Functional Encryption

Lecture 27
Functional Encryption

- Plain encryption: for secure communication. Does not allow modifying encrypted data.
- Homomorphic Encryption: allows computation on encrypted data, but result remains encrypted.
- Functional Encryption: allows computation so that results are available in the clear.
  - Many interesting applications
  - Active/evolving area of research
  - Will sample a few results
Functional Encryption

- Ciphertext: Enc(Msg). Msg is fully or partially hidden.
  - e.g., Msg = (T,M) where T is a public tag (a.k.a index).
- Key: KeyGen(f). Function f could be fully/partly hidden or not.
- “Decryption” Dec( Enc(Msg), KeyGen(f) ) → f(Msg)
  - Public-index FE: f(T,M) = ⊥ if g(T)=0; f'(M) if g(T)=1

  Should reveal nothing else
  - Can formulate different levels of security
- Can be public-key (anyone can encrypt) or not
- KeyGen requires a master secret-key. If public-key, encryption needs only master public-key, else needs master secret-key.
Functional Encryption

- Trivial Example: when the family of functions is small
  - Keys will be issued only for \( f \in \{f_1, \ldots, f_N\} \) for a small \( N \)
  - Can pre-compute all the functions, and encrypt the results!
    - \( \text{Enc}(\text{Msg}) = (c_1, \ldots, c_N) \), where \( c_i = E_{PK_i}(f_i(\text{Msg})) \) using a PKE encryption scheme (with \( N \) independent keys)
    - \( \text{KeyGen}(f_i) = (i, SK_i) \)
  - Not function-hiding
    - If not public-key, can make it function-hiding by numbering \( f \)'s randomly
Examples: IBE & ABE

- A public-index FE, where the index is the ID
  - Functions $f_{ID} : f_{ID}(ID', M) = M$ if $ID = ID'$; ⊥ otherwise
  - Fuzzy IBE: $f_{ID}(ID', M) = M$ if ID “close to” ID'; ⊥ otherwise

- Attribute-Based Encryption: if the index/key is not just a single ID, but a vector of “attributes” and a “policy” as to which attribute combinations allow revealing the message
  - Ciphertext-Policy ABE: Index is a policy (from a simple class); the function in the key gives a set of attributes
  - Key-Policy ABE: Index is a set of attributes; the function in the key gives a policy
Key-Policy ABE
Key-Policy ABE

(Binary) Attributes will be assigned to a ciphertext when creating the ciphertext
Key-Policy ABE

(Binary) Attributes will be assigned to a ciphertext when creating the ciphertext

Policies will be assigned to users/keys by an authority who creates the keys
Key-Policy ABE

(Binary) Attributes will be assigned to a ciphertext when creating the ciphertext

Policies will be assigned to users/keys by an authority who creates the keys

A key can decrypt only those ciphertexts whose attributes satisfy the policy
Key-Policy ABE

(Binary) Attributes will be assigned to a ciphertext when creating the ciphertext

Policies will be assigned to users/keys by an authority who creates the keys

A key can decrypt only those ciphertexts whose attributes satisfy the policy

E.g. Applications
Key-Policy ABE

(Binary) Attributes will be assigned to a ciphertext when creating the ciphertext.

Policies will be assigned to users/keys by an authority who creates the keys.

A key can decrypt only those ciphertexts whose attributes satisfy the policy.

E.g. Applications

Fuzzy IBE
Key-Policy ABE

- (Binary) Attributes will be assigned to a ciphertext when creating the ciphertext.
- Policies will be assigned to users/keys by an authority who creates the keys.
  - A key can decrypt only those ciphertexts whose attributes satisfy the policy.
- E.g. Applications
  - Fuzzy IBE
  - Audit log inspection: grant the auditor the authority to read only messages with certain attributes
A KP-ABE Scheme
A KP-ABE Scheme

A construction that supports “linear policies” (a.k.a. Monotone Span Programs)
A KP-ABE Scheme

A construction that supports “linear policies” (a.k.a. Monotone Span Programs)

Policy corresponds to a (monotonic) access structure (sets of attributes that when pooled satisfy the policy)
A KP-ABE Scheme

A construction that supports “linear policies” (a.k.a. Monotone Span Programs)

Policy corresponds to a (monotonic) access structure (sets of attributes that when pooled satisfy the policy)

Linear: Matrix $L$ with each row labeled by an attribute, such that a set of attributes $S$ satisfies the policy iff
A KP-ABE Scheme

A construction that supports “linear policies” (a.k.a. Monotone Span Programs)

Policy corresponds to a (monotonic) access structure (sets of attributes that when pooled satisfy the policy)

Linear: Matrix L with each row labeled by an attribute, such that a set of attributes S satisfies the policy iff

there is a vector v such that \( v \cdot L = [1 \ 1 \ ... \ 1] \)
A KP-ABE Scheme

A construction that supports “linear policies” (a.k.a. Monotone Span Programs)

Policy corresponds to a (monotonic) access structure (sets of attributes that when pooled satisfy the policy)

Linear: Matrix L with each row labeled by an attribute, such that a set of attributes S satisfies the policy iff

there is a vector v such that \( v \cdot L = [1 \ 1 \ \ldots \ \ 1] \)
and, labels corresponding to non-zero entries of v are all contained in S
A KP-ABE Scheme

A construction that supports “linear policies” (a.k.a. Monotone Span Programs)

Policy corresponds to a (monotonic) access structure (sets of attributes that when pooled satisfy the policy)

Linear: Matrix L with each row labeled by an attribute, such that a set of attributes S satisfies the policy iff

- there is a vector v such that $v L = [1 \ 1 \ ... \ 1]$
- and, labels corresponding to non-zero entries of v are all contained in S

Linear algebra over some finite field (e.g. GF(p))
A KP-ABE Scheme

A construction that supports “linear policies” (a.k.a. Monotone Span Programs)

Policy corresponds to a (monotonic) access structure (sets of attributes that when pooled satisfy the policy)

Linear: Matrix \( L \) with each row labeled by an attribute, such that a set of attributes \( S \) satisfies the policy iff

- there is a vector \( v \) such that \( v \ L = [1 \ 1 \ ... \ 1] \)
- and, labels corresponding to non-zero entries of \( v \) are all contained in \( S \)

Linear algebra over some finite field (e.g. \( GF(p) \) )

For efficiency need a small matrix
Example of a “Linear Policy”
Example of a “Linear Policy”

Consider this policy, over 7 attributes
Example of a “Linear Policy”

Consider this policy, over 7 attributes
Example of a “Linear Policy”

Consider this policy, over 7 attributes

L:
Example of a “Linear Policy”

Consider this policy, over 7 attributes

L:

```
0 1 1 1
1 0 0 0
1 1 0 1
0 0 1 0
1 1 1 0
1 1 1 0
0 0 0 1
```
Example of a “Linear Policy”

Consider this policy, over 7 attributes

L:

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Can generalize AND/OR to threshold gates
A KP-ABE Scheme
A KP-ABE Scheme

MPK: $g, Y = e(g,g)^y, T = (g^{t_1}, ..., g^{t_n})$ (n attributes)
A KP-ABE Scheme

MPK: $g, Y = e(g,g)^y, T = (g^{t_1}, \ldots, g^{t_n})$ (n attributes)

MSK: $y$ and $t_a$ for each attribute $a$
A KP-ABE Scheme

- **MPK**: $g, Y = e(g,g)^y, T = (g^{t_1}, ..., g^{t_n})$ (n attributes)
- **MSK**: $y$ and $t_a$ for each attribute $a$
- **Enc(m,A;s)** = ( $A$, $\{ T_a^s \}_{a \in A}$, $M.Y^s$ )
A KP-ABE Scheme

- **MPK**: \( g, Y = e(g, g)^y, T = (g^{t_1}, \ldots, g^{t_n}) \) (n attributes)
- **MSK**: \( y \) and \( t_a \) for each attribute \( a \)
- **Enc(m,A;s)**: \( (A, \{ T_a^s \}_{a \in A}, M.Y^s) \)
- **SK for policy L (with d rows)**: Let \( u = (u_1 \ldots u_d) \) s.t. \( \sum_i u_i = y \). For each row \( i \), let \( x_i = \langle L_i, u \rangle / \text{label}(i) \). Let \( \text{Key } X = \{ g^{x_i} \}_{i=1 \text{ to } d} \)
A KP-ABE Scheme

MPK: \( g, Y = e(g, g)^y, T = (g^{t_1}, \ldots, g^{t_n}) \) (n attributes)

MSK: \( y \) and \( t_a \) for each attribute \( a \)

Enc\( (m, A; s) = (A, \{ T_a^s \}_{a \in A}, M.Y^s) \)

SK for policy \( L \) (with \( d \) rows): Let \( u = (u_1, \ldots, u_d) \) s.t. \( \sum_i u_i = y \).
For each row \( i \), let \( x_i = \langle L_i, u \rangle / t_{l_{\text{label}(i)}} \).
Let Key \( X = \{ g^{x_i} \}_{i=1}^d \)

Dec \( (A, \{ Z_a \}_{a \in A}, c); \{ X_i \}_{\text{row } i} ) \) : Get \( Y^s = \prod_{i: l_{\text{label}(i)} \in A} e(Z_{l_{\text{label}(i)}}, X_i)^{v_i} \)
where \( v = [v_1, \ldots, v_d] \) s.t. \( v_i = 0 \) if \( l_{\text{label}(i)} \notin A \), and \( v.L = [1 \ldots 1] \)
A KP-ABE Scheme

**MPK:** \( g, Y = e(g, g)^y, T = (g^{t_1}, \ldots, g^{t_n}) \) (n attributes)

**MSK:** \( y \) and \( t_a \) for each attribute \( a \)

\[\text{Enc}(m, A; s) = (A, \{ T_a^s \}_{a \in A}, M.Y^s)\]

**SK for policy L (with d rows):** Let \( u = (u_1 \ldots u_d) \) s.t. \( \sum_i u_i = y \).

For each row \( i \), let \( x_i = <L_i, u> / t_{\text{label}(i)} \). Let Key \( X = \{ g^{x_i} \}_{i=1}^d \)

\[\text{Dec} ((A, \{Z_a\}_{a \in A}, c); \{X_i\}_{\text{row } i}) : \text{Get } Y^s = \prod_{i : \text{label}(i) \in A} e(Z_{\text{label}(i)}, X_i)^{v_i}\]

where \( v = [v_1 \ldots v_d] \) s.t. \( v_i = 0 \) if \( \text{label}(i) \notin A \), and \( v.L = [1 \ldots 1]\)

**CPA security based on Decisional-BDH**
A KP-ABE Scheme

**MPK:** $g, Y=e(g,g)^y$, $T = (g^{t_1},..., g^{t_n})$ (n attributes)

**MSK:** $y$ and $t_a$ for each attribute $a$

**Enc($m,A;s$) =** $(A, \{ T_a^s \}_{a \in A}, M.Y^s )$

**SK for policy L (with d rows):** Let $u=(u_1 \ldots u_d)$ s.t. $\sum_i u_i = y$.

For each row $i$, let $x_i = <L_i,u>/t_{\text{label}(i)}$. Let Key $X = \{ g^{x_i} \}_{i=1 \text{ to } d}$

**Dec ( $(A,\{Z_a\}_{a \in A},c);$ $\{X_i\}_{\text{row } i}$ ) :** Get $Y^s = \prod_{i: \text{label}(i) \in A} e(Z_{\text{label}(i)},X_i)^{v_i}$

where $v = [v_1 \ldots v_d]$ s.t. $v_i=0$ if $\text{label}(i) \notin A$, and $v.L = [1\ldots1]$

**CPA security based on Decisional-BDH**

Choosing a random vector $u$ for each key helps in preventing collusion
Ciphertext-Policy ABE
Ciphertext-Policy ABE

Each user in the system has attributes; receives a key (or "key bundle") from an authority for its set of attributes.
Ciphertext-Policy ABE

Each user in the system has attributes; receives a key (or "key bundle") from an authority for its set of attributes.

Ciphertext contains a policy (a boolean predicate over the attribute space).
Ciphertext-Policy ABE

Each user in the system has attributes; receives a key (or “key bundle”) from an authority for its set of attributes.

Ciphertext contains a policy (a boolean predicate over the attribute space).

If a user’s attribute set satisfies the policy, can use its key bundle to decrypt the ciphertext.
Ciphertext-Policy ABE

Each user in the system has attributes; receives a key (or “key bundle”) from an authority for its set of attributes

Ciphertext contains a policy (a boolean predicate over the attribute space)

If a user’s attribute set satisfies the policy, can use its key bundle to decrypt the ciphertext

Multiple users cannot pool their attributes together
Ciphertext-Policy ABE

Each user in the system has attributes; receives a key (or “key bundle”) from an authority for its set of attributes.

Ciphertext contains a policy (a boolean predicate over the attribute space).

If a user’s attribute set satisfies the policy, can use its key bundle to decrypt the ciphertext.

Multiple users cannot pool their attributes together.

Application: End-to-End privacy in Attribute-Based Messaging.
Predicate Encryption
Predicate Encryption

Non-public-index FE where ciphertext $M=(c,m)$ (neither public) and function $f$ contains a predicate $\pi$ (also hidden) s.t.
Predicate Encryption

Non-public-index FE where ciphertext $M = (c, m)$ (neither public) and function $f$ contains a predicate $\pi$ (also hidden) s.t.

$$f(M) = m \text{ if } \pi(c) = 0; \bot \text{ otherwise}$$
Predicate Encryption

Non-public-index FE where ciphertext $M=(c,m)$ (neither public) and function $f$ contains a predicate $\pi$ (also hidden) s.t.

$$f(M) = m \text{ if } \pi(c)=0; \perp \text{ otherwise}$$

Application, e.g., to searching on encrypted data: Encrypted files tagged with Predicate-Encryption ciphertexts (with empty $m$). Client sends a key for a predicate to the server who sifts through all tags and retrieves matching ones.
Predicate Encryption

Non-public-index FE where ciphertext $M=(c,m)$ (neither public) and function $f$ contains a predicate $\pi$ (also hidden) s.t.

$f(M) = m$ if $\pi(c)=0$; $\perp$ otherwise

Application, e.g., to searching on encrypted data: Encrypted files tagged with Predicate-Encryption ciphertexts (with empty $m$). Client sends a key for a predicate to the server who sifts through all tags and retrieves matching ones

e.g., *Inner-product predicate*: $M=(c,m)$ where $c$ is a vector. Predicate $\pi_d$ contains a vector $d$; $\pi_d(c)=0$ iff $\langle c,d \rangle = 0$
Predicate Encryption

Non-public-index FE where ciphertext $M=(c,m)$ (neither public) and function $f$ contains a predicate $\pi$ (also hidden) s.t.

- $f(M) = m$ if $\pi(c)=0$; $\perp$ otherwise

Application, e.g., to searching on encrypted data: Encrypted files tagged with Predicate-Encryption ciphertexts (with empty $m$). Client sends a key for a predicate to the server who sifts through all tags and retrieves matching ones

- e.g., Inner-product predicate: $M=(c,m)$ where $c$ is a vector. Predicate $\pi_d$ contains a vector $d$; $\pi_d(c)=0$ iff $\langle c,d \rangle=0$

  A building block for many other predicates
**Predicate Encryption**

Non-public-index FE where ciphertext $M=(c,m)$ (neither public) and function $f$ contains a predicate $\pi$ (also hidden) s.t.

$$f(M) = m \text{ if } \pi(c)=0; \perp \text{ otherwise}$$

Application, e.g., to searching on encrypted data: Encrypted files tagged with Predicate-Encryption ciphertexts (with empty $m$). Client sends a key for a predicate to the server who sifts through all tags and retrieves matching ones.

E.g., *Inner-product predicate*: $M=(c,m)$ where $c$ is a vector. Predicate $\pi_d$ contains a vector $d$; $\pi_d(c)=0$ iff $\langle c, d \rangle = 0$

A building block for many other predicates

Constructions based on the Decision Linear assumption
Predicate Encryption

Non-public-index FE where ciphertext $M=(c,m)$ (neither public) and function $f$ contains a predicate $\pi$ (also hidden) s.t.

$f(M) = m$ if $\pi(c)=0$; $\bot$ otherwise

Application, e.g., to searching on encrypted data: Encrypted files tagged with Predicate-Encryption ciphertexts (with empty $m$). Client sends a key for a predicate to the server who sifts through all tags and retrieves matching ones

e.g., *Inner-product predicate*: $M=(c,m)$ where $c$ is a vector. Predicate $\pi_d$ contains a vector $d$; $\pi_d(c)=0$ iff $\langle c,d \rangle = 0$

A building block for many other predicates

Constructions based on the Decision Linear assumption

$(f,g,h,f^x,g^y,h^{x+y})$ and $(f,g,h,f^x,g^y,h^z)$ indistinguishable for random $f$, $g$, $h$, $x$, $y$, $z$. 
Single-Key FE

- In which key for only one function will be ever be released
  - Function is not known when ciphertexts are created (otherwise trivial [Why?])

- A single-key FE scheme supporting arbitrary functions (with circuits of a priori bounded size)
  - Encryption of $m$ is a Garbled circuit encoding the universal function: $F(x,f) = f(x)$, with $x$ set to $m$
  - Plus, $2n$ encrypted wire labels for the $n$ input wires of $f$ (using $2n$ public-keys in the master public-key)
  - Key for $f$: $n$ secret-keys corresponding to the $n$ bits of $f$
  - Can decrypt the labels of $f$ → can evaluate $F(x,f)$
No Unbounded Sim-FE

Suppose we require simulation-based security for FE

Then there are function families which have no FE scheme that supports releasing an unbounded number of keys

e.g., The message is the seed of the PRF. The function evaluates the PRF on an input (i.e., one key for each input)

Even suppose that the simulator knows a priori the set of inputs for which the adversary will obtain keys

\{ PRF_s(x_i) \mid i=1 \text{ to } N \} \text{ are } N \text{ k-bit pseudorandom strings}

Simulation should encode them into an L-bit string (i.e., the simulated ciphertext)

If \( Nk \gg L \), not possible for truly random strings, and hence for pseudorandom strings too
Unbounded FE from Obfuscation

- Indistinguishability based definition for FE
- Indistinguishability Obfuscation (iO) suffices
- Simpler if we have a slightly stronger obfuscation:

  - $KeyGen(f) = (f, \text{sign}_{SK}(f))$, where $SK$ is the signing key corresponding to a $VK$ in the master public-key

  - $Enc(msg) = \text{Obfuscation of the following program}$:

    - Accept $(f, \sigma)$. If $\text{Verify}_{VK}(f, \sigma)$, then output $f(msg)$

  - $Dec(C, K) : \text{run } C \text{ (which is a program) on input } K=(f, \sigma)$
Consider implementing an encrypted database: all values are kept encrypted, but insertion, deletion, look-up etc. should be possible publicly.

Need to compare pairs of ciphertexts. Not a ciphertext and a key.

More generally, compute $f(x_1, ..., x_d)$ given independently generated ciphertexts of $x_i$'s (for a fixed $f$, or a family of $f$'s).

Public-key or private-key setting

- Or a mix: some arguments to $f$ can be publicly encrypted, and others cannot be.

IND security: cannot learn a challenge bit from keys/ciphertexts, if it cannot be learned in an IDEAL model.
Multi-Input FE

- Can be constructed using obfuscation

Enc(x,i), i.e., encrypt x as $i^{th}$ argument: $E_{PK_i}(x)$, where E is the encryption algorithm in a CCA-secure PKE scheme. PK_i's in master PK

KeyGen(f) : Obfuscate the following program:

- Accept $d$ ciphertexts $c_1,...,c_d$. $x_i \leftarrow D_{SK_i}(c_i)$ for all i.

  If all decryptions valid, output $f(x_1,...,x_d)$

CCA-security needed to prevent the adversary from evaluating f on inputs related to encrypted messages

- To use “realizable” obfuscation (involving only one hidden bit): instead of CCA security, use $(c,c',\pi)$, where $\pi$ is a “proof” that c and c' encrypt the same message under two keys.
Today
Today

- Functional Encryption
Today

- Functional Encryption
- A relatively new and powerful primitive
Today

- Functional Encryption
  - A relatively new and powerful primitive
  - (Greatly) Generalizes Identity-Based Encryption
Today

- Functional Encryption
  - A relatively new and powerful primitive
  - (Greatly) Generalizes Identity-Based Encryption
  - Constructions using bilinear-pairings for special cases (e.g., Attribute-Based Encryption for “linear policies”, Inner-product Predicate encryption)
Today

Functional Encryption

A relatively new and powerful primitive

(Greatly) Generalizes Identity-Based Encryption

Constructions using bilinear-pairings for special cases (e.g., Attribute-Based Encryption for “linear policies”, Inner-product Predicate encryption)

Fairly practical
Today

- Functional Encryption
  - A relatively new and powerful primitive
  - (Greatly) Generalizes Identity-Based Encryption
  - Constructions using bilinear-pairings for special cases (e.g., Attribute-Based Encryption for “linear policies”, Inner-product Predicate encryption)
  - Fairly practical
  - Based on multi-linear maps/obfuscation in general
Today

- Functional Encryption
  - A relatively new and powerful primitive
  - (Greatly) Generalizes Identity-Based Encryption
  - Constructions using bilinear-pairings for special cases (e.g., Attribute-Based Encryption for “linear policies”, Inner-product Predicate encryption)
  - Fairly practical
  - Based on multi-linear maps/obfuscation in general
  - Not yet practical