Secure Computation
EWSCS, Lecture 4

Claudio Orlandi
Grand Plan

• **Before:** 2PC protocols with focus on:
  – Minimal assumptions (semi-tp, oblivious transfer) ☺
  – Practical Efficiency (preprocess., OT extension, …) ☺

• **Today:** 2PC from **Fully Homomorphic Encryption**
  – Stronger assumptions ☹
  – Not practical (yet?) ☹
  – 2PC with small communication ☺
  – 2PC with small workload for the client ☺
  – (In some cases) hide the size of the input in 2PC ☺
(Very brief intro to)
Fully Homomorphic Encryption
Fully Homomorphic Encryption

- Fully Homomorphic Encryption
  \((Gen, Enc, Dec, Eval)\)

  - Correctness:
    \[ Dec_{sk}(Eval_{pk}(f, Enc_{pk}(x_1), Enc_{pk}(x_n))) = f(x_1, \ldots, x_n) \]
“Trivial FHE”

• $(Gen, Enc, Dec)$ IND-CPA secure scheme.

• Define

$$Eval(f, C_1, \ldots C_n) = (C_1, \ldots C_n, f) = C^*$$

• (Re)-Define

$$Dec^*(sk, C^*) = f(Dec(sk, C_1), \ldots, Dec(sk, C_n))$$

• Fully-homomorphic!

(But clearly not what you were thinking of)
Fully Homomorphic Encryption

• Fully Homomorphic Encryption
  \((Gen, Enc, Dec, Eval)\)

  – Correctness:
  \(Dec_{sk}(Eval_{pk}(f, Enc_{pk}(x_1), Enc_{pk}(x_n)) = f(x_1, ..., x_n)\)

  – Circuit privacy:
  \((sk, Eval_{pk}(f, Enc_{pk}(x_1), Enc_{pk}(x_n))) \approx (sk, Enc_{pk}(f(x_1, ..., x_n)))\)
2PC from FHE

Alice

\[(pk, sk) \leftarrow \text{Gen}\]

\[C_1 \leftarrow \text{Enc}(pk, x)\]

\[f(x,y) = \text{Dec}(sk, C_2)\]

Bob

\[C_2 \leftarrow \text{Eval}(pk, f(\cdot, y), C_1)\]
(Insecure) GSW d-he

- **Secret key**: vector $s$
- **Ciphertext**: matrix $C$ s.t. $Cs = ms$
  
  // $s$ is eigenvector, $m$ is eigenvalue
- If
  
  $C_1s = m_1s$ and $C_2s = m_2s$
- Then
  
  $(C_1 + C_2)s = (m_1 + m_2)s$
  $(C_1 * C_2)s = (m_1 * m_2)s$
LWE-based GSW \textit{d-he}

- \textbf{Secret key}: vector $s$
- \textbf{Ciphertext}: matrix $C$ s.t. $Cs = ms + e$

If

$$C_1s = m_1s + e_1 \text{ and } C_2s = m_2s + e_2$$

Then

$$(C_1 + C_2)s = (m_1 + m_2)s + (e_1 + e_2)$$
$$(C_1 * C_2)s = (m_1 * m_2)s + E$$

After “few” multiplications, the error becomes too big to allow for decryption.
d-he (or somewhat-he, or leveled-he)

\((\text{gen, enc, dec, eval})\)

- **Correctness**: if \(f\) is a circuit of depth \(< d\)
  \[\text{dec}_{sk}(\text{eval}_{pk}(f, \text{enc}_{pk}(x_1), \text{enc}_{pk}(x_n))) = f(x_1, \ldots, x_n)\]

- **Circuit privacy**: 
  \[\left(\text{sk, eval}_{pk}(f, \text{enc}_{pk}(x_1), \text{enc}_{pk}(x_n))\right) \approx \left(\text{sk, enc}_{pk}(f(x_1, \ldots, x_n))\right)\]
From \textit{d-he} to \textit{\infty-HE}

• Useful notation
  – $f^j$ circuit of depth $j$
  – $c^0 = \text{enc}(pk, m)$ is a level 0 ciphertext

• Then
  
  \[ c^{i+j} = \text{eval}(pk, f^i, c^j) \]

  is a level $i+j$ ciphertext

• Correctness
  \[ \text{dec}(sk, c^i) = m \]

  holds as long as $i < d$
From $d$-he to $\infty$-HE

• Ingredients
  – Take a $d$-he
    $$(\text{gen,enc,dec,eval})$$
  – where $\text{dec}(\ast,c^i)$ is a shallow circuit (depth $< d-1$)
  – Circular secure: it is "safe" to
    $\text{enc}(pk,sk)$
From $d$-he to $\infty$-HE

- $\infty$-HE: $\text{Enc} = \text{enc}$, $\text{Dec} = \text{dec}$, $SK = sk$
  - $PK = (pk, ck^0 = \text{enc}(pk, sk))$
  - For any ciphertext $C$ s.t. $\text{dec}(sk, C) = m$ define:
    \[ f_C(x) = \text{dec}(x, C) \quad // f \text{ has depth } < d - 1 \]

- Warmup: define
  \[ \text{Eval}(PK, Id, C) = \text{eval}(pk, f_C, ck^0) = Z \]
  \[ \text{dec}(sk, Z) = f_C(x = sk) \]
  \[ = \text{dec}(sk, C) \]
  \[ = m \]
From $d$-he to $\infty$-HE

- $Eval(PK,Id,C) = eval(pk,f_C,ck^0) = Z$
  
  $\text{dec}(sk,Z) = f_C(x=sk)$
  $= \text{dec}(sk,C)$
  $= m$

C has some level $i < d$

Z has level $< 0 + (d-1)$ (independent of $i$)
From \textit{d-he} to \textit{$\infty$-HE}

- \( f_{C_1,C_2}(x) = (\text{dec}(x,C_1) \text{ NAND } \text{dec}(x,C_2)) \)
  
  \hspace{1cm} \text{//} f \text{ has depth } < (d - 1) + 1 = d

- \( \text{Eval}(PK,\text{NAND},C_1,C_2) = \text{eval}(pk,f_{C_1,C_2},ck^0) = Z \)
  \[
  \text{dec}(sk,Z) = f_{C_1,C_2}(sk) = (\text{dec}(sk,C_1) \text{ NAND } \text{dec}(sk,C_2))
  \]
  \[
  = m_1 \text{ NAND } m_2
  \]
Hiding the Input Size in 2PC
(Lindell, Nissim, Orlandi, Asiacrypt’13)
Privacy on Facebook (or a more privacy sensitive social network)

My friends should only see our common friends
Privacy on Facebook (or a more privacy sensitive social network)
Privacy on Facebook (or a more privacy sensitive social network)

Friend list ➔ 2PC protocol ➔ Intersection + size of friend list ➔ Friend list
You learned more than you were supposed to!

Don’t worry, it’s only metadata!

(size of friend list!)
Secure Computation

- Privacy
- Correctness
- Input Independence
- “The protocol is as secure as the ideal world”

Or is it?
Size matters!

- Private Set Intersection: the size of a list might be confidential
- Padding?
  - Just add a lot of “fake entries” to your DB
  - Requires an upper bound 😞
  - Inherent inefficiency 😞
Impossibility of Size-Hiding: Proof by Authority

[G04] “...making no restriction on the relationship among the lengths of the two inputs disallows the existence of secure protocols for computing any nondegenerate functionality...”

[IP07] “...hiding the size of both inputs is impossible for interesting functions...”

[HL10] “...We remark that some restriction on the input lengths is unavoidable because, as in the case of encryption, to some extent such information is always leaked...”
Impossibility of Size-Hiding: Proof by Authority

[G04] “…making no restriction on the relationship among the lengths of the two inputs disallows the existence of secure protocols for computing any nondegenerate functionality…”

[IP07] “…hiding the size of both inputs is impossible for interesting functions…”

[HL10] “…We remark that some restriction on the input lengths is unavoidable because, as in the case of encryption, to some extent such information is always leaked…”

The input size can be protected! (In some interesting cases)
A Test Case: Standard Definition

• Standard definition, e.g. [Gol04]

• Need to know other party’s size in advance
  – Introduces problem of input size dependence
  – One party can choose its input after knowing the size of the other party’s input (outside the scope of the protocol)
Ideal Model - Classes

• Classes
  – 0: both input-sizes are leaked
  – 1: Bob learns $|x|$, Alice does not learn $|y|$  
  – 2: both input-sizes are protected

• Subclasses
  – Who gets output?
  – Is the output size leaked?

• Our classification is complete for symmetric functions $f(x, y) = f(y, x)$
Class 0

\[ x \rightarrow 1^{\lvert y \rvert}, f(x, y) \quad \text{Class 0} \quad y \rightarrow 1^{\lvert x \rvert}, f(x, y) \]
Class 1

Essentially equivalent classes (outputs have same length)
Class 2

2.a
\[
x \\ f(x, y) \\ y \\ f(x, y)
\]

2.b
\[
x \\ f(x, y) \\ y \\ f(x, y)
\]

2.c
\[
x \\ f(x, y) \\ y \\ 1 |f(x, y)|
\]
Lower Bounds
(the bad news)
• There are functions that cannot be computed while hiding both parties’ input size.
  – (Not everything can be computed in Class 2)

• For instance: *Inner product, Set Intersection, Hamming distance, etc.*
**Proof idea:**
- $IP(x, y) = \sum_i x_i y_i \mod 2$

**Size Hiding IP must have comm. complexity $poly(k)$**
- A size-hiding IP protocol must hide the size of the inputs
- The length of the transcript should not leak information
- Therefore the communication complexity cannot depend on the input length

**But! Lower bounds in communication complexity say that to compute $IP(x, y)$ at least $\min(|x|, |y|)$ need to be communicated!**

**Contradiction!**
• Size-hiding OT:
  – $x =$ selection bit
  – $y = (y_0, y_1)$ two strings of different length
  – $f(x, y) = y_x$

• Thm: OT cannot be computed in Class 1.b
Class 1.b

Proof idea:

R should not learn anything about $y_1 - x$
- So the #bits S sends to R must be independent of size of $y_1 - x$

S should not learn anything about $x$
- If $|y_0| \neq |y_1|$ and the #bits S sends to R depend on $y_x$ then S learns information about $x$!
- So the length of the transcript must be independent of $y_x$ too

So the transcript length \textbf{must} be $t = \text{poly}(k)$
- But! If $|y_x| = t + 1$, S communicated $t + 1$ bits to R by sending only $t$ bits.
- Impossible! (Incompressibility of random data)
Feasibility results
(the good news!)
Class 1.a

\[(pk, sk) \leftarrow \text{Gen}(1^k)\]
\[c_x \leftarrow \text{Enc}_{pk}(x)\]
\[z = \text{Dec}_{sk}(c_z)\]

\[c_z = \text{Eval}_{pk}(f(\cdot, y), c)\]
How big should the output be?

\[ E(x) \]

\[ f(\cdot, y) \]

\[ E(z) \]

e.g. \( z = x \cup y \)

Clear that \( |z| \leq |x| + |y| \)

But how long exactly?

Any upper bound reveals info about \( |y| \)
\( \text{Thm: FHE} \Rightarrow \forall f \) can be securely computed in Classes 1.a/c/e
How big should the output be?

Alice opens $\ell = |z|$

Send to Alice

$E(x)$

$E(|z|)$

$E(\ell(\cdot, y))$

$E(z)$
Size-hiding comparison
(the gazillionaire protocol)
How to compare long strings

\[
\begin{array}{cccccccccccc}
\times & 0000 & 0000 & 0000 & 0000 & 0000 & 0000 & 0110 & 1100 & 0011 \\
\end{array}
\]

\[
\begin{array}{cccccccccccc}
\times & 0000 & 0000 & 0000 & 0000 & 0000 & 0000 & 0110 & 1100 & 0011 \\
\end{array}
\]
How to compare long strings

\[ x \quad 0000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0110 \ 1100 \ 0011 \]

\[ y \quad 0000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0011 \ 0111 \ 0110 \ 0000 \]
How to compare long strings

Equal $\Rightarrow$ go right
How to compare long strings

\[ x \]

0000 0000 0000 0000 0000 0000 0110 1100 0011

\[ y \]

0000 0000 0000 0000 0000 0011 0111 0110 0000

Different \(\rightarrow\) go left
How to compare long strings

x 0000 0000 0000 0000 0000 0000 0110 1100 0011

\[ \ldots \]

y 0000 0000 0000 0000 0000 0011 0111 0110 0000

\[ \ldots \]
How do we construct one protocol that works for any input size (and does not reveal it?)
Tools: Merkle-Damgård trees  
(with a twist)

• Let $H : \{0,1\}^{2k} \rightarrow \{0,1\}^k$

\[
Tree(x) = \begin{cases} 
  x & \text{if } |x| = k \\
  H(Tree(x_L), Tree(x_R)) & \text{o.w.}
\end{cases}
\]

• Assume $H(0^k, 0^k) = 0^k$

• Then we can compute $Tree(x)$ for any $x$ of any polynomial length
Merkle Tree

Root

0

0 0

0

0 0

\( k^{\log k} \)

\( \log^2 k \)
Size-Hiding "greater than"

\[ x = (x_L, x_R) \]
\[ \text{root} \leftarrow \text{Tree}(x_L) \]

Until \(|x| > k\)

\[ |x| = k \]

\[ x \leftarrow x_{R/L} \]

\[ y = (y_L, y_R) \]

\[ \text{if } \text{root} = \text{Tree}(y_L) \]
\[ R \text{ else } L \]
\[ y \leftarrow y_{R/L} \]

Output
\[ z \leftarrow (x > y) \]
Protocol properties

• Comm. complexity independent of input size!
  – #rounds: $O(\log^2 k)$
  – #bits: $k$ per round

• This protocol is insecure
  – Reveals where the inputs differ
  – this can be fixed using FHE like for Class 1 protocols
Future directions

• (More) efficient protocols for specific tasks?
• Malicious security?
• Dealing with side-channel attacks (timing)?
• Relationship to FHE?

• Hiding the input size is (sometimes) possible.
  – Don’t give up!
• Landscape of size-hiding 2PC is very rich
  – Many positive and negative results.
Aarhus, Denmark, May 5-9 2014
(right before EUROCRYPT)

- Registration
- (preliminary) Program
- Confirmed speakers
- Stipends (deadline March 15th)

http://cfem.au.dk/events/mpc-2014/