Predicate Encryption for Private and Searchable Remote Storage

Giuseppe Persiano

Dipartimento di Informatica ed Appl.
"Renato M. Capocelli"
Università di Salerno
giuper@dia.unisa.it

Workshop on Cryptography and Security in Clouds
Zurich, Switzerland
March 15-16, 2011

This talk describes joint work with:

1. Carlo Blundo
2. Angelo De Caro
3. Vincenzo Iovino

Outline

Outline

Outline

1. Storing Data in a Cloud
2. Hidden Vector Encryption
3. Implementation

Outline

Outline

Secure Remote Storage

Secure Remote Storage

- A Cloud has huge storage capabilities and can be accessed from anywhere;
- We consider simple case of a Data Owner storing his data on an Untrusted Storage;

Secure Remote Storage

Secure Remote Storage

- A Cloud has huge storage capabilities and can be accessed from anywhere;
- We consider simple case of a Data Owner storing his data on an Untrusted Storage;
- DOwner can assume:
  - UStorage does not destroy the data

Secure Remote Storage

Secure Remote Storage

- A Cloud has huge storage capabilities and can be accessed from anywhere;
- We consider simple case of a Data Owner storing his data on an Untrusted Storage;
- DOwner can assume:
  - UStorage does not destroy the data (enforce using Duplication);

Secure Remote Storage

Secure Remote Storage

- A Cloud has huge storage capabilities and can be accessed from anywhere;
- We consider simple case of a Data Owner storing his data on an Untrusted Storage;
- DOwner can assume:
  - UStorage does not destroy the data (enforce using Duplication);
  - UStorage does not modify data

Secure Remote Storage

Secure Remote Storage

- A Cloud has huge storage capabilities and can be accessed from anywhere;
- We consider simple case of a Data Owner storing his data on an Untrusted Storage;
- DOwner can assume:
  - UStorage does not destroy the data (enforce using Duplication);
  - UStorage does not modify data (enforce using Authentication Code);

Secure Remote Storage

Secure Remote Storage

- A Cloud has huge storage capabilities and can be accessed from anywhere;
- We consider simple case of a Data Owner storing his data on an Untrusted Storage;
- DOwner can assume:
  - UStorage does not destroy the data (enforce using Duplication);
  - UStorage does not modify data (enforce using Authentication Code);
  - UStorage does not read the data

Secure Remote Storage

Secure Remote Storage

- A Cloud has huge storage capabilities and can be accessed from anywhere;
- We consider simple case of a Data Owner storing his data on an Untrusted Storage;
- DOwner can assume:
  - UStorage does not destroy the data (enforce using Duplication);
  - UStorage does not modify data (enforce using Authentication Code);
  - UStorage does not read the data (enforce using Encryption);

Secure Remote Storage

Secure Remote Storage

- A Cloud has huge storage capabilities and can be accessed from anywhere;
- We consider simple case of a Data Owner storing his data on an Untrusted Storage;
- DOwner can assume:
  - UStorage does not destroy the data (enforce using Duplication);
  - UStorage does not modify data (enforce using Authentication Code);
  - UStorage does not read the data (enforce using Encryption);

Secure Remote Storage

1. In the beginning is the Data

First Name	Last Name	Affiliation
Christian	Cachin	IBM
Giuseppe	Persiano	SAL
Ahmad-Reza	Sadeghi	TUD
Matthias	Schunter	IBM
Paulo	Verissimo	LIS

Secure Remote Storage

1. In the beginning is the Data

First Name	Last Name	Affiliation
Christian	Cachin	IBM
Giuseppe	Persiano	SAL
Ahmad-Reza	Sadeghi	TUD
Matthias	Schunter	IBM
Paulo	Verissimo	LIS

2. Encrypt and obtain

First Name	Last Name	Affiliation
E(PK, Christian)	E(PK, Cachin)	E(PK, IBM)
E(PK, Giuseppe)	E(PK, Persiano)	E(PK, SAL)
E(PK, Ahmad-Reza)	E(PK, Sadeghi)	E(PK, TUD)
E(PK, Matthias)	E(PK, Schunter)	E(PK, IBM)
E(PK, Paulo)	E(PK, Verissimo)	E(PK, LIS)

Secure Remote Storage

1. In the beginning is the Data

First Name	Last Name	Affiliation
Christian	Cachin	IBM
Giuseppe	Persiano	SAL
Ahmad-Reza	Sadeghi	TUD
Matthias	Schunter	IBM
Paulo	Verissimo	LIS

2. Encrypt and obtain

First Name	Last Name	Affiliation
E(PK, Christian)	E(PK, Cachin)	E(PK, IBM)
E(PK, Giuseppe)	E(PK, Persiano)	E(PK, SAL)
E(PK, Ahmad-Reza)	E(PK, Sadeghi)	E(PK, TUD)
E(PK, Matthias)	E(PK, Schunter)	E(PK, IBM)
E(PK, Paulo)	E(PK, Verissimo)	E(PK, LIS)

3. Authenticate by using MAC.

Secure Remote Storage

1. In the beginning is the Data

First Name	Last Name	Affiliation
Christian	Cachin	IBM
Giuseppe	Persiano	SAL
Ahmad-Reza	Sadeghi	TUD
Matthias	Schunter	IBM
Paulo	Verissimo	LIS

2. Encrypt and obtain

First Name	Last Name	Affiliation
E(PK, Christian)	E(PK, Cachin)	E(PK, IBM)
E(PK, Giuseppe)	E(PK, Persiano)	E(PK, SAL)
E(PK, Ahmad-Reza)	E(PK, Sadeghi)	E(PK, TUD)
E(PK, Matthias)	E(PK, Schunter)	E(PK, IBM)
E(PK, Paulo)	E(PK, Verissimo)	E(PK, LIS)

3. Authenticate by using MAC.
4. Disperse by using data replication algorithm.

Secure Remote Storage

1. In the beginning is the Data

First Name	Last Name	Affiliation
Christian	Cachin	IBM
Giuseppe	Persiano	SAL
Ahmad-Reza	Sadeghi	TUD
Matthias	Schunter	IBM
Paulo	Verissimo	LIS

2. Encrypt and obtain

First Name	Last Name	Affiliation
E(PK, Christian)	E(PK, Cachin)	E(PK, IBM)
E(PK, Giuseppe)	E(PK, Persiano)	E(PK, SAL)
E(PK, Ahmad-Reza)	E(PK, Sadeghi)	E(PK, TUD)
E(PK, Matthias)	E(PK, Schunter)	E(PK, IBM)
E(PK, Paulo)	E(PK, Verissimo)	E(PK, LIS)

3. Authenticate by using MAC.
4. Disperse by using data replication algorithm.

Caveat. For the Crypto-savvy, "Encrypt and Mac" has some subtleties.

Searching on data on a UStorage

Want all persons from IBM

1. Download the data using the retrieve algorithm;
2. Check it has not been modified;
3. Decrypt the whole table;
4. Execute the query;

Searching on data on a UStorage

Want all persons from IBM

1. Download the data using the retrieve algorithm;
2. Check it has not been modified;
3. Decrypt the whole table;
4. Execute the query;

Not really what we want

1. We need to store locally the table.

Searching on data on a UStorage

Want all persons from IBM

1. Download the data using the retrieve algorithm;
2. Check it has not been modified;
3. Decrypt the whole table;
4. Execute the query;

Not really what we want

1. We need to store locally the table.
2. We might not have enough local storage, that's why we resorted to the UStorage.

Searching on data on a UStorage

Want all persons from IBM

1. Download the data using the retrieve algorithm;
2. Check it has not been modified;
3. Decrypt the whole table;
4. Execute the query;

Not really what we want

1. We need to store locally the table.
2. We might not have enough local storage, that's why we resorted to the UStorage.
3. Question: can we ask the UStorage to perform the search for us?

Searching on data on a UStorage

Want all persons from IBM

1. Download the data using the retrieve algorithm;
2. Check it has not been modified;
3. Decrypt the whole table;
4. Execute the query;

Not really what we want

1. We need to store locally the table.
2. We might not have enough local storage, that's why we resorted to the UStorage.
3. Question: can we ask the UStorage to perform the search for us?
4. Answer 1: give UStorage the decryption query.

Searching on data on a UStorage

Want all persons from IBM

1. Download the data using the retrieve algorithm;
2. Check it has not been modified;
3. Decrypt the whole table;
4. Execute the query;

Not really what we want

1. We need to store locally the table.
2. We might not have enough local storage, that's why we resorted to the UStorage.
3. Question: can we ask the UStorage to perform the search for us?
4. Answer 1: give UStorage the decryption query. why did we encrypt?

Searching on data on a UStorage

Want all persons from IBM

1. Download the data using the retrieve algorithm;
2. Check it has not been modified;
3. Decrypt the whole table;
4. Execute the query;

Not really what we want

1. We need to store locally the table.
2. We might not have enough local storage, that's why we resorted to the UStorage.
3. Question: can we ask the UStorage to perform the search for us?
4. Answer 1: give UStorage the decryption query. why did we encrypt?
5. Answer 2: not with the current encryption schemes.

Predicate Encryption

Predicate Encryption for $\mathcal{P}$

- Ciphertexts and Keys have attributes.
- Key K with attribute $\vec{y}$ can decrypt ciphertext Ct with attribute $\vec{x}$ iff and only if $\mathcal{P}(\vec{x}, \vec{y}) = 1$.

Delegating decryption

1. Alice generates master secret key (MSK) and public key (PK') ;
2. Alice publishes PK';
3. Bob has a private message M to Alice;
   - Bob computes $E(\mathrm{PK}', M, \mathit{private})$;
4. Dean has a work message M' to Alice;
   - Dean computes $E(\mathrm{PK}', M', \mathit{work})$;
5. Alice gives key for work to secretary;
6. Alice keepts key for private for herself.

Searching encrypted data

Let $\mathcal{P}$ be a predicate such that

$$\mathcal{P}((FN, LN, A), (\star, \star, \text{“IBM”})) = 1 \text{ iff } A = \text{“IBM”}.$$

Searching encrypted data

Let $\mathcal{P}$ be a predicate such that

$$\mathcal{P}((FN, LN, A), (\star, \star, \text{"IBM"})) = 1 \text{ iff } A = \text{"IBM"}.$$

First Name	Last Name	Affiliation	Attributes
E(PK, Christian)	E(PK, Cachin)	E(PK, IBM)	E(PK',(C,C,I),0)
E(PK, Giuseppe)	E(PK, Persiano)	E(PK, SAL)	E(PK',(G,P,S),0)
E(PK, Ahmad-Reza)	E(PK, Sadeghi)	E(PK, TUD)	E(PK',(A,S,T),0)
E(PK, Matthias)	E(PK, Schunter)	E(PK, IBM)	E(PK',(M,S,I),0)
E(PK, Paulo)	E(PK, Verissimo)	E(PK, LIS)	E(PK',(P,V,L),0)

Predicate Encryption

The SELECT procedure

1. DOwner computes key K with attribute $(\star, \star, \mathit{IBM})$ and sends it to UStorage;
2. UStorage tries to decrypt $E(\mathsf{PK}', (\text{Christian,Cachin,IBM}), 0)$ with K and obtains 0;
   the row is selected
3. UStorage tries to decrypt $E(\mathsf{PK}', (\text{Giuseppe,Persiano,SAL}), 0)$ with K and obtains $\perp$;
   the row is not selected;
4. . . . .
5. UStorage sends the two selected rows to the DOwner;
6. DOwner decrypts the received rows;

Hidden Vector Encryption

Hidden Vector Encryption

- Ciphertext Ct is associated with *attribute* vector $\vec{x}$ of length ℓ over alphabet Σ.
- Key K is associated with *pattern* vector $\vec{y}$ of length ℓ over alphabet $\Sigma_\star = \Sigma \cup \{\star\}$.
- Predicate $\mathrm{Match}(\vec{x}, \vec{y})$ which is true if and only if $\vec{x} = \langle x_1, \ldots, x_\ell \rangle$ and $\vec{y} = \langle y_1, \ldots, y_\ell \rangle$ agree in all positions i for which $y_i \neq \star$.

Hidden Vector Encryption

Hidden Vector Encryption

- Ciphertext Ct is associated with *attribute* vector $\vec{x}$ of length ℓ over alphabet Σ.
- Key K is associated with *pattern* vector $\vec{y}$ of length ℓ over alphabet $\Sigma_\star = \Sigma \cup \{\star\}$.
- Predicate $\mathsf{Match}(\vec{x}, \vec{y})$ which is true if and only if $\vec{x} = \langle x_1, \ldots, x_\ell \rangle$ and $\vec{y} = \langle y_1, \ldots, y_\ell \rangle$ agree in all positions i for which $y_i \neq \star$.

If patterns vectors $\vec{y} \in \Sigma^\ell$ we have the original notion of searchable encryption.

Hidden Vector Encryption – The syntax

Hidden Vector Encryption (Attribute Only)

1. Setup$(1^n, 1^\ell)$ outputs the *public key* PK and the *secret key* SK.
2. Encryption$(\mathsf{PK}, \vec{x})$ outputs an *encrypted attribute vector* $\tilde{X}$.
3. GenToken$(\mathsf{SK}, \vec{y})$ outputs *key* $K_{\vec{y}}$.
4. Test$(\tilde{X}, T_{\vec{y}})$ returns Match$(\vec{x}, \vec{y})$ with overwhelming probability.

Semantic Security - Selective

$\mathsf{SemanticExp}_{\mathcal{A}}(1^n, 1^\ell)$

1. Initialization Phase. $\mathcal{A}$ announces two challenge attribute vectors $\vec{z}_0, \vec{z}_1 \in \Sigma^\ell$.
2. Key-Generation Phase. $\mathcal{C}$ computes $(\mathsf{PK}, \mathsf{SK}) \leftarrow \mathsf{Setup}(1^n, 1^\ell)$. PK is given to $\mathcal{A}$.
3. Query Phase I. $\mathcal{A}$ can make any number of key queries. $\mathcal{C}$ answers key queries only for patterns $\vec{y}$ such that $\mathsf{Match}(\vec{z}_0, \vec{y}) = \mathsf{Match}(\vec{z}_1, \vec{y}) = 0$.
4. Challenge construction. $\mathcal{C}$ chooses random $\eta \in \{0, 1\}$ and returns $\mathsf{Encryption}(\mathsf{PK}, \vec{z}_\eta)$ to $\mathcal{A}$.
5. Query Phase II. Identical to Query Phase I.
6. Output phase. $\mathcal{A}$ returns η'. If $\eta = \eta'$ then the experiments returns 1 else 0.

Known Constructions

Pairing (symmetric version)

- multiplicative groups $\mathbb{G}$ and $\mathbb{G}_T$ of order p;
- non-degenerate pairing function $\mathsf{e} : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$;
  - for all $x \in \mathbb{G}$, $x \neq 1$, and $a, b \in \mathbb{Z}_p$,

$$\mathsf{e}(x, x) \neq 1 \text{ and } \mathsf{e}(x^a, x^b) = \mathsf{e}(x, x)^{ab}.$$

Known Constructions

Pairing (symmetric version)

- multiplicative groups $\mathbb{G}$ and $\mathbb{G}_T$ of order p;
- non-degenerate pairing function $\mathsf{e} : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$;
  - for all $x \in \mathbb{G}$, $x \neq 1$, and $a, b \in \mathbb{Z}_p$,

$$\mathsf{e}(x, x) \neq 1 \text{ and } \mathsf{e}(x^a, x^b) = \mathsf{e}(x, x)^{ab}.$$

Constructions

- Boneh and Waters [TCC 07] gave a construction based on complexity assumption for pairing with composite order group;
- Iovino and P. [Pairing 08] gave a construction for **prime** order groups;

BH needs about 1024-bit moduli.
For IP we can use 160-bit moduli.

A Construction for HVE

Hidden Vector Encryption [IP08]

1 Setup$(1^n, 1^\ell)$ outputs
Pick an instance $\mathcal{I}$ with n-bit prime and random $t_{i,b}, v_{i,b} \in \mathbb{Z}_p$ for $i \in [\ell]$ and $b \in \Sigma$.

$$\begin{aligned}\mathsf{PK} &= [\mathcal{I}, (T_{i,b} = g^{t_{i,b}}, V_{i,b} = g^{v_{i,b}})_{i\in[\ell], b\in\Sigma}] \\ \mathsf{SK} &= [\mathcal{I}, (\hat{T}_{i,b} = g^{1/t_{i,b}}, \hat{V}_{i,b} = g^{1/v_{i,b}})_{i\in[\ell], b\in\Sigma}]\end{aligned}$$

2 Encryption$(\mathsf{PK}, \vec{x})$ for $\vec{x} = \langle x_1, \ldots, x_\ell \rangle$ outputs $\tilde{X} = (X_i, W_i)_{i=1}^{\ell}$ where

$$X_i = T_{i,x_i}^{s-s_i} \qquad W_i = V_{i,x_i}^{s_i}$$

for randomly chosen $s, s_1, \ldots, s_\ell \in \mathbb{Z}_p$.

A Construction for HVE

Hidden Vector Encryption IP08

3 GenToken(SK, $\vec{y}$) outputs *key* $K_{\vec{y}} = (Y_i, L_i)_{i=1}^{\ell}$ where

$$Y_i = \begin{cases} \hat{T}_{i,y_i}^{a_i}, & \text{if } \ y_i \neq \star; \\ \emptyset, & \text{if } \ y_i = \star. \end{cases} \quad \text{and} \quad L_i = \begin{cases} \hat{V}_{i,y_i}^{a_i}, & \text{if } \ y_i \neq \star; \\ \emptyset, & \text{if } \ y_i = \star. \end{cases}$$

where for i such that $y_i \neq \star$, the a_i's are random under the constraint that $\sum_i a_i = 0$.

Construction

Test

4

$$\text{Test}(\tilde{X}, K_{\vec{y}}) = \prod_{i:y_i \neq \star} e(X_i, Y_i) \cdot (W_i, L_i).$$

Observation:

$$x_i = y_i \Rightarrow e(T_{i,x_i}, \hat{T}_{i,y_i}) = e(g, g).$$

$$x_i = y_i \Rightarrow e(V_{i,b}, \hat{V}_{i,b}) = e(g, g).$$

Implementation

Implementation uses:

1. PBC: Pairing Based Cryptography Library `http://crypto.stanford.edu/pbc/` for basic pairing and elliptic curves computation. Written in C.
2. jPBC: Java Pairing Based Cryptography Library `http://gas.dia.unisa.it/projects/jpbc/`
    1. a Java Porting of the PBC library;
    2. a Java Wrapper of the PBC library;

Three versions tested:

1. jPBC: uses the the Java porting of the PBC library;
2. jPBC+precomputation: uses the the Java porting of the PBC library but with precomputation;
3. jPBC+PBC+precomputation: uses the Java Wrapper (low level computation delegated to more efficient PBC C code) and precomputation.

Parameters

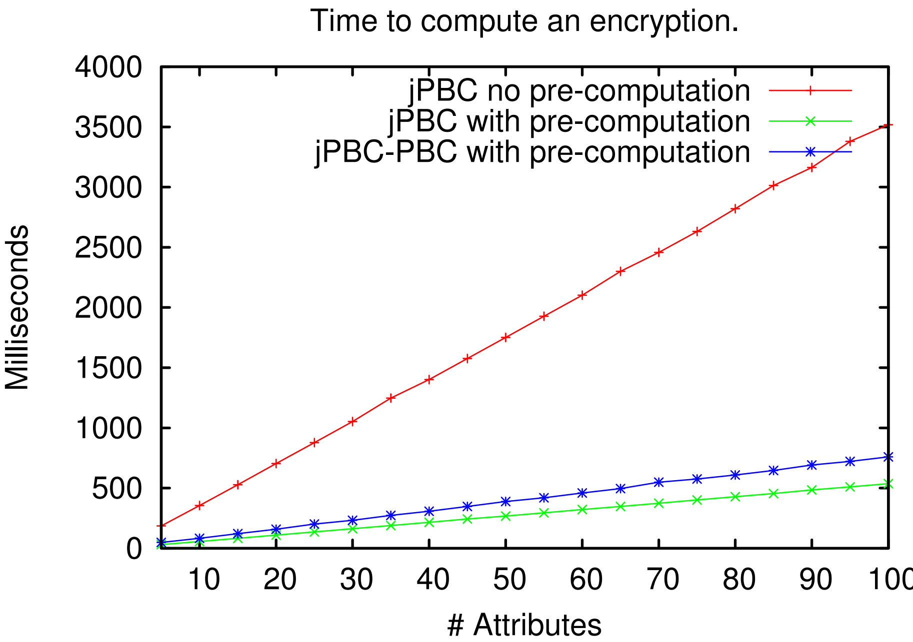
Curve

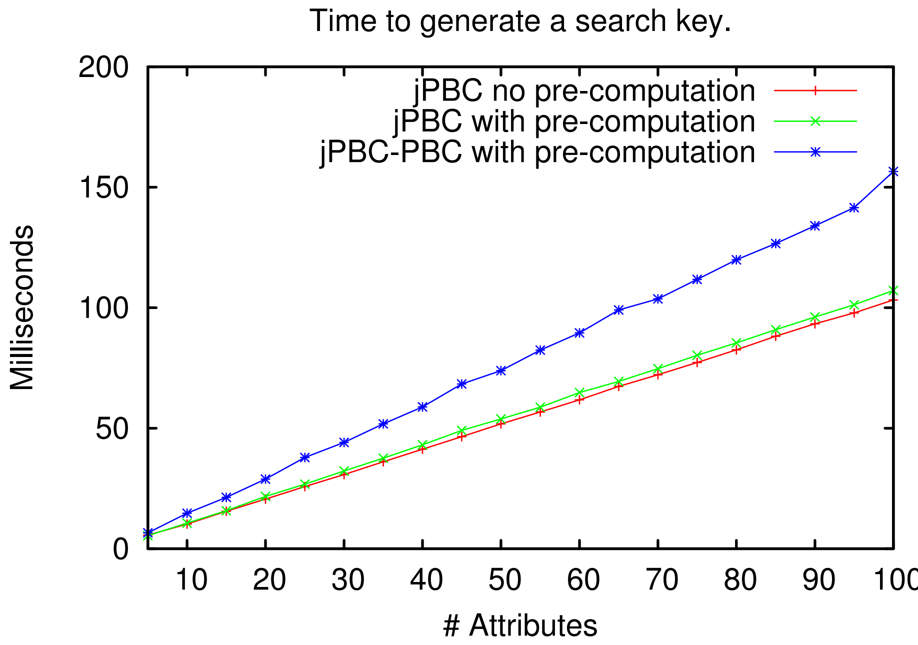
- Supersingular curve $y^2 = x^3 + x$ over the field F_q for some prime $q = 3 \mod 4$. (Type A symmetric pairings)
- The order p is a prime factor of $q + 1$.

$$
\begin{aligned}
q &= 1112516189738354695660623681779709216838322823798404116 \\
& 19891970830748504680026008670522117985647539911425452 \\
& 4050414866145727834858675222143950902758166111 \\
& \text{512 bit} \\
r &= 730750818665451459101842416358141509827966795777 \\
& \text{160 bit}
\end{aligned}
$$

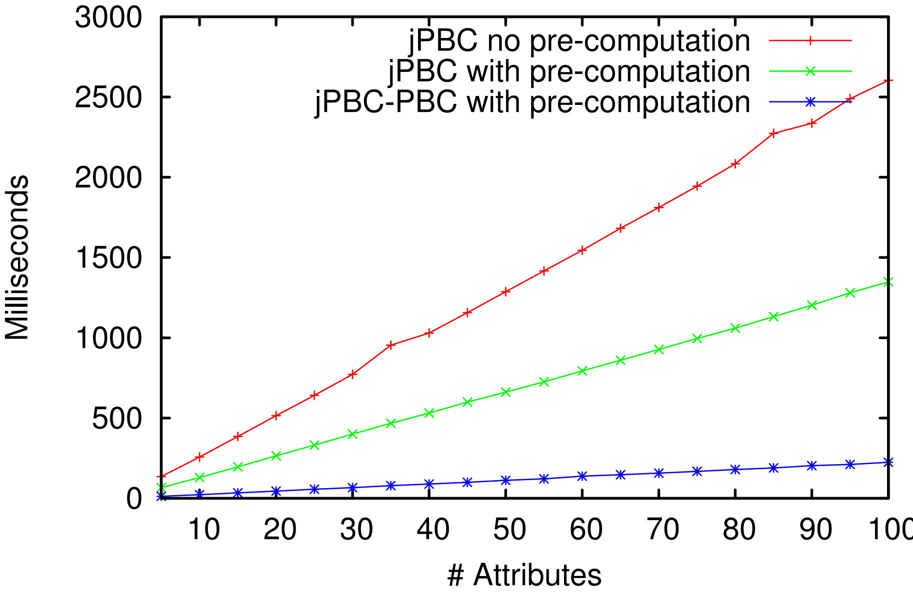
Experimental setup

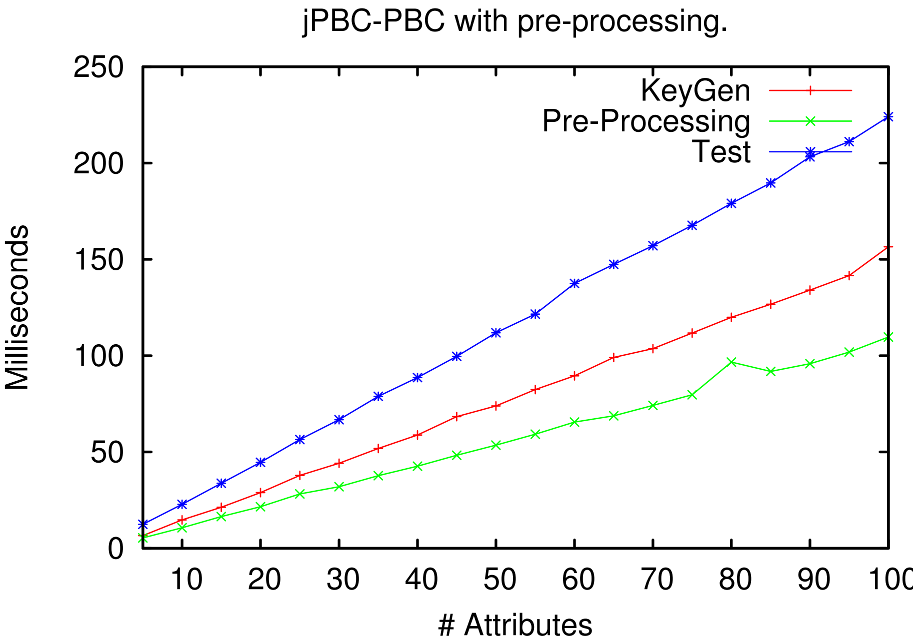
Model Name: iMac
Model Identifier: iMac8.1
Processor Name: Intel Core 2 Duo
Processor Speed: 2.66 GHz
Number Of Processors: 1
Total Number of Cores: 2
L2 Cache: 6 MB
Memory: 4GB
Bus Speed: 1.07 GHz


Time to compute an encryption.
jPBC no pre-computation
jPBC with pre-computation
jPBC-PBC with pre-computation
Milliseconds
Attributes
4000
3500
3000
2500
2000
1500
1000
500
0
10
20
30
40
50
60
70
80
90
100


Time to generate a search key.
jPBC no pre-computation
jPBC with pre-computation
jPBC-PBC with pre-computation
Milliseconds
Attributes
0
50
100
150
200
10
20
30
40
50
60
70
80
90
100

Time to test a ciphertext against a search key.




jPBC-PBC with pre-processing.
KeyGen
Pre-Processing
Test
Milliseconds
Attributes
250
200
150
100
50
0
10
20
30
40
50
60
70
80
90
100

Future work

Current implementation by Angelo De Caro.

- more code optimization
- Dropbox-like user interfact
- Map-Reduce
- different types of pairings
  - the scheme can be implemented with asymmetric pairing.

Back to Theory

Full Security

1. Key-Generation Phase. $\mathcal{C}$ computes $(\mathsf{PK}, \mathsf{SK}) \leftarrow \mathsf{Setup}(1^n, 1^\ell)$. PK is given to $\mathcal{A}$.
2. Query Phase I. $\mathcal{A}$ can make any number of key queries.
3. Initialization Phase. $\mathcal{A}$ announces two challenge attribute vectors $\vec{z}_0, \vec{z}_1 \in \Sigma^\ell$.
4. Challenge construction. $\mathcal{C}$ chooses random $\eta \in \{0, 1\}$ and returns $\mathsf{Encryption}(\mathsf{PK}, \vec{z}_\eta)$ to $\mathcal{A}$.
5. Query Phase II. Identical to Query Phase I.
6. Output phase. $\mathcal{A}$ returns η'.
7. Winning condition. $\mathcal{A}$ wins if $\eta = \eta'$.

Restricting the queries

Impossible to achieve

If $\mathcal{A}$ has asked a key for $\vec{y}$ such that $\mathsf{Match}(\vec{z}_0, \vec{y}) \neq \mathsf{Match}(\vec{z}_1, \vec{y})$, $\mathcal{A}$ wins.

Restricting the queries

Impossible to achieve

If $\mathcal{A}$ has asked a key for $\vec{y}$ such that $\mathrm{Match}(\vec{z}_0, \vec{y}) \neq \mathrm{Match}(\vec{z}_1, \vec{y})$, $\mathcal{A}$ wins.

Unrestricted queries

Winning condition. $\mathcal{A}$ wins if $\eta = \eta'$ and for all $\vec{y}$ for which $\mathcal{A}$ has a key

$$\mathrm{Match}(\vec{z}_0, \vec{y}) = \mathrm{Match}(\vec{z}_1, \vec{y})$$

Restricting the queries

Impossible to achieve

If $\mathcal{A}$ has asked a key for $\vec{y}$ such that $\text{Match}(\vec{z}_0, \vec{y}) \neq \text{Match}(\vec{z}_1, \vec{y})$, $\mathcal{A}$ wins.

Unrestricted queries

Winning condition. $\mathcal{A}$ wins if $\eta = \eta'$ and for all $\vec{y}$ for which $\mathcal{A}$ has a key

$$\text{Match}(\vec{z}_0, \vec{y}) = \text{Match}(\vec{z}_1, \vec{y})$$

Restricted queries

Winning condition. $\mathcal{A}$ wins if $\eta = \eta'$ and for all $\vec{y}$ for which $\mathcal{A}$ has a key

$$\text{Match}(\vec{z}_0, \vec{y}) = \text{Match}(\vec{z}_1, \vec{y}) = 0$$

Full Security with Unrestricted Queries

1. Key-Generation Phase. $\mathcal{C}$ computes $(\mathsf{PK}, \mathsf{SK}) \leftarrow \mathsf{Setup}(1^n, 1^\ell)$. PK is given to $\mathcal{A}$.
2. Query Phase I. $\mathcal{A}$ can make any number of key queries.
3. Initialization Phase. $\mathcal{A}$ announces two challenge attribute vectors $\vec{z}_0, \vec{z}_1 \in \Sigma^\ell$.
4. Challenge construction. $\mathcal{C}$ chooses random $\eta \in \{0, 1\}$ and returns $\mathsf{Encryption}(\mathsf{PK}, \vec{z}_\eta)$ to $\mathcal{A}$.
5. Query Phase II. Identical to Query Phase I.
6. Output phase. $\mathcal{A}$ returns η'.
7. Winning condition. $\mathcal{A}$ wins if $\eta = \eta'$ **and** for all queries $\vec{y}$ it holds that

$$\mathsf{Match}(\vec{z}_0, \vec{y}) = \mathsf{Match}(\vec{z}_1, \vec{y})$$

Full Security

De Caro, Iovino, P, 2011

Fully secure HVE with unrestricted queries in composite (product of 4 primes) order bilinear groups.

Caveat: 160 bits become 2048.

Selective vs Full Security

- Selective security assumes that the adversary attacks the **data** and not the public key.
- The adversary declares that he wants to distinguish ciphertexts with Affiliation=IBM from ciphertexts with Affiliation=SAL.
- Full security allows the adversary to base his attack on the public key (which is chosen independently from the data) and on the keys obtained.

Key Security

Security threat

UStorage knows all searches I have done.

Key Security

Security threat

UStorage knows all searches I have done.

Key security is impossible for public key

- storage manager receives $K_{\vec{y}}$ and wants to check if $\mathrm{Match}(\langle 1, \ldots, 1 \rangle, \vec{y}) = 1$;
  - encrypt $\langle 1, \ldots, 1 \rangle$ using PK;
  - run Test to obtain the answer;

We should go private key!

Key Security

Security threat

UStorage knows all searches I have done.

Key security is impossible for public key

- storage manager receives $K_{\vec{y}}$ and wants to check if $\mathrm{Match}(\langle 1, \ldots, 1\rangle, \vec{y}) = 1$;
  - encrypt $\langle 1, \ldots, 1\rangle$ using PK;
  - run Test to obtain the answer;

We should go private key! **Or maybe not....**

Public vs. Private Key

Why Public Key?

- In the scenario with DBowner and UStorage, private key is **sufficient**.
- All write operations must go through the DBowner.
- In the Alice/Secretary settings we need public key.

Partial Public Key Model

Key Policy

- public keys associated with policies;
- a policy Pol is a vector of subsets of Σ;
- it encodes the set $\mathbb{X}_{\mathsf{Pol}}$ of attribute vectors that can be encrypted;

Partial Public Key Model

Key Policy

- public keys associated with policies;
- a policy Pol is a vector of subsets of Σ;
- it encodes the set $\mathbb{X}_{\mathsf{Pol}}$ of attribute vectors that can be encrypted;
  - for $\ell = 3$ and $\Sigma = \{0, 1\}$
    $\mathsf{Pol} = \langle \{0, 1\}, \{0\}, \{0, 1\} \rangle$

Partial Public Key Model

Key Policy

- public keys associated with policies;
- a policy Pol is a vector of subsets of Σ;
- it encodes the set $\mathbb{X}_{\mathsf{Pol}}$ of attribute vectors that can be encrypted;
  - for $\ell = 3$ and $\Sigma = \{0, 1\}$
    $\mathsf{Pol} = \langle\{0, 1\}, \{0\}, \{0, 1\}\rangle \Rightarrow$ vectors with a 0 entry in position 2;

Partial Public Key Model

Key Policy

- public keys associated with policies;
- a policy Pol is a vector of subsets of Σ;
- it encodes the set $\mathbb{X}_{\mathsf{Pol}}$ of attribute vectors that can be encrypted;
  - for $\ell = 3$ and $\Sigma = \{0, 1\}$
    $\mathsf{Pol} = \langle \{0, 1\}, \{0\}, \{0, 1\} \rangle \Rightarrow$ vectors with a 0 entry in position 2;
  - $\mathsf{Pol} = \Sigma^{\ell}$

Partial Public Key Model

Key Policy

- public keys associated with policies;
- a policy Pol is a vector of subsets of Σ;
- it encodes the set $\mathbb{X}_{\mathsf{Pol}}$ of attribute vectors that can be encrypted;
  - for $\ell = 3$ and $\Sigma = \{0, 1\}$
    $\mathsf{Pol} = \langle\{0, 1\}, \{0\}, \{0, 1\}\rangle \Rightarrow$ vectors with a 0 entry in position 2;
  - $\mathsf{Pol} = \Sigma^{\ell} \Rightarrow$ all vectors (public-key setting);

Partial Public Key Model

Key Policy

- public keys associated with policies;
- a policy Pol is a vector of subsets of Σ;
- it encodes the set $\mathbb{X}_{\mathsf{Pol}}$ of attribute vectors that can be encrypted;
  - for $\ell = 3$ and $\Sigma = \{0, 1\}$
    $\mathsf{Pol} = \langle \{0, 1\}, \{0\}, \{0, 1\} \rangle \Rightarrow$ vectors with a 0 entry in position 2;
  - $\mathsf{Pol} = \Sigma^\ell \Rightarrow$ all vectors (public-key setting);
  - if any entry is $\emptyset$

Partial Public Key Model

Key Policy

- public keys associated with policies;
- a policy Pol is a vector of subsets of Σ;
- it encodes the set $\mathbb{X}_{\mathsf{Pol}}$ of attribute vectors that can be encrypted;
  - for $\ell = 3$ and $\Sigma = \{0, 1\}$
    $\mathsf{Pol} = \langle\{0,1\},\{0\},\{0,1\}\rangle \Rightarrow$ vectors with a 0 entry in position 2;
  - $\mathsf{Pol} = \Sigma^{\ell} \Rightarrow$ all vectors (public-key setting);
  - if any entry is $\emptyset \Rightarrow$ no vector (private-key setting);

Partial Public Key Model

Key Policy

- public keys associated with policies;
- a policy Pol is a vector of subsets of Σ;
- it encodes the set $\mathbb{X}_{\mathsf{Pol}}$ of attribute vectors that can be encrypted;
  - for $\ell = 3$ and $\Sigma = \{0, 1\}$
    $\mathsf{Pol} = \langle\{0,1\},\{0\},\{0,1\}\rangle \Rightarrow$ vectors with a 0 entry in position 2;
  - $\mathsf{Pol} = \Sigma^\ell \Rightarrow$ all vectors (public-key setting);
  - if any entry is $\emptyset \Rightarrow$ no vector (private-key setting);

Hidden Vector Encryption

1. Setup$(1^n, 1^\ell)$ outputs the *secret key* SK.
2. PPKeyGen(SK, Pol) outputs the *partial public key* $\mathsf{PPK}_{\mathsf{Pol}}$.
3. Encryption$(\mathsf{PPK}_{\mathsf{Pol}}, \vec{x})$ outputs *encrypted attribute vector* $\tilde{X}$ for *attribute vector* $\vec{x} \in \mathbb{X}_{\mathsf{Pol}}$.
4. GenToken$(\mathsf{SK}, \vec{y})$ outputs *key* $K_{\vec{y}}$.
5. Test$(\tilde{X}, T_{\vec{y}})$ returns Match$(\vec{x}, \vec{y})$ with overwhelming probability.

Known Constructions

Constructions

- Boneh-Waters [2007] gave a construction based on groups with order product of four primes. Need 2048-bit moduli.

- Blundo, Iovino, P. [2009, 2010] gave a construction based on groups of prime order.

Finish

Semantic Security with Partial Public Keys

1. Initialization Phase. $\mathcal{A}$ announces two challenge attribute vectors $\vec{z}_0, \vec{z}_1 \in \Sigma^\ell$ and policy $\mathsf{Pol} \in (2^\Sigma)^\ell$.
2. Key-Generation Phase. $\mathcal{C}$ computes $\mathsf{SK} \leftarrow \mathsf{Setup}(1^n, 1^\ell)$ and $\mathsf{PPK}_{\mathsf{Pol}} \leftarrow \mathsf{PPKeyGen}(\mathsf{SK}, \mathsf{Pol})$.
   $\mathsf{PPK}_{\mathsf{Pol}}$ is given to $\mathcal{A}$.
3. Query Phase I. $\mathcal{A}$ can make any number of key queries.
   $\mathcal{C}$ answers key queries only for patterns $\vec{y}$ such that $\mathsf{Match}(\vec{z}_0, \vec{y}) = \mathsf{Match}(\vec{z}_1, \vec{y}) = 0$.
4. Challenge construction. $\mathcal{C}$ chooses random $\eta \in \{0, 1\}$ and returns $\mathsf{Encryption}(\mathsf{SK}, \vec{z}_\eta)$.
5. Query Phase II. Identical to Query Phase I.
6. Output phase. $\mathcal{A}$ returns η'.
   If $\eta = \eta'$ then the experiments returns 1 else 0.

Token Security with Partial Public Keys

1. Initialization Phase. $\mathcal{A}$ announces $\vec{y}_0, \vec{y}_1 \in \Sigma_\star^\ell$ **with $\star$ in the same positions** and a policy Pol such that
$$\vec{x} \in \mathbb{X}_{\mathsf{Pol}} \Rightarrow \mathsf{Match}(\vec{x}, \vec{y}_0) = \mathsf{Match}(\vec{x}, \vec{y}_1) = 0.$$
2. Key-Generation Phase. $\mathcal{C}$ computes $\mathsf{SK} \leftarrow \mathsf{Setup}(1^n, 1^\ell)$ and $\mathsf{PPK}_{\mathsf{Pol}} \leftarrow \mathsf{PPKeyGen}(\mathsf{SK}, \mathsf{Pol})$.
   $\mathsf{PPK}_{\mathsf{Pol}}$ is given to $\mathcal{A}$.
3. Query Phase I. $\mathcal{A}$ can make any number of key queries.
   $\mathcal{A}$ gets $\mathsf{GenToken}(\mathsf{SK}, \vec{y})$.
4. Challenge construction. η is chosen at random from $\{0, 1\}$ and receives $\mathsf{GenToken}(\mathsf{SK}, \vec{y}_\eta)$.
5. Query Phase II. Identical to Query Phase I.
6. Output phase. $\mathcal{A}$ returns η'.
   If $\eta = \eta'$ then the experiments returns 1 else 0.

Construction for Partial Public Key HVE

Setup $(1^n, 1^\ell)$

1. Select a symmetric bilinear instance $\mathcal{I} = [p, \mathbb{G}, \mathbb{G}_T, g, \mathsf{e}]$.
2. For $i \in [2\ell - 1]$, choose random $t_{1,i,0}, t_{2,i,0}, t_{1,i,1}, t_{2,i,1} \in \mathbb{Z}_p$ and set

$$\begin{aligned}
\mathsf{K}_i &= \begin{pmatrix} T_{1,i,0} = g^{t_{1,i,0}}, & T_{2,i,0} = g^{t_{2,i,0}} \\ T_{1,i,1} = g^{t_{1,i,1}}, & T_{2,i,1} = g^{t_{2,i,1}} \end{pmatrix} \\
\bar{\mathsf{K}}_i &= \begin{pmatrix} \bar{T}_{1,i,0} = g^{1/t_{1,i,0}}, & \bar{T}_{2,i,0} = g^{1/t_{2,i,0}} \\ \bar{T}_{1,i,1} = g^{1/t_{1,i,1}}, & \bar{T}_{2,i,1} = g^{1/t_{2,i,1}} \end{pmatrix}.
\end{aligned}$$

3. Return $\mathsf{SK} = [\mathcal{I}, (\mathsf{K}_i, \bar{\mathsf{K}}_i)_{i \in [2\ell-1]}]$.

Construction for Partial Public Key HVE

Setup $(1^n, 1^\ell)$

1. Select a symmetric bilinear instance $\mathcal{I} = [p, \mathbb{G}, \mathbb{G}_T, g, \mathsf{e}]$.
2. For $i \in [2\ell - 1]$, choose random $t_{1,i,0}, t_{2,i,0}, t_{1,i,1}, t_{2,i,1} \in \mathbb{Z}_p$ and set

$$\begin{array}{rcl} \mathsf{K}_i & = & \left(\begin{array}{ll} T_{1,i,0} = g^{t_{1,i,0}}, & T_{2,i,0} = g^{t_{2,i,0}} \\ T_{1,i,1} = g^{t_{1,i,1}}, & T_{2,i,1} = g^{t_{2,i,1}} \end{array} \right) \\ \bar{\mathsf{K}}_i & = & \left(\begin{array}{ll} \bar{T}_{1,i,0} = g^{1/t_{1,i,0}}, & \bar{T}_{2,i,0} = g^{1/t_{2,i,0}} \\ \bar{T}_{1,i,1} = g^{1/t_{1,i,1}}, & \bar{T}_{2,i,1} = g^{1/t_{2,i,1}} \end{array} \right). \end{array}$$

3. Return $\mathsf{SK} = [\mathcal{I}, (\mathsf{K}_i, \bar{\mathsf{K}}_i)_{i \in [2\ell-1]}]$.

Notice: if $x = y$ then

$$\mathsf{e}(T_{b,i,x}, \bar{T}_{b,i,y}) = \mathsf{e}(g, g)$$

for all $i \in [2\ell - 1]$ and $b = 1, 2$.

Construction for Partial Public Key HVE

PPKeyGen $(\mathsf{SK}, \mathsf{Pol})$

1. For $i = 1, \ldots, \ell$,
   for every $b \in \mathsf{Pol}_i$, add $T_{1,i,b}$ and $T_{2,i,b}$ to PPK_i.
2. For $i = \ell + 1, \ldots, 2\ell - 1$,
   add $T_{1,i,0}$ and $T_{2,i,0}$ to PPK_i.
3. Return $\mathsf{PPK}_{\mathsf{Pol}} = [(\mathsf{PPK}_i)_{i \in [2\ell-1]}]$.

Construction for Partial Public Key HVE

Encryption$(\mathsf{PPK}_{\mathsf{Pol}}, \vec{x} = \langle x_1, \ldots, x_\ell \rangle)$

1. If $\vec{x} \notin \mathbb{X}_{\mathsf{Pol}}$ return $\perp$.
2. Append $(\ell - 1)$ 0-entries to $\vec{x}$.
3. Pick s at random from $\mathbb{Z}_p$.
4. $(s_1, \ldots, s_{2\ell-1}) \leftarrow \mathsf{LSS}(\ell, 2\ell - 1, 0)$.
5. For $i = 1, \ldots, 2\ell - 1$,
   set $X_{1,i} = T_{1,i,x_i}^{s-s_i}$ and $X_{2,i} = T_{2,i,x_i}^{-s_i}$.
6. Return $\tilde{X} = [(X_{1,i}, X_{2,i})_{i \in [2\ell-1]}]$.

Construction for Partial Public Key HVE

Encryption$(\mathsf{PPK}_{\mathsf{Pol}}, \vec{x} = \langle x_1, \ldots, x_\ell \rangle)$

1. If $\vec{x} \notin \mathbb{X}_{\mathsf{Pol}}$ return $\perp$.
2. Append $(\ell - 1)$ 0-entries to $\vec{x}$.
3. Pick s at random from $\mathbb{Z}_p$.
4. $(s_1, \ldots, s_{2\ell-1}) \leftarrow \mathsf{LSS}(\ell, 2\ell - 1, 0)$.
5. For $i = 1, \ldots, 2\ell - 1$,
   set $X_{1,i} = T_{1,i,x_i}^{s-s_i}$ and $X_{2,i} = T_{2,i,x_i}^{-s_i}$.
6. Return $\tilde{X} = [(X_{1,i}, X_{2,i})_{i \in [2\ell-1]}]$.

Notice: if $\vec{x} \in \mathbb{X}_{\mathsf{Pol}}$, then $\forall i \;\; T_{1,i,x_i}, T_{2,i,x_i} \in \mathsf{PPK}_{\mathsf{Pol}}$.

Linear Secret Sharing

(k, n) Linear Secret Sharing

- **Input:** a *secret* $s \in \mathbb{Z}_p$;
- **Output:** *n shares* $(s_1, \ldots, s_n)$ such that
  - any $k - 1$ (or fewer) shares are random and independent among themselves and are independent from the secret s;
  - for any $F \subseteq [n]$ of size k there exist reconstruction coefficients α_i such that

$$s = \sum_{i \in F} \alpha_i s_i.$$

Linear Secret Sharing

(k, n) Linear Secret Sharing

- **Input:** a *secret* $s \in \mathbb{Z}_p$;
- **Output:** n *shares* $(s_1, \ldots, s_n)$ such that
  - any $k-1$ (or fewer) shares are random and independent among themselves and are independent from the secret s;
  - for any $F \subseteq [n]$ of size k there exist reconstruction coefficients α_i such that

$$s = \sum_{i \in F} \alpha_i s_i.$$

Notice: the reconstruction coefficients depend only on the set F and not on the shares.

Construction for Partial Public Key HVE

GenToken $(\mathsf{SK}, \vec{y} = \langle y_1, \ldots, y_\ell \rangle)$

1. Pick random $r \in \mathbb{Z}_p$.
2. $h = \#$ of non-$\star$ entries of $\vec{y}$.
   append $(\ell - h)$ 0-entries and $(h-1)$ $\star$-entries
   $S_{\vec{y}}$ the non-$\star$ entries of the extended vector.
   Notice that $|S_{\vec{y}}| = \ell$.
3. $(r_1, \ldots, r_{2\ell-1}) \leftarrow \mathsf{LSS}(\ell, 2\ell-1, 0)$.
4. For $i \in S_{\vec{y}}$,
   set $Y_{1,i} = \bar{T}_{1,i,y_i}^{r_i}$ and $Y_{2,i} = \bar{T}_{2,i,y_i}^{r-r_i}$.
5. Return $T_{\vec{y}} = [S_{\vec{y}}, (Y_{1,i}, Y_{2,i})_{i \in S_{\vec{y}}}]$.

Construction for Partial Public Key HVE

Test $(\tilde{X} = [(X_{1,i}, X_{2,i})_{i \in [2\ell-1]}], T_{\vec{y}} = [S, (Y_{1,i}, Y_{2,i})_{i \in S}])$

1. Let $(v_i)_{i \in S}$ be the reconstruction coefficients for S.
2. Return 1 iff
$$\prod_{i \in S} [\mathrm{e}(X_{1,i}, Y_{1,i}) \cdot \mathrm{e}(X_{2,i}, Y_{2,i})]^{v_i} = 1$$

Construction for Partial Public Key HVE

Test $(\tilde{X} = [(X_{1,i}, X_{2,i})_{i \in [2\ell-1]}], T_{\vec{y}} = [S, (Y_{1,i}, Y_{2,i})_{i \in S}])$

1. Let $(v_i)_{i \in S}$ be the reconstruction coefficients for S.
2. Return 1 iff
$$\prod_{i \in S} [\mathrm{e}(X_{1,i}, Y_{1,i}) \cdot \mathrm{e}(X_{2,i}, Y_{2,i})]^{v_i} = 1$$

$$\begin{array}{lclcl} \mathrm{e}(X_{1,i}, Y_{1,i}) & = & \mathrm{e}(T_{1,i,x_i}^{s-s_i}, \bar{T}_{1,i,y_i}^{r_i}) & = & \mathrm{e}(g,g)^{r_i(s-s_i)} \\ \mathrm{e}(X_{2,i}, Y_{2,i}) & = & \mathrm{e}(T_{2,i,x_i}^{-s_i}, \bar{T}_{2,i,y_i}^{r-r_i}) & = & \mathrm{e}(g,g)^{-s_i(r-r_i)} \end{array}$$

Construction for Partial Public Key HVE

Test $(\tilde{X} = [(X_{1,i}, X_{2,i})_{i \in [2\ell-1]}], T_{\vec{y}} = [S, (Y_{1,i}, Y_{2,i})_{i \in S}])$

1. Let $(v_i)_{i \in S}$ be the reconstruction coefficients for S.
2. Return 1 iff
$$\prod_{i \in S} [\mathrm{e}(X_{1,i}, Y_{1,i}) \cdot \mathrm{e}(X_{2,i}, Y_{2,i})]^{v_i} = 1$$

$$\begin{array}{rcccl} \mathrm{e}(X_{1,i}, Y_{1,i}) & = & \mathrm{e}(T_{1,i,x_i}^{s-s_i}, \bar{T}_{1,i,y_i}^{r_i}) & = & \mathrm{e}(g,g)^{r_i(s-s_i)} \\ \mathrm{e}(X_{2,i}, Y_{2,i}) & = & \mathrm{e}(T_{2,i,x_i}^{-s_i}, \bar{T}_{2,i,y_i}^{r-r_i}) & = & \mathrm{e}(g,g)^{-s_i(r-r_i)} \end{array}$$

$$\mathrm{e}(X_{1,i}, Y_{1,i}) \cdot \mathrm{e}(X_{2,i}, Y_{2,i}) = \mathrm{e}(g,g)^{sr_i} \cdot \mathrm{e}(g,g)^{-rs_i}$$

Construction for Partial Public Key HVE

Test $(\tilde{X} = [(X_{1,i}, X_{2,i})_{i\in[2\ell-1]}], T_{\vec{y}} = [S, (Y_{1,i}, Y_{2,i})_{i\in S}])$

1. Let $(v_i)_{i\in S}$ be the reconstruction coefficients for S.
2. Return 1 iff
$$\prod_{i\in S}[\mathrm{e}(X_{1,i}, Y_{1,i}) \cdot \mathrm{e}(X_{2,i}, Y_{2,i})]^{v_i} = 1$$

$$\begin{array}{lcccc}
\mathrm{e}(X_{1,i}, Y_{1,i}) & = & \mathrm{e}(T^{s-s_i}_{1,i,x_i}, \bar{T}^{r_i}_{1,i,y_i}) & = & \mathrm{e}(g,g)^{r_i(s-s_i)} \\
\mathrm{e}(X_{2,i}, Y_{2,i}) & = & \mathrm{e}(T^{-s_i}_{2,i,x_i}, \bar{T}^{r-r_i}_{2,i,y_i}) & = & \mathrm{e}(g,g)^{-s_i(r-r_i)} \\
 & & & & \\
\mathrm{e}(X_{1,i}, Y_{1,i}) & \cdot & \mathrm{e}(X_{2,i}, Y_{2,i}) & = & \mathrm{e}(g,g)^{sr_i} \cdot \mathrm{e}(g,g)^{-rs_i} \\
 & & & & \\
\mathrm{e}(g,g)^{s\sum_i r_i v_i} & \cdot & \mathrm{e}(g,g)^{-r\sum_i s_i v_i} & = & \mathrm{e}(g,g)^{s\cdot 0} \cdot \mathrm{e}(g,g)^{-r\cdot 0}
\end{array}$$

Security proofs

Security proofs

Can prove semantic and key security on complexity assumptions

Private-Key Searchable Encryption – The syntax

Private-Key Searchable Encryption

1. Setup$(1^n, 1^\ell)$ outputs the *secret key* SK.
2. Encryption$(\mathsf{SK}, \vec{x})$ outputs ciphertext $\mathsf{Ct}_{\vec{x}}$ with attribute $\vec{x} \in \Sigma^\ell$.
3. GenToken$(\mathsf{SK}, \vec{y})$ outputs *key* $K_{\vec{y}}$ for pattern $\vec{y} \in \Sigma^\ell$.
4. Test$(\mathsf{Ct}_{\vec{x}}, K_{\vec{y}})$ returns 1 iff $\vec{x} = \vec{y}$.

Semantic Security with Private Keys

1. Initialization Phase. $\mathcal{A}$ announces two challenge attribute vectors $\vec{x}_0, \vec{x}_1 \in \Sigma^\ell$.
2. Key-Generation Phase. $\mathcal{C}$ computes $\mathsf{SK} \leftarrow \mathsf{Setup}(1^n, 1^\ell)$.
3. Query Phase I. $\mathcal{A}$ can make any number of encryption and key queries for patterns $\vec{y} \neq \vec{x}_0, \vec{x}_1$.
4. Challenge construction. $\mathcal{C}$ chooses random $\eta \in \{0, 1\}$ and returns $\mathsf{Encryption}(\mathsf{SK}, \vec{x}_\eta)$.
5. Query Phase II. Identical to Query Phase I.
6. Output phase. $\mathcal{A}$ returns η'.
   If $\eta = \eta'$ then the experiments returns 1 else 0.

Token Security with Private Keys

1. Initialization Phase. $\mathcal{A}$ announces $\vec{y}_0, \vec{y}_1 \in \Sigma^\ell$.
2. Key-Generation Phase. $\mathcal{C}$ computes $\mathsf{SK} \leftarrow \mathsf{Setup}(1^n, 1^\ell)$.
3. Query Phase I. $\mathcal{A}$ can make any number of key queries and encryption queries for attributes $\vec{x} \neq \vec{y}_0, \vec{y}_1$.
4. Challenge construction. η is chosen at random from $\{0, 1\}$ and receives $\mathsf{GenToken}(\mathsf{SK}, \vec{y}_\eta)$.
5. Query Phase II. Identical to Query Phase I.
6. Output phase. $\mathcal{A}$ returns η'.
   If $\eta = \eta'$ then the experiments returns 1 else 0.

Construction for Private-Key Searchable Encryption

Setup $(1^n, 1^\ell)$

1. Select a symmetric bilinear instance $\mathcal{I} = [p, \mathbb{G}, \mathbb{G}_T, g, \mathsf{e}]$.
2. For $i \in [\ell]$, choose random $t_{1,i,0}, t_{2,i,0}, t_{1,i,1}, t_{2,i,1} \in \mathbb{Z}_p$ and set

$$\begin{aligned} \mathsf{K}_i &= \begin{pmatrix} T_{1,i,0} = g^{t_{1,i,0}}, & T_{2,i,0} = g^{t_{2,i,0}} \\ T_{1,i,1} = g^{t_{1,i,1}}, & T_{2,i,1} = g^{t_{2,i,1}} \end{pmatrix} \\ \bar{\mathsf{K}}_i &= \begin{pmatrix} \bar{T}_{1,i,0} = g^{1/t_{1,i,0}}, & \bar{T}_{2,i,0} = g^{1/t_{2,i,0}} \\ \bar{T}_{1,i,1} = g^{1/t_{1,i,1}}, & \bar{T}_{2,i,1} = g^{1/t_{2,i,1}} \end{pmatrix}. \end{aligned}$$

3. Return $\mathsf{SK} = [\mathcal{I}, (\mathsf{K}_i, \bar{\mathsf{K}}_i)_{i \in [\ell]}]$.

Finish...

Construction for Private-Key Searchable Encryption

Encryption $(\mathsf{SK}, \vec{x})$

1. Pick random $s \in \mathbb{Z}_p$.
2. Pick random $s_1, \ldots, s_\ell \in \mathbb{Z}_p$ that sum up to 0.
3. For $i = 1, \ldots, \ell$,
   set $X_{1,i} = T_{1,i,x_i}^{s-s_i}$ and $X_{2,i} = T_{2,i,x_i}^{-s_i}$.
4. Return $\mathsf{Ct}_{\vec{x}} = [(X_{1,i}, X_{2,i})_{i\in[\ell]}]$.

Construction for Private-Key Searchable Encryption

Encryption $(\mathsf{SK}, \vec{x})$

1. Pick random $s \in \mathbb{Z}_p$.
2. Pick random $s_1, \ldots, s_\ell \in \mathbb{Z}_p$ that sum up to 0.
3. For $i = 1, \ldots, \ell$,
   set $X_{1,i} = T_{1,i,x_i}^{s-s_i}$ and $X_{2,i} = T_{2,i,x_i}^{-s_i}$.
4. Return $\mathsf{Ct}_{\vec{x}} = [(X_{1,i}, X_{2,i})_{i\in[\ell]}]$.

GenToken $(\mathsf{SK}, \vec{y})$

1. Pick random $r \in \mathbb{Z}_p$.
2. Pick random $r_1, \ldots, r_\ell \in \mathbb{Z}_p$ that sum up to 0.
3. For $i = 1, \ldots, \ell$,
   set $Y_{1,i} = \bar{T}_{1,i,y_i}^{r-r_i}$ and $Y_{2,i} = \bar{T}_{2,i,y_i}^{-r_i}$.
4. Return $K_{\vec{y}} = [(Y_{1,i}, Y_{2,i})_{i\in[\ell]}]$.

Security proof strategy

Semantic vs. Token security

- Encryption uses keys K_i, $i \in [\ell]$;

Security proof strategy

Semantic vs. Token security

- Encryption uses keys K_i, $i \in [\ell]$;
- Token generation is encryption w.r.t. to keys $\bar{K}_i$, $i \in [\ell]$;

Security proof strategy

Semantic vs. Token security

- Encryption uses keys K_i, $i \in [\ell]$;
- Token generation is encryption w.r.t. to keys $\bar{K}_i$, $i \in [\ell]$;
- In the game for semantic security, $\mathcal{A}$ can ask
  - any encryption query for keys K_i;
  - encryption queries for keys $\bar{K}_i$ and pattern $\vec{y} \neq \vec{x}_0, \vec{x}_1$;

Security proof strategy

Semantic vs. Token security

- Encryption uses keys K_i, $i \in [\ell]$;
- Token generation is encryption w.r.t. to keys $\bar{K}_i$, $i \in [\ell]$;
- In the game for semantic security, $\mathcal{A}$ can ask
  - any encryption query for keys K_i;
  - encryption queries for keys $\bar{K}_i$ and pattern $\vec{y} \neq \vec{x}_0, \vec{x}_1$;
- In the game for key security, $\mathcal{A}$ can ask
  - any encryption query for keys $\bar{K}_i$;
  - encryption queries for keys K_i and attributes $\vec{x} \neq \vec{y}_0, \vec{y}_1$;

Security proof strategy

Semantic vs. Token security

- Encryption uses keys K_i, $i \in [\ell]$;
- Token generation is encryption w.r.t. to keys $\bar{K}_i$, $i \in [\ell]$;
- In the game for semantic security, $\mathcal{A}$ can ask
  - any encryption query for keys K_i;
  - encryption queries for keys $\bar{K}_i$ and pattern $\vec{y} \neq \vec{x}_0, \vec{x}_1$;
- In the game for key security, $\mathcal{A}$ can ask
  - any encryption query for keys $\bar{K}_i$;
  - encryption queries for keys K_i and attributes $\vec{x} \neq \vec{y}_0, \vec{y}_1$;

Semantic Security $\Longleftrightarrow$ Token Security

Zero Sum Assumption

Consider the following game between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.

$\text{ZeroSumExp}_{\mathcal{A}}(1^n, 1^\ell)$

$\mathcal{C}$ randomly picks $a_1, \ldots, a_\ell$ such that $\sum_i a_i = 0$;
$\mathcal{C}$ chooses instance $\mathcal{I} = [p, \mathbb{G}, \mathbb{G}_T, g, \mathrm{e}]$ with security parameter 1^n;
for $i \in [\ell]$
    $\mathcal{C}$ chooses random $u_i \in \mathbb{Z}_p$ and sets $U_i = g^{u_i}$ and $V_i = U_i^{a_i}$;
$\mathcal{C}$ chooses random $\eta \in \{0, 1\}$;
if $\eta = 0$ **then** $\mathcal{C}$ sets V_1 to a random element of $\mathbb{G}$;
$\mathcal{C}$ runs $\mathcal{A}$ on input $[\mathcal{I}, (U_i)_{i \in [\ell]}, (V_i)_{i \in [\ell]}]$;
Let η' be $\mathcal{A}$'s guess for η;
if $\eta = \eta'$ **then** return 1 **else** return 0.

Split Zero Sum Assumption

Consider the following game between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.

$\mathsf{SplitZeroSumExp}_{\mathcal{A}}(1^n, 1^\ell)$

01. $\mathcal{C}$ randomly picks $a_1, \ldots, a_\ell$ such that $\sum_i a_i = 0$;
02. $\mathcal{C}$ chooses instance $\mathcal{I} = [p, \mathbb{G}, \mathbb{G}_T, g, \mathrm{e}]$ with security parameter 1^n;
03. $\mathcal{C}$ chooses random $u, w \in \mathbb{Z}_p$ and sets $W = g^w$;
04. **for** $i \in [\ell]$
    $\mathcal{C}$ chooses random $u_i \in \mathbb{Z}_p$;
    sets $U_i = g^{u_i}, V_i = U_i^{a_i}, A_i = g^{a_i}$, and $S_i = U_i^u$;
05 $\mathcal{C}$ sets $\hat{U} = U_1^w$;
06. $\mathcal{C}$ chooses random $\eta \in \{0, 1\}$;
07. **if** $\eta = 1$ **then** $\mathcal{C}$ sets $Z = W^{u - a_1}$ **else** $\mathcal{C}$ chooses random $Z \in \mathbb{G}$;
08. $\mathcal{C}$ runs $\mathcal{A}$ on input $[\mathcal{I}, (U_i)_{i \in [\ell]}, (V_i)_{i \in [\ell]}, (A_i)_{i \in [\ell]}, (S_i)_{i \in [\ell] \setminus \{1\}}, W, \hat{U}, Z]$;
09. Let η' be $\mathcal{A}$'s guess for η;
10. **if** $\eta = \eta'$ **then** return 1 **else** return 0.

Theorem

Under the Zero Sum Assumption and the Split Zero Sum Assumption, there exists private-key searchable encryption with semantic and key security.

Notice: construction based on pairings on prime order groups.

Further directions

Search

Is sublinear search possible?

Verifiability

A lazy UStorage might say that he found no match.
Can we verify the result?

Thank you