Efficient Private Disjointness Testing

David Pointcheval
ENS - CNRS - INRIA

CANS 2016 - Milan, Italy - November 15th, 2016

Joint work with Geoffroy Couteau and Thomas Peters

The Cloud

iCloud Drive

Dropbox

SkyDrive

Google Drive

amazon cloud drive

Access from Anywhere

Available for Everything

One can

- Store documents, photos, etc
- Share them with colleagues, friends, family
- Process the data
- Ask queries on the data

With Current Solutions

The Cloud provider

- knows the content
- and claims to actually
  - identify users and apply access rights
  - safely store the data
  - securely process the data
  - protect privacy

But...

For economical reasons, by accident, or attacks

- data can get deleted
- any user can access the data
- one can log
  - all the connected users
  - all the queries

to analyze and sell/negotiate the information

Requirements

Users need more

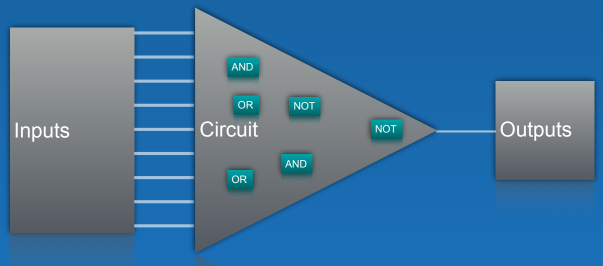
- **Storage** guarantees
- **Privacy** guarantees
  - **confidentiality** of the data
  - **anonymity** of the users
  - **obliviousness** of the queries

How to process users' queries?

FHE: The Killer Tool

[Rivest-Adleman-Dertouzos - FOCS '78]
[Gentry - STOC '09]

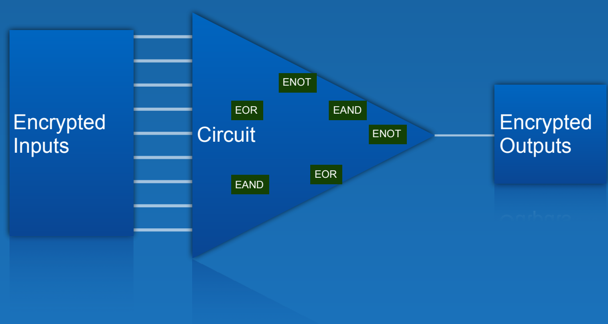
Fully Homomorphic Encryption allows to process encrypted data, and get the encrypted output



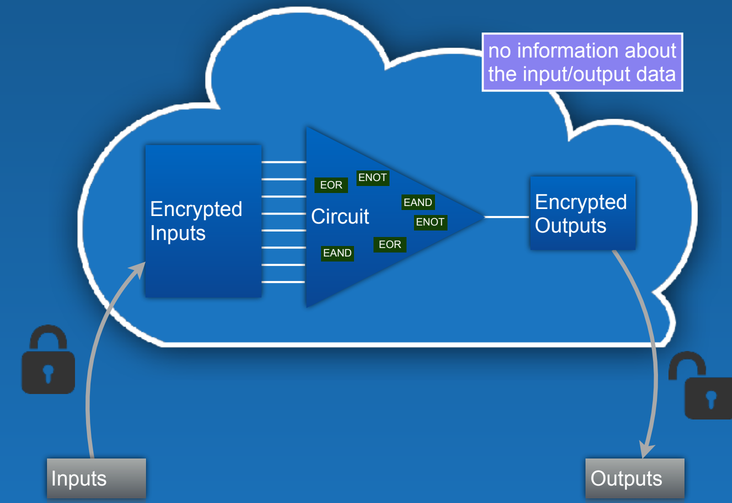
FHE: The Killer Tool

[Rivest-Adleman-Dertouzos - FOCS '78]
[Gentry - STOC '09]

Fully Homomorphic Encryption allows to process encrypted data, and get the encrypted output

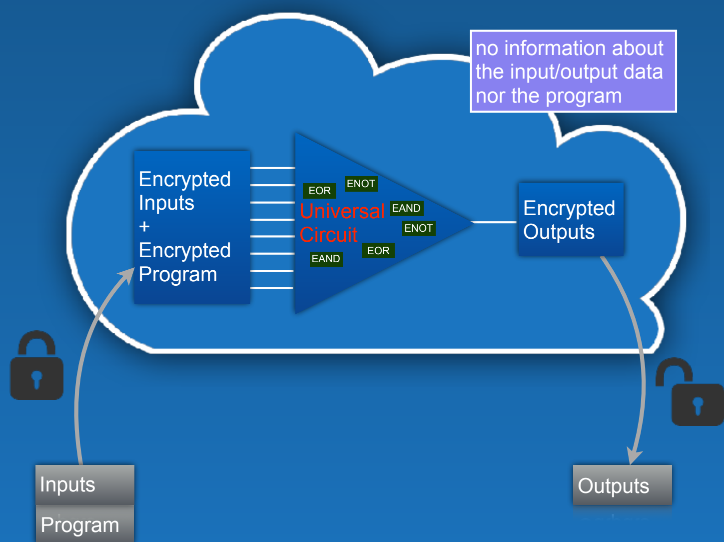


Outsourced Processing



Symmetric encryption (*secret key*) is enough

Strong Privacy



FHE: Ideal Solution?

- Allows private storage
- Allows private computations
  - Private queries in an encrypted database
  - Private « googling »
- The provider does not learn
  - the content
  - the queries
  - the answers

Privacy by design…

… But each gate requires huge computations…

Confidentiality & Sharing

Encryption allows to protect data

- the provider stores them without knowing them
- nobody can access them either, except the owner

How to share them with friends?

- Specific people have **full** access to some data:
  with public-key encryption for multiple recipients
- Specific people have **partial** access
  such as statistics or aggregation of the data

Broadcast Encryption

[Fiat-Naor - Crypto '94]

The sender can select the target group of receivers

- This allows to control **who** will access to the data

Functional Encryption

[Boneh-Sahai-Waters - TCC '11]

The user generates sub-keys K_y according to the input y

- From $C =$ **Encrypt**(x), **Decrypt**(K_y, C) outputs $f(x,y)$
- This allows to control the **amount** of shared data

Outline

- **Broadcast Encryption**
  - Efficient solutions for sharing data
- **Functional Encryption**
  - Some recent efficient solutions for inner product
- **Fully Homomorphic Encryption**
  - Despite recent improvements, this is still inefficient

With 2-party computation
one can get an efficient alternative

Multi-Party Computation

- **Secure** Multi-Party Computation
  - Ideally: each party gives its input and just learns its output
    for **any** ideal functionality

Multi-Party Computation

- **Secure** Multi-Party Computation
  - Ideally: each party gives its input and just learns its output
    for **any** ideal functionality
  - In practice: **many interactions** between the parties

Latency too high over the Internet……

Two-Party Computation

$$z = f(x, y)$$

- General construction: Yao Garbled Circuits
- For specific construction: quite inefficient

$$f(x, y) = (x + y)^e \bmod n$$

Encryption Switching Protocols

$$f(x, y) = (x + y)^e \bmod n$$

With **additive** encryption E^+, **multiplication** encryption $E^\times$, for which Alice and Bob **share** the **decryption** keys, and an interactive **switch** from c^+ to $c^\times$:

- Alices sends $c^+_A = E^+(x)$, and Bob sends $c^+_B = E^+(y)$
- They compute $c = c^+_A \oplus c^+_B = E^+(x+y)$
- They run the **interactive switch** to get $c' = E^\times(x+y)$
- They compute $C = \otimes^e c' = E^\times((x+y)^e)$
- They run the **interactive decryption** to gets z

[Couteau-Peters-P - Crypto '16]

Homomorphic Encryption

[Paillier - Eurocrypt '99]

Additive encryption on $\mathbb{Z}_n$: Paillier encryption

Public key:	$n = pq$
Secret key:	$d = [\lambda^{-1} \bmod n] \times \lambda$
Encryption:	$c = (1+n)^m \cdot r^n \bmod n^2$
Decryption:	$m = [c^d - 1 \bmod n^2]/n$

- Additively homomorphic
- Efficient interactive decryption

Homomorphic Encryption

[ElGamal - IEEE TIT '85]

Multiplicative encryption on $\mathbb{G}$: ElGamal encryption

Secret key:	$x \in \mathbb{Z}_p$
Public key:	$h = g^x$
Encryption:	$c = (c_0 = g^r, c_1 = h^r \cdot m)$
Decryption:	$m = c_1 / c_0^x$

- Multiplicatively homomorphic
- Efficient interactive decryption

If $n = pq$, with safe primes $p = 2p' + 1$ and $q = 2q' + 1$
Works for $\mathbb{G} = \mathsf{QR}_n$, under the DDH in $\mathbb{Z}_{p'}^*$ and $\mathbb{Z}_{q'}^*$
Works for $\mathbb{G} = \mathbb{J}_n$, under the additional QR assumption
But does not work in $\mathbb{Z}_n^*$…

Encoding of Messages

Multiplicative encryption on $\mathbb{Z}_n^*$: by encoding $m \in \mathbb{Z}_n^*$ into $\mathbb{J}_n$

For $n = pq$,	generator g of $\mathbb{J}_n$ of order λ
$\chi \in \mathbb{Z}_n^* \backslash \mathbb{J}_n$,	using the CRT:
	$\chi = g^{t_p} \bmod p$, for an even t_p: $\chi \in \mathsf{QR}_p$
	$\chi = g^{t_q} \bmod q$, for an odd t_p: $\chi \notin \mathsf{QR}_q$
	hence $\chi \in \mathbb{Z}_n^* \backslash \mathbb{J}_n$
For $m \in \mathbb{Z}_n^*$,	$a \in_R \{1, \ldots, n/2\}$, so that $\chi^a \cdot m \in \mathbb{J}_n$
	$m_1 = g^a \bmod n$ and $m_2 = \chi^a \cdot m$
From m_1,	one gets $\alpha = \chi^a \bmod n$ using the CRT:
	$\alpha = m_1^{t_p} \bmod p$ and $\alpha = m_1^{t_q} \bmod q$
From m_2,	one gets $m = m_2/\alpha \bmod n$

Homomorphic Encryption

Multiplicative encryption on $\mathbb{Z}_n^*$: for $n = pq$

Secret key:	$x, t_p, t_q \in \mathbb{Z}_\lambda$
Public key:	$\chi \in \mathbb{Z}_n^* \backslash \mathbb{J}_n$, $\mathbb{J}_n = \langle g \rangle$, $h = g^x$ (ElGamal in $\mathbb{J}_n$)
Encryption:	encode m into $(m_1 = g^a, m_2 = \chi^a \cdot m) \in \mathbb{J}_n^2$
	encrypt m_2 under h, to get (c_0, c_1)
	the ciphertext is $C = (c_0, c_1, m_1)$
Decryption:	decrypt (c_0, c_1) using x, to get m_2
	convert $m_1 = g^a$ into $\alpha = \chi^a$ using the CRT
	get $m = m_2/\alpha \bmod n$

- Multiplicatively homomorphic
- Efficient interactive decryption
- Efficient encryption switching protocols with the Paillier encryption

Two-Party Computation?

The two homomorphic encryption schemes together with the encryption switching protocols:

- Efficient two-party computation
- But in the intersection of the plaintext spaces!

$$\mathbb{Z}_n \cap \mathbb{Z}_n^* = \mathbb{Z}_n^*$$

- **Cannot deal with zero!**
- But **cannot avoid zero** either during computations!

How to Handle Zero?

In order to multiplicatively encrypt $m \in \mathbb{Z}_n$:

One defines $b = 1$ if $m = 0$, and $b = 0$ otherwise
One encrypts $A = m + b \bmod n$
One encrypts $B = T^b \bmod n$ for a random square T

One can note that
$A \in \mathbb{Z}_n^*$, unless m is a non-trivial multiple of p or q
$B \in \mathsf{QR}_n$
$\Longrightarrow$ they can both be encrypted
with appropriate ElGamal-like encryption

- Multiplicatively homomorphic: 0 is absorbing in B
- **Encrypted Zero Test** protocols: $E^+(m) \to E^+(b)$

Set Disjointness Testing

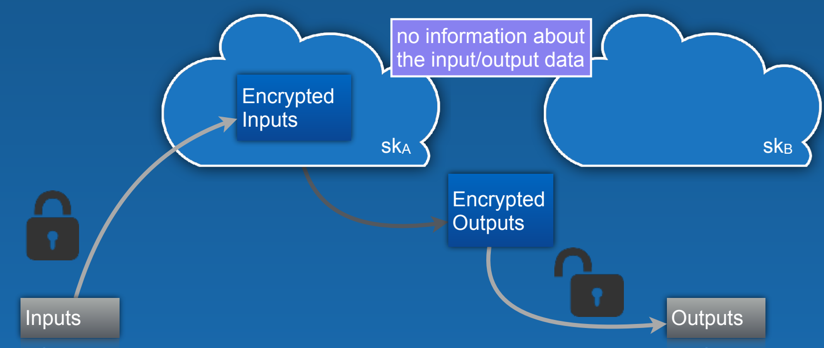
Alice's friends: **A** = $\{a_1, \ldots, a_m\}$ Bob's friends: **B** = $\{b_1, \ldots, b_n\}$

$$\mathbf{A} \cap \mathbf{B} = \emptyset\ ?$$

- Alice computes $P(X) = \prod_i (X - a_i) = \sum_i A_i X^i$,
  and **sends** $C_i = E^+(A_i)$
- Bob computes $B_j = E^+(P(b_j)) = \sum_i b_j^i C_i$
- They **switch** to $B'_j = E^\times(P(b_j))$
- They compute $C' = E^\times(\prod_j P(b_j)) = \prod_j B'_j$
- They **decrypt** $C' \to c = \prod_j P(b_j) = \prod_j \prod_i (b_j - a_i)$

$$c = 0 \Leftrightarrow \mathbf{A} \cap \mathbf{B} \neq \emptyset$$

Outsourced Computations



- The user possesses $n=pq$
- The user gives the shares to 2 independent servers

Interactive Fully Homomorphic Encryption

Homomorphic Encryption

[Bresson-Catalano-P. - Asiacrypt '03]

Additive encryption on $\mathbb{Z}_n$: BCP encryption

Parameters: $n = pq$ and a square $g \in \mathbb{Z}_{n^2}^*$

Secret key: $x \in \mathbb{Z}_{n\lambda(n)}$

Public key: $h = g^x \bmod n^2$

Encryption: $c_0 = g^r \bmod n^2$, for $n \in [1..n^2/2]$

$c_1 = h^r(1 + mn) \bmod n^2$

Decryption: $m = [c_1/c_0^x - 1 \bmod n^2]/n$

Alternatively: with $\lambda(n) \rightarrow x_0 = x \bmod n$

(where $x = x_0 + nx_1$)

$$\begin{aligned} c_1/c_0^{x_0} &= g^{(x-x_0)r} \cdot (1+mn) = (g^{rx_1})^n \cdot (1+mn) \\ &= u^n \cdot (1+n)^m \bmod n^2 \end{aligned}$$

Multi-User Setting

- The two independent servers share the Paillier's secret key for $n{=}pq$ and setup a BCP scheme
- The servers can convert BCP ciphertexts into Paillier ciphertexts, and run the 2-party protocol
- The servers can convert a Paillier ciphertext into a BCP ciphertext for a specific user

$\Rightarrow$ **Secure efficient outsourced computations**

More servers can be used:
unless all the servers corrupted, privacy guaranteed

Conclusion

- **Threat**

  However strong the trustfulness of the Cloud provider may be, any system or human vulnerability can be exploited against privacy
- **Privacy by design**

  Tools to limit data access
- The provider is just trusted to
  - store the data (*can be controlled*)
  - process and answer any request (*or DoS*)