Secure Multi-Party Computation

Gunnar Kreitz

KTH – Royal Institute of Technology
gkreitz@kth.se

October 4 2012
Ingredients

- *n* parties
- *n* inputs (one per party)
- A function $f(x_1, \ldots, x_n)$ to compute
Goal (intuitive)

- Parties learn $f(x_1, \ldots, x_n)$
- No one learns anything more
Example time!

Let’s pick a function
The classic examples (Millionaire’s Problem)
The classic examples (Mental Poker)

Mental Poker

Adi Shamir, Ronald L. Rivest and Leonard M. Adleman

Massachusetts Institute of Technology

ABSTRACT

Can two potentially dishonest players play a fair game of poker without using any cards—for example, over the phone? This paper provides the following answers:

1 No. (Rigorous mathematical proof supplied.)
The classic examples (Dining Cryptographers)

\[ [1] \leftarrow 1 \oplus k \]
\[ [2] \leftarrow k \]
\[ [1] \oplus [2] = 1 \]
Example time!

\[ \sum x_i \]
Private Summation (cont’d)

- The protocol does one round of input randomization (*blinding*)
- Then, any (non-private) summation protocol is run on the blinded inputs
- The blinding preserves the sum of the inputs
- Information-theoretically secure
Summation Protocol by Example

\[ x_1' = x_1 \]
\[ x_2' = x_2 \]
\[ x_3' = x_3 \]
Summation Protocol by Example

\[ x_1' = x_1 - r_{12} \]
\[ x_2' = x_2 + r_{12} \]
\[ x_3' = x_3 \]
Summation Protocol by Example

\[
\begin{align*}
    x'_1 &= x_1 - r_{12} - r_{13} \\
    x'_2 &= x_2 + r_{12} \\
    x'_3 &= x_3 + r_{13}
\end{align*}
\]
Summation Protocol by Example

\[ x'_1 = x_1 - r_{12} - r_{13} + r_{21} \]
\[ x'_2 = x_2 + r_{12} - r_{21} \]
\[ x'_3 = x_3 + r_{13} \]
Summation Protocol by Example

\[ x'_1 = x_1 - r_{12} - r_{13} + r_{21} \]
\[ x'_2 = x_2 + r_{12} - r_{21} - r_{23} \]
\[ x'_3 = x_3 + r_{13} + r_{23} \]
Summation Protocol by Example

\[ x_1' = x_1 - r_{12} - r_{13} + r_{21} + r_{31} \]
\[ x_2' = x_2 + r_{12} - r_{21} - r_{23} \]
\[ x_3' = x_3 + r_{13} + r_{23} - r_{31} \]
Summation Protocol by Example

Let $x_1, x_2, x_3$ be the inputs for parties 1, 2, and 3, respectively. The protocol for summation is as follows:

- Party 1\: $x'_1 = x_1 - r_{12} - r_{13} + r_{21} + r_{31}$
- Party 2\: $x'_2 = x_2 + r_{12} - r_{21} - r_{23} + r_{32}$
- Party 3\: $x'_3 = x_3 + r_{13} + r_{23} - r_{31} - r_{32}$
Private Summation Protocol

- Each party $P_i$ with input $x_i$ proceeds as follows:
  1. Send random $r_{i,j}$ to each neighbor $P_j$
  2. Wait for $r_{j,i}$ from each neighbor $P_j$
  3. Compute

$$x'_i = x_i + \sum_{P_j\text{neighbor}} r_{j,i} - \sum_{P_j\text{neighbor}} r_{i,j}$$

- We could now publish $x'_i$ and still remain private!
How to proceed?

- Do we develop protocols for each and every $f$?
- (Are they all this simple?)
- How do we define security?
Security definitions

- Noone should learn anything but result
- Noone should be able to affect computation in an untoward way
A Trusted Third Party

- Is there someone we all trust?
- Can send measurements to the Trusted Third Party
- She performs computation and tells everyone result
- Given a Trusted Third Party, problem is easy
Sometimes There is no Trusted Third Party
Secure?

What do we mean by security?

- In an ideal world, we have a trusted third party
- We want our protocols to be as secure as the ideal world
- Cheating parties must not:
  - learn more than they do in the ideal world
  - be able to do more than they can in the ideal world
What is an attack?

Functionality: $\sum_i x_i \pmod{p}$. Adversary corrupts party 1.

- Adversary learns $x_1$?
What is an attack?

Functionality: $\sum_i x_i \pmod{p}$. Adversary corrupts party 1.

- Adversary learns $x_1$? No.
What is an attack?

Functionality: $\sum_i x_i \pmod{p}$. Adversary corrupts party 1.

▶ Adversary learns $x_1$? No.
▶ Adversary learns $x_{10}$?
What is an attack?

Functionality: $\sum_i x_i \pmod{p}$. Adversary corrupts party 1.

- Adversary learns $x_1$? No.
- Adversary learns $x_{10}$? Yes.
What is an attack?

Functionality: $\sum_i x_i \pmod{p}$. Adversary corrupts party 1.

- Adversary learns $x_1$? No.
- Adversary learns $x_{10}$? Yes.
- Adversary learns sum of all other parties’ input?
What is an attack?

Functionality: $\sum_i x_i \mod p$. Adversary corrupts party 1.

- Adversary learns $x_1$? No.
- Adversary learns $x_{10}$? Yes.
- Adversary learns sum of all other parties’ input? No.
What is an attack?

Functionality: $\sum_i x_i \pmod{p}$. Adversary corrupts party 1.

- Adversary learns $x_1$? No.
- Adversary learns $x_{10}$? Yes.
- Adversary learns sum of all other parties’ input? No.
- Adversary learns $\sum_{i<n/2} x_i$?
What is an attack?

Functionality: $\sum_i x_i \pmod{p}$. Adversary corrupts party 1.

- Adversary learns $x_1$? No.
- Adversary learns $x_{10}$? Yes.
- Adversary learns sum of all other parties’ input? No.
- Adversary learns $\sum_{i < n/2} x_i$? Yes.
What is an attack?

Functionality: $\sum_i x_i \pmod{p}$. Adversary corrupts party 1.

- Adversary learns $x_1$? No.
- Adversary learns $x_{10}$? Yes.
- Adversary learns sum of all other parties’ input? No.
- Adversary learns $\sum_{i<n/2} x_i$? Yes.
- Adversary learns sum, everyone else gets random value?
What is an attack?

Functionality: $\sum_i x_i \pmod{p}$. Adversary corrupts party 1.

- Adversary learns $x_1$? No.
- Adversary learns $x_{10}$? Yes.
- Adversary learns sum of all other parties’ input? No.
- Adversary learns $\sum_{i<n/2} x_i$? Yes.
- Adversary learns sum, everyone else gets random value? No (pick random $x_1$).
What is an attack?

Functionality: \( \sum_i x_i \pmod{p} \). Adversary corrupts party 1.

- Adversary learns \( x_1 \)? No.
- Adversary learns \( x_{10} \)? Yes.
- Adversary learns sum of all other parties’ input? No.
- Adversary learns \( \sum_{i<n/2} x_i \)? Yes.
- Adversary learns sum, everyone else gets random value? No (pick random \( x_1 \)).
- Adversary ensures result is \( c \)?
What is an attack?

Functionality: $\sum_i x_i \pmod{p}$. Adversary corrupts party 1.

- Adversary learns $x_1$? No.
- Adversary learns $x_{10}$? Yes.
- Adversary learns sum of all other parties’ input? No.
- Adversary learns $\sum_{i<n/2} x_i$? Yes.
- Adversary learns sum, everyone else gets random value? No (pick random $x_1$).
- Adversary ensures result is $c$? Yes.
How Powerful is Our Adversary?

- Two main models of adversary’s evilness:
  - Passive/semi-honest (*Honest-but-curious*): follows protocol but tries to deduce more information
  - Active/malicious (*Byzantine*): arbitrary deviations from protocol
How Powerful is Our Adversary?

- Two main models of adversary’s power:
  - Computational Security: Probabilistic polynomial time
  - Information-Theoretic Security: Unlimited computation time
- In this talk, we consider both notions
One protocol to rule them all

- How can we get around having to design one protocol per functionality?
One protocol to rule them all

- How can we get around having to design one protocol per functionality?
- Something that can evaluate a circuit.
Main idea

- Keep all intermediary values secret shared
- Evaluate circuit gate by gate, gate inputs and outputs being secret shared
- Open up values of output gates to everyone
- We’ll need protocols for addition (XOR) and multiplication (AND)
Different variations

- Built on Shamir/Verifiable Secret Sharing [BGW88, CCD88]
- Built on Oblivious Transfer [GMW87]
- Built on Homomorphic Encryption
Shamir secret sharing

- Math is now in a finite field (“mod a prime”)
- Pick a polynomial $P(x)$ of degree $t$, with $P(0) = s$
- Knowing evaluations at $t + 1$ points uniquely determines $P(x)$
- Evaluations at $t$ coordinates ($\neq 0$) reveal nothing about $s$
Secure computation: addition (XOR)

- Input: two polynomials* $f(x)$, $g(x)$ with $f(0) = a$, $g(0) = b$
- Output: polynomial* $h(x)$ such that $h(0) = a + b$
Secure computation: addition (XOR)

- Input: two polynomials\( f(x), g(x) \) with \( f(0) = a, g(0) = b \)
- Output: polynomial\( h(x) \) such that \( h(0) = a + b \)
- \( h(x) = f(x) + g(x) \) has the right property
Secure computation: addition (XOR)

- Input: two polynomials* $f(x), g(x)$ with $f(0) = a$, $g(0) = b$
- Output: polynomial* $h(x)$ such that $h(0) = a + b$
- $h(x) = f(x) + g(x)$ has the right property
- Party $P_i$ knows $f(i), g(i)$. Need a protocol for her to learn $h(i)$
Secure computation: addition (XOR)

- Input: two polynomials \( f(x), g(x) \) with \( f(0) = a, g(0) = b \)
- Output: polynomial \( h(x) \) such that \( h(0) = a + b \)
- \( h(x) = f(x) + g(x) \) has the right property
- Party \( P_i \) knows \( f(i), g(i) \). Need a protocol for her to learn \( h(i) \)
- \( h(i) = f(i) + g(i) \) — XOR gates can be evaluated locally!
Secure computation: multiplication (AND)

- Input: two polynomials \( f(x), g(x) \) with \( f(0) = a, g(0) = b \)
- Output: polynomial \( h(x) \) such that \( h(0) = ab \)
Secure computation: multiplication (AND)

- Input: two polynomials* $f(x), g(x)$ with $f(0) = a, g(0) = b$
- Output: polynomial* $h(x)$ such that $h(0) = ab$
- $h(x) = f(x)g(x)$ has the right property
Secure computation: multiplication (AND)

- Input: two polynomials \( f(x), g(x) \) with \( f(0) = a, g(0) = b \)
- Output: polynomial \( h(x) \) such that \( h(0) = ab \)
- \( h(x) = f(x)g(x) \) has the right property
- But, it is a bad choice!
- It has degree \( 2t \)
- It is not uniformly random (e.g., cannot be irreducible)
Secure computation: multiplication (AND) (cont’d)

- \[ h(x) = f(x)g(x) \]
Secure computation: multiplication (AND) (cont’d)

- $h(x) = f(x)g(x)$
- To make it uniformly random: add random polynomials with $p(0) = 0$
- Each party picks one: $h'(x) = f(x)g(x) + \sum_i p_i(x)$
Secure computation: multiplication (AND) (cont’d)

- \( h(x) = f(x)g(x) \)
- To make it uniformly random: add random polynomials with \( p(0) = 0 \)
- Each party picks one: \( h'(x) = f(x)g(x) + \sum_i p_i(x) \)
- Degree reduction is slightly more involved
- Boils down to evaluating a linear form of the shares and opening it to each party
Is it used?

- Research area going back to the early 80’s
- Beautiful results
- Real-world use?
Is it used?

- Research area going back to the early 80’s
- Beautiful results
- Real-world use?
- Not much, yet
Efficiency

- Efficiency is a huge problem
- Time to encrypt 128 bytes using AES?
- Time to sort 16384 integers?
- 3 parties, passive adversary
Efficiency

- Efficiency is a huge problem
- Time to encrypt 128 bytes using AES? 2 seconds [DK10]
- Time to sort 16384 integers?
- 3 parties, passive adversary
Efficiency

- Efficiency is a huge problem
- Time to encrypt 128 bytes using AES? 2 seconds [DK10]
- Time to sort 16384 integers? 3.5 minutes [JKU11]
- 3 parties, passive adversary
Recently, a number of implementation efforts

- FairplayMP
  [http://www.cs.huji.ac.il/project/Fairplay/](http://www.cs.huji.ac.il/project/Fairplay/)
- Viff [http://viff.dk/](http://viff.dk/)
Secure Multiparty Computation Goes Live*

Peter Bogetoft§, Dan Lund Christensen†, Ivan Damgård‡, Martin Geisler‡, Thomas Jakobsen†, Mikkel Krøigaard †, Janus Dam Nielsen‡, Jesper Buus Nielsen‡, Kurt Nielsen†, Jakob Pagter†, Michael Schwartzbach‡, and Tomas Toft††

† Inst. of Food and Resource Economics, University of Copenhagen
‡ Department of Computer Science, University of Aarhus
§Dept. of Economics, Copenhagen Business School
¶The Alexandra Institute
†† CWI Amsterdam and TU/e

Abstract. In this note, we report on the first large-scale and practical application of secure multiparty computation, which took place in January 2008. We also report on the novel cryptographic protocols that were used.
Will it be used?

- Abundance of development environments
- Moore’s law chipping away at performance issue
- Nice security guarantees