Computing on Encrypted Data

SENY KAMARA
MICROSOFT RESEARCH
Homomorphic Encryption
Homomorphic Encryption

- Gen($1^k$) $\mapsto$ (pk, sk)
- Enc(pk, m) $\mapsto$ c
- Eval(f, c_1, ..., c_n) $\mapsto$ c’
- Dec(sk, c’) $\mapsto$ f(Dec(c_1), ..., Dec(c_n))

- Encryption that supports computation
- Privacy homomorphisms [Rivest-Adleman-Dertouzos78]
Homomorphic Encryption

- **Multiplicatively HE**
  - Vanilla RSA: \( c_1 c_2 = m_1^e \cdot m_2^e = (m_1 m_2)^e \)
  - El Gamal: \( c_1 c_2 = (g^{r_1}, g^{s \cdot r_1} \cdot m_1) \times (g^{r_2}, g^{s \cdot r_2} \cdot m_2) = (g^{r_1 + r_2}, g^{s \cdot (r_1 + r_2)} \cdot m_1 m_2) \)

- **Additively HE** [Benaloh88, Paillier99, Damgard-Jurik03]

- **XOR HE** [Goldwasser-Micali82]

- **Quadratic polynomials** [Boneh-Goh-Nissim05, Gentry-Halevi-Vaikuntanathan10]

- **Branching programs** [Ishai-Paskin07]
Fully-Homomorphic Encryption
[Gentry09]

- Supports general-purpose computation
  - Any polynomial-size circuit
  - Suffices to support addition & multiplication
    - Gives a NAND gate
    - $1+b_1xb_2$ over $GF(2)$
Gentry’s Blueprint

- Construct somewhat homomorphic encryption
  - Handles limited amount of computation
- Construct Leveled-FHE via bootstraping
  - If SHE handles at least own decryption algorithm…
  - …we transform SHE into FHE with O(d)-length pk
- Construct a bootstrappable SHE via squashing
  - If SHE decryption is too complex, …
  - make it less complex by squashing
Bootstrapping

- SHE schemes are noisy
  - Noise increases after every homomorphic operation
  - If too much noise then decryption stops working
  - This is why SHE handles limited computations

- Bootstrapping manages noise
  - Fresh encryptions have minimum amount of noise
  - **Idea:** if too much noise, then homomorphically decrypt!
  - **Only works if SHE can homomorphically decrypt**
    - SHE is bootstrappable
Suppose we have

$(pk_1, sk_1)$ and $c_1 \leftarrow Enc(pk_1, m)$ with too much noise

Compute

$(pk_2, sk_2) \leftarrow Gen(1^k)$

$c_2 \leftarrow Enc(pk_2, c_1)$

$c_3 \leftarrow Enc(pk_2, sk_1)$

$c_4 \leftarrow Eval(Dec, c_2, c_3)$

Homomorphically decrypting $c_1$ with $sk_1$

$c_4 = Enc(pk_2, m)$

$c_4$ is “fresh” and has less noise!
Bootstrapping

- Bootstrapped Evaluation
  - While evaluating a circuit, ...
  - ...bootstrap when too much noise

- Leveled FHE
  - Bootstrapping requires adding encrypted sk to pk
  - \( pk = pk_1, pk_2, \text{Enc}(pk_2, sk_1), \ldots, pk_n, \text{Enc}(pk_n, sk_{n-1}) \)

- FHE
  - If scheme is circular-secure then pk has 1 enc sk
  - [Gentry09] is shown circular-secure in ROM
FHE Security

- [Gentry09] is CPA-secure
  - Under ideal lattice assumptions
  - Under subset-sum assumption (for squashing)
  - In random oracle model (for circular security)
- CCA2-security is impossible for FHE
  - Since it is homomorphic!
Recent Improvements

- LWE-based FHE \([\text{Brakerski-Vaikuntanathan11b}]\)
  - Tensoring, key switching & dimension reduction
  - No ideals
  - No circular security
- \([\text{Brakerski-Gentry-Vaikuntanathan12}]\)
  - Modulus switching
  - No bootstrapping
- Low-noise FHE \([\text{Brakerski12}]\)
  - Noise grows only linearly with each multiplication
FHE Performance

- [Gentry-Halevi11]
  - Optimized version of [Gentry09]
  - 64-bit quad-core, Intel Xeon E5450 proc, 3GHz, 24GB
  - pk size: 2.25GB
  - Key generation: 2.2 hours
  - Encryption: 3 mins
  - Decryption: 0.66 sec
  - Refresh: 31 mins
FHE Performance

- [Gentry-Halevi-Smart12]
  - Evaluation of AES on 128-bit message w/ 128-bit key
  - [Brakerski-Gentry-Vaikuntanathan12] (no bootstrapping)
  - 256GB of RAM
  - Evaluation: 36 hours
SHE Performance

- [Lauter-Naehrig-Vaikuntanathan11]
- [Groepel-Lauter-Naehrig12]
- Implement [Brakerski-Vaikuntanathan11b]
  - Intel Core i5, 3.2GHz
  - Key generation: 156 ms
  - Encryption (4095-bit message): 379 ms
  - Decryption: 29 ms
  - Addition: 1 ms
  - Multiplication: 106 ms
Applications of SHE
[Lauter-Naehrig-Vaikuntanathan11]

- Statistical functions
  - Average, standard deviation, logistical regression
- Medical scenarios
  - Doctors, medical devices send encrypted data to cloud...
  - ...cloud processes data and sends results to patient
- Financial scenarios
  - Stock price, inventory, performance etc. sent encrypted to cloud...
  - ...cloud processes data and sends analysis to trader
Applications of SHE

[Lauter-Naehrig-Vaikuntanathan11]

- Advertising
  - Your mobile phone sends encrypted behavioral info to cloud…
  - …cloud processes info and sends back targeted ad
Applications of SHE

[Groepel-Lauter-Naehrig12]

- Machine learning
  - Linear regression

- Supervised learning
  - Training data sent encrypted to cloud…
  - …cloud trains a classifier…
  - …client sends encrypted data to cloud…
  - …cloud classifies data and sends result back
Limitations of FHE/SHE

- FHE is slow (for now)
- No guarantee computation is correct
- Inherently slow for some applications
  - Search on encrypted data (?)
- Cannot handle some applications
  - [van Dijk-Juels10]
  - Private multi-client computations with access control
Garbled Circuits
Garbled Circuits [Yao82]

- $\text{GC}(t^k, C) \iff (C, dk, sk)$
- $\text{GI}(sk, x) \iff X$
- $\text{Eval}(C, X) \iff y$
- $\text{Dec}(dk, y_i) \iff \{\bot, y_i\}$

AND:

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

Enc_{K_0}(Enc_{K_0}(K_0))
Enc_{K_0}(Enc_{K_1}(K_0))
Enc_{K_1}(Enc_{K_0}(K_0))
Enc_{K_1}(Enc_{K_1}(K_1))
Garbled Circuits [Yao82]

\[\text{Enc}_{K_0}(\text{Enc}_{K_0}(K_0)) \]
\[\text{Enc}_{K_0}(\text{Enc}_{K_1}(K_1)) \]
\[\text{Enc}_{K_1}(\text{Enc}_{K_0}(K_0)) \]
\[\text{Enc}_{K_1}(\text{Enc}_{K_1}(K_1)) \]
\[\text{Enc}_{K_0}(\text{Enc}_{K_0}(K_0)) \]
\[\text{Enc}_{K_0}(\text{Enc}_{K_1}(K_1)) \]
\[\text{Enc}_{K_1}(\text{Enc}_{K_0}(K_0)) \]
\[\text{Enc}_{K_1}(\text{Enc}_{K_1}(K_1)) \]
Applications of Garbled Circuits

- Two-party secure computation [Yao82]
- Single-server-aided MPC [K.-Raykova-Mohassel]
- One-time programs [Goldwasser-Kalai-Rothblum10]
- Verifiable computation [Gennaro-Gentry-Parno10]
- FHE-like systems
Garbled Circuits “FHE”

- **Gen($1^k$)**
  - $(sk, pk) \leftarrow PKE.Gen(1^k)$

- **Enc($pk, C, m$)**
  - $(C, dk, sk) \leftarrow GC(1^k, C)$
  - $M \leftarrow GI(sk, m)$
  - $c_1 \leftarrow PKE.Enc(pk, dk)$
  - $c = (c_1, C, M)$

- **Eval($c$)**
  - $\gamma \leftarrow GC.Eval(C, M)$
  - $c' = (c_1, \gamma)$

- **Dec($sk, c'$)**
  - $dk \leftarrow PKE.Dec(sk, c_1)$
  - $m \leftarrow GC.Dec(dk, \gamma)$

$O(|C|)$
FHE vs Garbled Circuits

- **Computations**
  - GC: arbitrary
  - FHE: arbitrary

- **Encryption time**
  - GC: $O(|C| + n)$
  - FHE: $O(n)$

- **Decryption time**
  - GC: $O(n)$
  - FHE: $O(n)$

- **Evaluation time**
  - GC: $O(|C|)$
  - FHE: $O(|C|)$

- **Constants**
  - GC: small
  - FHE: very large

- **# of Evaluations**
  - GC: one
  - FHE: unlimited

ERSICS/Aix-Marseille
FHE vs Garbled Circuits

- Ciphertext size
  - GC: $O(\| C \|)$
  - FHE: $O(n)$

- Ciphertext size for AES
  - GC: 500 Kbytes
Garbled Circuits
Implementations

- FastGC [Huang-Evans-Katz-Malka10]
  - Evaluation of AES on 128-bit message w/ 128-bit key
  - Evaluation: 0.2 s (for 2-party computation)

- Recall
  - [Gentry-Halevi-Smart12]
  - Evaluation of AES on 128-bit message w/ 128-bit key
  - Using [Brakerski-Gentry-Vaikuntanathan12] (no bootstrapping)
  - 256GB of RAM
  - Evaluation: 36 hours