Access Control Encryption for General Policies from Standard Assumptions

Sam Kim    David J. Wu
Stanford University
Symmetric Encryption
Symmetric Encryption

Alice \[\text{Enc}(k, m)\] Bob
Symmetric Encryption

Alice \rightarrow Enc(k, m) \rightarrow m \rightarrow Bob
Symmetric Encryption
Symmetric Encryption

\[ Enc(k, m) \]
Symmetric Encryption

Must have many shared keys!
Functional Encryption [BSW11]
Functional Encryption [BSW11]
Functional Encryption [BSW11]
Functional Encryption [BSW11]

Functional Encryption: Fine-Grained Control of Decryptors
Can we also have fine-grained control of encryptors?
Can we also have fine-grained control of encryptors?

Applicable to many real-world scenarios:
Can we also have fine-grained control of encryptors?

Applicable to many real-world scenarios:

- Users allowed to work with data
Can we also have fine-grained control of encryptors?

Applicable to many real-world scenarios:

- Users allowed to work with data
- Users not allowed to publicly reveal data
Access Control Encryption [DHO16]

Can we also have fine-grained control of encryptors?

Applicable to many real-world scenarios:

- Users allowed to work with data
- Users not allowed to publicly reveal data
- Want to prevent malicious senders
Can we also have fine-grained control of encryptors?

Applicable to many real-world scenarios:

- Users allowed to work with data
- Users not allowed to publicly reveal data
- Want to prevent malicious senders
- Want to prevent mistakes or virus
Can we also have fine-grained control of encryptors?

Applicable to many real-world scenarios:

- Users allowed to work with data
- Users not allowed to publicly reveal data
- Want to prevent malicious senders
- Want to prevent mistakes or virus

Clearly impossible without some additional hardware measures.
Can we also have fine-grained control of encryptors?

Applicable to many real-world scenarios:

- Users allowed to work with data
- Users not allowed to publicly reveal data
- Want to prevent malicious senders
- Want to prevent mistakes or virus

Clearly impossible without some additional hardware measures.

[DHO16]: Access Control Encryption (ACE)
Access Control Encryption [DHO16]

Can we also have fine-grained control of encryptors?

Applicable to many real-world scenarios:

- Users allowed to work with data
- Users not allowed to publicly reveal data
- Want to prevent malicious senders
- Want to prevent mistakes or virus

Clearly impossible without some additional hardware measures.

[DHO16]: Access Control Encryption (ACE)

Each ciphertext is processed by a Sanitizer
Access Control Encryption \[\text{[DHO16]}\]

**Encryptors**

- \(id_1\)
- \(id_2\)
- \(id_3\)

**Decryptors**

- \(id_4\)
- \(id_5\)
- \(id_6\)

**Sanitizer**

**Fixed Predicate:** \(\pi(id_i, id_j) \rightarrow \{0, 1\}\)
Access Control Encryption [DHO16]

Fixed Predicate: \( \pi(id_i, id_j) \rightarrow \{0, 1\} \)
Access Control Encryption [DHO16]

Fixed Predicate: $\pi(id_i, id_j) \rightarrow \{0, 1\}$
Access Control Encryption [DHO16]

Fixed Predicate: \( \pi(id_i, id_j) \rightarrow \{0, 1\} \)
Correctness: User id\(_i\)’s ciphertext decryptable by user id\(_j\) iff
\(\pi(id_i, id_j) = 1\).
Access Control Encryption \[DHO16\]

**General Goal:** Minimize what Sanitizer learns!
No-Read Security

**Encryptors**

Fixed Predicate: $\pi(id_i, id_j) \rightarrow \{0, 1\}$

**Decryptors**
No-Read Security

**Requirement**: Without a predicate-satisfying key, adversary learns no information.
No-Write Security

Fixed Predicate: \( \pi(id_i, id_j) \rightarrow \{0, 1\} \)
No-Write Security

Fixed Predicate: \( \pi(id_i, id_j) \rightarrow \{0, 1\} \)
**No-Write Security**

**Requirement:** Without a predicate-satisfying pair of keys, adversary cannot distinguish: honestly sanitized `ctxt` vs. randomly generated `ctxt`
## Previous Works

<table>
<thead>
<tr>
<th>Construction</th>
<th>Predicate</th>
<th>Ciphertext Size</th>
<th>Assumption</th>
</tr>
</thead>
<tbody>
<tr>
<td>DHO16</td>
<td>arbitrary</td>
<td>$O(2^n)$</td>
<td>DDH or DCR</td>
</tr>
<tr>
<td>DHO16</td>
<td>arbitrary</td>
<td>$\text{poly}(n)$</td>
<td>iO</td>
</tr>
<tr>
<td>Construction</td>
<td>Predicate</td>
<td>Ciphertext Size</td>
<td>Assumption</td>
</tr>
<tr>
<td>--------------</td>
<td>------------</td>
<td>-----------------</td>
<td>------------</td>
</tr>
<tr>
<td>DHO16</td>
<td>arbitrary</td>
<td>$O(2^n)$</td>
<td>DDH or DCR</td>
</tr>
<tr>
<td>DHO16</td>
<td>arbitrary</td>
<td>poly($n$)</td>
<td>iO</td>
</tr>
<tr>
<td>FGKO17</td>
<td>restricted</td>
<td>poly($n$)</td>
<td>SXDH</td>
</tr>
<tr>
<td>Construction</td>
<td>Predicate</td>
<td>Ciphertext Size</td>
<td>Assumption</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------</td>
<td>-----------------</td>
<td>----------------</td>
</tr>
<tr>
<td>DHO16</td>
<td>arbitrary</td>
<td>$O(2^n)$</td>
<td>DDH or DCR</td>
</tr>
<tr>
<td>DHO16</td>
<td>arbitrary</td>
<td>poly($n$)</td>
<td>iO</td>
</tr>
<tr>
<td>FGKO17</td>
<td>restricted</td>
<td>poly($n$)</td>
<td>SXDH</td>
</tr>
<tr>
<td>TZMT17</td>
<td>arbitrary</td>
<td>$O(2^n)$</td>
<td>LWE</td>
</tr>
</tbody>
</table>
### Our Results

<table>
<thead>
<tr>
<th>Construction</th>
<th>Predicate</th>
<th>Ciphertext Size</th>
<th>Assumption</th>
</tr>
</thead>
<tbody>
<tr>
<td>DHO16</td>
<td>arbitrary</td>
<td>$O(2^n)$</td>
<td>DDH or DCR</td>
</tr>
<tr>
<td>DHO16</td>
<td>arbitrary</td>
<td>poly($n$)</td>
<td>iO</td>
</tr>
<tr>
<td>FGKO17</td>
<td>restricted</td>
<td>poly($n$)</td>
<td>SXDH</td>
</tr>
<tr>
<td>TZMT17</td>
<td>arbitrary</td>
<td>$O(2^n)$</td>
<td>LWE</td>
</tr>
</tbody>
</table>

**This Work:** arbitrary poly($n$) DDH, RSA, LWE
Let’s first achieve: **No-Write Security**
Let’s first achieve: No-Write Security

**Simple Idea:** Provide cert $\sigma_{id}$ of id as encryption key
Let’s first achieve: **No-Write Security**

**Simple Idea:** Provide cert \( \sigma_{\text{id}} \) of id as encryption key

Encryptor can attach \((\text{id}_i, \sigma_{\text{id}_i})\) to message.
Let’s first achieve: No-Write Security

**Simple Idea:** Provide cert $\sigma_{id}$ of id as encryption key

Encryptor can attach $(id_i, \sigma_{id_i})$ to message.

Sanitizer verifies $(id_i, \sigma_{id_i})$ and encrypt to all identities $id_j$ for which

$$\pi(id_i, id_j) = 1$$
Starting Point: Basic Cert

Fixed Predicate: \( \pi(id_i, id_j) \rightarrow \{0, 1\} \)

Encryptors

id_1
\( \sigma_{id_1} \)

id_2
\( \sigma_{id_2} \)

id_3
\( \sigma_{id_3} \)

Decryptors

sk_{id_4}

sk_{id_5}

sk_{id_6}
Starting Point: Basic Cert

Fixed Predicate: \( \pi(id_i, id_j) \rightarrow \{0, 1\} \)
Starting Point: Basic Cert

Fixed Predicate: $\pi(id_i, id_j) \rightarrow \{0, 1\}$

Encryptors

Decryptors

Sanitizer

$\text{Enc}(sk_{id_4}, m) || \text{Enc}(sk_{id_6}, m)$
Starting Point: Basic Cert

Fixed Predicate: \( \pi(id_i, id_j) \rightarrow \{0, 1\} \)
Starting Point: Basic Cert

Fixed Predicate: \( \pi(id_i, id_j) \rightarrow \{0, 1\} \)
Next Goal: Compact Ciphertext
Next Goal: Compact Ciphertext

**Useful Tool:** Predicate Encryption [BW07,KSW08]
Compact Ciphertext: Predicate Encryption

Next Goal: Compact Ciphertext

**Useful Tool:** Predicate Encryption [BW07,KSW08]

Encryption algorithm takes in \((\text{attribute}, \text{message})\) pair

\[
\text{Enc}(x, m) \rightarrow \text{ct}_{x,m}.
\]

Decryption keys associated with function \(f\): \(sk_f\)

\[
\text{Dec}(sk_f, \text{ct}_{x,m}) = \begin{cases} 
m & f(x) = 1 \\
\bot & f(x) = 0 \end{cases}.
\]
Next Goal: Compact Ciphertext

**Useful Tool:** Predicate Encryption [BW07,KSW08]

Encryption algorithm takes in (attribute, message) pair

\[ \text{Enc}(x, m) \rightarrow \text{ct}_{x,m}. \]

Decryption keys associated with function \( f \): \( \text{sk}_f \)

\[ \text{Dec}(\text{sk}_f, \text{ct}_{x,m}) = \begin{cases} m & f(x) = 1 \\ \bot & f(x) = 0 \end{cases}. \]

**Security guarantee:** \((x, m)\) hidden if only given \( \text{sk}_f \)'s for which \( f(x) = 0 \)
Compact Ciphertext: Predicate Encryption

Next Goal: Compact Ciphertext

**Useful Tool:** Predicate Encryption [BW07,KSW08]

Encryption algorithm takes in (attribute, message) pair

\[ \text{Enc}(x,m) \rightarrow ct_{x,m}. \]

Decryption keys associated with function \( f \): \( sk_f \)

\[
\text{Dec}(sk_f, ct_{x,m}) = \begin{cases} 
  m & f(x) = 1 \\
  \bot & f(x) = 0 
\end{cases}
\]

**Security guarantee:** \((x,m)\) hidden if only given \( sk_f \)'s for which \( f(x) = 0 \)

**Idea:**

- Sanitizer when given \((id_i, \sigma_{id_i}, m)\) encrypts \( \text{Enc}(id_i, m) \)
- Decryptor with \( id_j \) given \( sk_{\pi(\cdot,id_j)} \)
Compact Ciphertext: Predicate Encryption

Fixed Predicate: \( \pi(id_i, id_j) \rightarrow \{0, 1\} \)
Compact Ciphertext: Predicate Encryption

**Fixed Predicate:** $\pi(id_i, id_j) \rightarrow \{0, 1\}$

Encryptors

- $id_1$
- $\sigma_{id_1}$

- $id_2$
- $\sigma_{id_2}$

- $id_3$
- $\sigma_{id_3}$

Decryption

- $sk_{\pi(\cdot, id_4)}$
- $sk_{\pi(\cdot, id_5)}$
- $sk_{\pi(\cdot, id_6)}$
Compact Ciphertext: Predicate Encryption

Fixed Predicate: $\pi(id_i, id_j) \rightarrow \{0, 1\}$

Encryptors:
- $id_1$
- $\sigma_{id_1}$

- $id_2$
- $\sigma_{id_2}$

- $id_3$
- $\sigma_{id_3}$

Decryptors:
- $sk_{\pi(\cdot, id_4)}$

Sanitizer:
- $Enc(id_1, m)$

- $sk_{\pi(\cdot, id_5)}$

- $sk_{\pi(\cdot, id_6)}$
Compact Ciphertext: Predicate Encryption

Fixed Predicate: $\pi(id_i, id_j) \rightarrow \{0, 1\}$
Next Goal: No-Read Security
Next Goal: No-Read Security

**Problem:** Sanitizer learns too much information!
Next Goal: **No-Read Security**

**Problem:** Sanitizer learns too much information!

Current task:

1. Check signature \((id, \sigma_{id})\)
2. Encrypt Enc(id, \(m\))
Next Goal: **No-Read Security**

**Problem**: Sanitizer learns too much information!

Current task:

1. Check signature \((id, \sigma_{id})\)
2. Encrypt \(\text{Enc}(id, m)\)

Can the sanitizer perform its task blind-folded?
Let’s use \textit{Functional Encryption} [BSW11]!
Let’s use Functional Encryption [BSW11]!

**Bounded key** Functional Encryption can be constructed from standard assumptions [SS10, GVW12, GKPVZ13, ...]
Let’s use Functional Encryption [BSW11]!

Bounded key Functional Encryption can be constructed from standard assumptions [SS10, GVW12, GKPVZ13, ...]

**Idea**: Provide FE secret key $sk_F$ where $F$ defined as follows:

$F(id, \sigma, m)$:

1. Check $(id, \sigma)$
2. If valid, then return $Enc(id, m)$
Hiding Information from Sanitizer

Encryptors

Fixed Predicate: \( \pi(id_i, id_j) \rightarrow \{0, 1\} \)

Decryptors

\(id_1\)
\(\sigma_{id_1}\)

\(id_2\)
\(\sigma_{id_2}\)

\(id_3\)
\(\sigma_{id_3}\)

\(sk_F\)

\(sk_{\pi(\cdot, id_4)}\)

\(sk_{\pi(\cdot, id_5)}\)

\(sk_{\pi(\cdot, id_6)}\)
Hiding Information from Sanitizer

Encryptors

\[ \text{Fixed Predicate: } \pi(id_i, id_j) \rightarrow \{0, 1\} \]

Decryptors

\[ \text{FE.Enc}(id_1, \sigma_{id_1}, m) \]

\[ \text{sk}_F \]

\[ \text{sk}_{\pi(\cdot, id_4)} \]

\[ \text{sk}_{\pi(\cdot, id_5)} \]

\[ \text{sk}_{\pi(\cdot, id_6)} \]
Hiding Information from Sanitizer

Encryptors

\[ \pi(\text{id}_i, \text{id}_j) \rightarrow \{0, 1\} \]

Sanitizer

\[ \text{PE.Enc(id}_1, m) \]

Decryptors

[Diagram showing relationships and keys]

62
Hiding Information from Sanitizer

Fixed Predicate: $\pi(id_i, id_j) \rightarrow \{0, 1\}$
Hiding Information from Sanitizer

Encryptors

Fixed Predicate: \( \pi(id_i, id_j) \rightarrow \{0, 1\} \)

Decryptors

Sanitizer
Requires a number of subtle details to argue security
Security Details

Requires a number of subtle details to argue security

**Note**: encryption is a randomized function!
Requires a number of subtle details to argue security

**Note**: encryption is a randomized function!

Need to embed additional PRF keys as part of the message and FE keys to derive randomness
Requires a number of subtle details to argue security

**Note:** encryption is a randomized function!

Need to embed additional PRF keys as part of the message and FE keys to derive randomness

Rely on FE for **randomized functionalities** [GJKS15, AW17]

Construction can be based on DDH + RSA.
Extensions

Additional extensions to ACE definition:

1. Dynamic policies
Additional extensions to ACE definition:

1. Dynamic policies
2. Fine-grained sender policies
Extensions

Additional extensions to ACE definition:

1. Dynamic policies
2. Fine-grained sender policies
3. Beyond All-or-Nothing Decryption
Extensions

Additional extensions to ACE definition:

1. Dynamic policies
2. Fine-grained sender policies
3. Beyond All-or-Nothing Decryption

Provide modifications of our constructions can achieve these extended definition.
Conclusion

Future directions:

• Can we construct ACE from a single algebraic assumption?
Future directions:

- Can we construct ACE from a single algebraic assumption?
- Connection to other cryptographic primitives?
Future directions:

- Can we construct ACE from a single algebraic assumption?
- Connection to other cryptographic primitives?

Thanks!