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

\[
\mathring{T} = g^{r_1 + r_2}
\]
\[
\mathring{U}_1 = g^{u_2}, \mathring{U}_2 = g^{u_1}
\]

General components:

\[
\mu \cdot Z^s, g^s
\]

Slot component:

\[
h_1^s \mathring{T}^s
\]

Attribute component:

\[
h_2^s \mathring{U}_1^s
\]

pk_1 = g^{r_1}

\( S_1 = \{1\} \)

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

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

**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**

\( \hat{T} = g^{r_1+r_2} \)
\( \hat{U}_1 = g^{u_2}, \hat{U}_2 = g^{u_1} \)

\[ p_{k_1} = g^{r_1} \]
\[ S_1 = \{1\} \]

**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

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: \( U_1 = g^{u_1}, U_2 = g^{u_2} \)

\[
\begin{align*}
\text{Aggregated master public key} & \quad \widehat{T} = g^{r_1 + r_2} \\
\widehat{U}_1 & = g^{u_2}, \quad \widehat{U}_2 = g^{u_1}
\end{align*}
\]

\[
\begin{align*}
pk_1 & = g^{r_1} \\
S_1 & = \{1\}
\end{align*}
\]

General components: \( \mu \cdot Z^s, g^s \)

Slot component: \( h_1^s \widehat{T}^s \)

Attribute component: \( h_2^s \widehat{U}_1^s \)

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

- User should have the secret key for slot \( i \)
- Attributes associated with slot \( i \) should satisfy the challenge policy

Enforced by the **slot** components

Enforced by the **attribute** components
Constructing Slotted Registered ABE

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

\[ \hat{T} = g^{r_1+r_2} \]

\[ \hat{U}_1 = g^{u_2}, \quad \hat{U}_2 = g^{u_1} \]

\[ \text{pk}_1 = g^{r_1} \]
\[ S_1 = \{1\} \]

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 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

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

\( \hat{T} = g^{r_1+r_2} \)
\( \hat{U}_1 = g^{u_2}, \hat{U}_2 = g^{u_1} \)

pk_1 = g^{r_1}
S_1 = \{1\}

Key technical approach:
• 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

Replace attribute components with **linear secret sharing** of \( s \) to support policies with a linear secret sharing scheme
Reducing the CRS Size

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 $g^{t_iu_j}$ 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

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

**Goal:** give out \( g^{t_iu_j} \) for all \( i \neq j \), but without ability to compute \( g^{t_iu_i} \)

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

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

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

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 $d_1, \ldots, d_L$?**

**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 $\{d_1, \ldots, d_L\} = \mathcal{D}$; number of cross terms is then at most $2 \max \mathcal{D}$
Reducing the CRS Size

**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** \(d_1, \ldots, d_L\)?

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

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 \(\{d_1, \ldots, d_L\} = \mathcal{D}\); number of cross terms is then at most \(2 \cdot \max \mathcal{D}\)
Progression-Free Sets

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 $O(L^{\log_2 3})$

**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 $L^{1+o(1)} \Rightarrow$ registered ABE with CRS size $L^{1+o(1)}$.
Progression-Free Sets

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

\[
\begin{align*}
1 &= 001 \\
3 &= 010 \\
4 &= 011 \\
9 &= 100 \\
10 &= 101 \\
12 &= 110 \\
13 &= 111
\end{align*}
\]

**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 $L \log_3 2$

Implies registered ABE scheme with CRS size $O(L \log_3 2)$

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 $L^{1+o(1)} \Rightarrow$ registered ABE with CRS size $L^{1+o(1)}$

Achieves nearly linear CRS, but this approach cannot get to linear-size CRS
Registered ABE Summary

Key issuer replaced with key curator

"faculty" "CS"

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
- [HLW23, ZZGQ23]: quadratic-size CRS
- [GLW24]: 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 [HLW23] or witness encryption [FW23]

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

Suppose we want to encrypt a message to \{pk_1, pk_3, 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

Distributed broadcast encryption [BZ14]

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

Encrypt(pp, \{pk_i\}_{i \in S}, m) \rightarrow ct

Can encrypt a message \( m \) to any set of public keys.

Efficiency: |ct| = |m| + poly(\( \lambda, \log|S| \))

Decrypt(pp, \{pk_i\}_{i \in S}, sk, ct) \rightarrow m

Any secret key associated with broadcast set can decrypt.

Decryption does require knowledge of public keys in broadcast set.
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 $x$.

- Public-key directory:
  - $(1, pk_1, x)$
  - $(2, pk_2, x)$
  - $(3, pk_3, x)$
  - $(4, pk_4, x)$
  - $(5, pk_5, x)$

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

Aggregate public keys using slotted registered ABE scheme.

Encrypt with respect to $mpk$ to policy $P$ that accepts $x$.

Encrypt($mpk, x, P$)
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 $S = \{2, 3, 5\}$.

Encrypt with respect to $mpk$ to policy $P$ that accepts $x$.

**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.
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 $\Rightarrow$ 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 ($\ell$-succinct LWE)
Removing Trust from Functional Encryption

Encrypt(mpk, x) → x
mpk is essentially a key for a functional encryption scheme

(Alice, $f_1$) → $f_1(x)$
(Bob, $f_2$) → $f_2(x)$
(Carol, $f_3$) → $f_3(x)$

Aggregate public keys together

Goal: Support capabilities of functional encryption without a trusted authority
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 $\ell$-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!


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


Dario Fiore, Dimitris Kolonelos, and Paola de Perthuis. Cuckoo Commitments: Registration-Based Encryption and Key-Value Map Commitments for Large Spaces. ASIACRYPT 2023.
<table>
<thead>
<tr>
<th>Reference</th>
<th>Title</th>
</tr>
</thead>
</table>


