Practical Secure Two-Party Computation and Applications

Lecture 2: Private Set Intersection

Estonian Winter School in Computer Science 2016
Overview of this lecture

Special Purpose Protocols
- Arithmetic Circuit
- Boolean Circuit
- Homomorphic Encryption
- GMW
- Yao
- One-Time Pad

Generic Protocols
- Symmetric Crypto
- Public Key Crypto

Private Set Intersection

OT ➔ Symmetric Crypto ➔ One-Time Pad

OT ➔ Public Key Crypto ➔ Symmetric Crypto ➔ One-Time Pad

OT ➔ Homomorphic Encryption ➔ Boolean Circuit ➔ Symmetric Crypto ➔ One-Time Pad

OT ➔ Homomorphic Encryption ➔ Boolean Circuit ➔ Public Key Crypto ➔ Symmetric Crypto ➔ One-Time Pad

OT ➔ Homomorphic Encryption ➔ Boolean Circuit ➔ Public Key Crypto ➔ Symmetric Crypto ➔ One-Time Pad

OT ➔ Homomorphic Encryption ➔ Boolean Circuit ➔ Public Key Crypto ➔ Symmetric Crypto ➔ One-Time Pad
Lecture 3: Private Set Intersection

B. Pinkas, T. Schneider, M. Zohner:  
*Faster Private Set Intersection based on OT extension.*  
In USENIX Security’14.

B. Pinkas, T. Schneider, G. Segev, M. Zohner:  
*Phasing: Private set intersection using permutation-based hashing.*  
In USENIX Security’15.
Private Set Intersection (PSI)
Application: Common Contact Discovery

Check which contacts in address book use the service / are online.
Application: Measuring ad conversion rates

Measure impact of advertisements on shopping in the physical world.
Further Applications

- Secure database join
- Botnet detection
- Cheater detection in online games
- Testing human genomes
- Relationship path discovery
A naïve PSI protocol

Input: $x_1, \ldots, x_n$

$H(x_1), \ldots, H(x_n)$

Input: $y_1, \ldots, y_n$

$H(y_1), \ldots, H(y_n)$

$H(x_i) \overset{?}{=} H(y_j)$, for $0 < i, j < n$

- **Pro**: fast, little communication

- **Con**: can leak privacy of Bob's inputs (guessing attack)
The insecure protocol is used in practice...

**Private contact discovery**
Secure messaging applications TextSecure and Secret

**Measuring ad conversion rates**
Facebook with Datalogix, Epsilon, and Acxiom (companies that have transaction records of loyalty card holders in the US)
Our Contributions

1) Classify and survey PSI literature

2) Optimize existing PSI protocols

3) Develop new highly efficient PSI protocols

4) Compare performance of all schemes
1) Classify and Survey PSI Literature
PSI Classification

a) Public-key Cryptography

b) Generic Secure Computation

c) Oblivious Transfer
a) Public-key Cryptography

Protocols have existed for three decades

Encrypt elements using public-key crypto

Protocols based on public-key cryptography:

• DH-based Protocol [Meadows86], $O(n)$ pk-crypto & comm
• Blind RSA Protocol [CristofaroTsudik10], $O(n)$ pk-crypto & comm
a) Diffie-Hellman-based Protocol [Meadows86]

Input: $x_1, \ldots, x_n$

Choose random $\alpha$

$H(x_1)^\alpha, \ldots, H(x_n)^\alpha$

$H(y_1)^\beta, \ldots, H(y_n)^\beta$

Choose random $\beta$

$H(y_1)^\beta, \ldots, H(y_n)^\beta$

$H(x_1)^\alpha^\beta, \ldots, H(x_n)^\alpha^\beta$

$\overline{X \cap Y} = H(x_i)^\alpha^\beta = H(y_j)^\beta^\alpha$, for $0 < i, j < n$
b) Generic Secure Computation

Represent PSI as Boolean circuit and evaluate it using Yao or GMW

The sort-compare-shuffle (SCS) PSI circuit of [HuangEvansKatz12] needs $O(n\sigma \log n)$ sym-crypto & comm, for $n$ elements of bit-length $\sigma$.
c) Oblivious Transfer

PSI requires PK crypto:
because PSI => OT => PK
(DH-based protocol with 2 exponentiations per element seemed optimal)

But:
OT extension allows to use mostly SK crypto => basis for efficient PSI

Garbled Bloom Filter protocol of [DCW13] for \( n \) elements and symmetric security parameter \( \kappa \):
- \( O(n\kappa) \) sym-crypto
- \( O(n\kappa^2) \) communication
2) Optimize Existing Protocols

- Improve circuit-based PSI of [HEK12] using GMW
  • Multiplexer complexity independent of bit-length
  • Reduce computation / communication by factor 2
  • Also applicable to other functionalities

- Randomize Garbled Bloom filter of [DongChenWen13]
  • Reduce computation by factor 3
  • Reduce communication by factor 4
  • Whole protocol can be parallelized
PSI based on OT

Private Equality Test: $x \sim y$

Private Set Inclusion: $x \subseteq Y$

Private Set Intersection: $X \cap Y$

looks like $O(n^2)$ comparisons, but we can do better...
PSI based on OT (Equality Test)

- **Input:** Alice has $x$, Bob has $y$. **Output:** $x = y$

- **Example:** $x = 001$, $y = 011$

- Bob sends $\lambda$-bit mask $0 \oplus 1 \oplus 1$ to Alice

- Alice computes $0 \oplus 0 \oplus 1$ and compares
PSI based on OT (Set Inclusion)

- **Input:** Alice has $x$, Bob has $Y = \{y_1, \ldots, y_n\}$. **Output:** $x \in Y$

- Run $n$ Private Equality Tests in parallel
  - Alice's OT choices for all $y_1, \ldots, y_n$ are the same
    => same number of OTs, but on longer strings
  - Bob sends $n\lambda$ bits to Alice
Improved Private Set Inclusion [PSSZ15]

**Input:** Alice has $x$, Bob has $Y = \{y_1, \ldots, y_n\}$. **Output:** $x \in Y$

Run the Private Equality Tests as usual
- Inputs to and outputs of OT are the same
  - $\Rightarrow$ same number of OTs, but on short strings
- Difference: Bob sends $n\lambda$ bits to Alice

Break correlations using Random Oracle

\[
\begin{align*}
    y_1 &= 011: \quad H(0 \oplus 1 \oplus 1) \\
    y_2 &= 101: \quad H(1 \oplus 0 \oplus 1) \\
    \quad \ldots \ldots \ldots \ldots \quad \ldots \ldots \ldots \ldots \\
    y_n &= 110: \quad H(1 \oplus 1 \oplus 0)
\end{align*}
\]
Input: Alice has $X = \{x_1, \ldots, x_n\}$, Bob has $Y = \{y_1, \ldots, y_n\}$.
Output: $X \cap Y$.

Run $n$ Private Set Inclusions in parallel
- Requires $n^2$ comparisons, hence not an option
Hashing to Bins

Hash elements to bins to reduce number of comparisons

Example: Alice holds \( X = \{x_1, x_2, x_3\} \), Bob holds \( Y = \{y_1, y_2, y_3\} \)

Cuckoo hashing

\[
\begin{align*}
H(x_1) & \to x_1 \\
H(x_2) & \to x_2 \\
H(x_3) & \to x_3
\end{align*}
\]

Simple hashing

\[
\begin{align*}
H(x_1) & \to O(n) \\
H(x_2) & \to 1 \\
H(x_3) & \to O(\log n / \log \log n)
\end{align*}
\]

Reduces comparisons from \( O(n^2) \) to \( O(n \log n / \log \log n) \)
Hashing Failures

In Cuckoo hashing the mapping of elements to the table can fail which results in either:

- Information leakage about the whole set (if we abort)
- Failure in correctness (if we continue)

We can bound this probability by introducing a stash of size $s$ to which elements are mapped that cause Cuckoo hashing failures:

- Reduces hashing error to $O(n^{-s})$

However, the bounds on the probability are only asymptotic.
We empirically analyzed the hashing failure probability for Cuckoo hashing for different stash sizes $s$ over $2^{30}$ iterations.
Comparison Results

- PSI on $n = 2^{18}$ elements of 32-bit length for 128-bit security on Gbit LAN

PK-Based:
- high run-time for large security parameters
  + best communication
Comparison Results

- PSI on $n = 2^{18}$ elements of 32-bit length for 128-bit security on Gbit LAN

Circuit-Based:
- high run-time & communication
  + easily extensible to arbitrary functions
Comparison Results

- PSI on $n = 2^{18}$ elements of 32-bit length for 128-bit security on Gbit LAN

![Comparison Results Diagram]

OT-Based:
+ good communication and run-time

OT+Hash >10x gap

Naïve

GMW'12

Yao'12

Opt.GMW

Opt.GBF

GBF'13

Blind-RSA'10

DH-FFC'86

DH-ECC'86

Run-time (s)

Communication (MBytes)
Conclusion [PSZ14]

Rule of Thumb:

- OT-based protocols in general case

- DH-based ECC if communication is bottleneck

- Circuit-based protocols for easy extension
Can we do better
to bridge the 10x efficiency gap?
Our Contributions [PSSZ15]

Goal: Make PSI protocols more efficient

Phasing:
PSI using Permutation-based Hashing

Circuit-Phasing:
Improvements on Circuit-based PSI [HEK12]

OT-Phasing:
Improvements on OT+Hashing [PSZ14]
Phasing: PSI using Permutation-based Hashing
Recap: Hashing to Bins [PSZ14]

Hash elements to bins to reduce number of comparisons

Example: Alice holds $X = \{x_1, x_2, x_3\}$, Bob holds $Y = \{y_1, y_2, y_3\}$

Reduces comparisons from $O(n^2)$ to $O(n \log n / \log \log n)$
Permutation-based Hashing [PSSZ15]

In [PSZ14] elements are **compared bit-wise**

- Hence, smaller elements require less overhead

Idea: “hash” elements to a **smaller representation**

- To avoid collisions the birthday paradox states that the hash must be $\lambda + 2 \log(n)$ bits ($= 40 + 2 \times 20 = 80$ bits for $\lambda = 40$, $n = 1 \text{ Mio}$)

Instead: use a permutation to map elements to bins and store a shorter representation

- Used for smaller hash tables [ArbitmanNorSegev10]
- Here: first use in crypto
Permutation-based Hashing

Split \( x = x_L | x_R \) with \( |x_L| = \log n \) bits
(Example: 20|12 bits for \( \sigma=32, \ n=1\text{Mio} \))

Let \( f: [1...2^{|x_R|}] \rightarrow [1...2^{|x_L|}] \) be a random function and \( p(x) = x_L \oplus f(x_R) \)

Hashing is done by storing \( x_R \) in bin \( p(x) \)
(12 bits)

Securely compare \( x_R \) which is only \( \sigma \cdot \log n \) bits long

- Less complexity for comparison
- Larger sets result in even shorter representations 😊
Circuit Phasing
Recap: SCS Circuit of [HuangEvansKatz12]

For \( n \) elements of bitlength \( \sigma \) this circuit takes:

- \( 3n\sigma \log n \) sym. crypto and communication
- \( O(\log(n) \log(\sigma)) \) depth (\#communication rounds in GMW)
Circuit-Phasing

Idea: Use **permutation-based hashing** to hash elements into bins and compare bins on elements with reduced length.

For each bin compare the element of Alice with each element in the same bin of Bob using bit-wise comparison circuit.

Advantages over SCS circuit of [HuangEvansKatz12]:
- Circuit depth (#rounds) independent of set size n
- Same circuit evaluated multiple times allows SIMD (Single Instruction Multiple Data) evaluation
Bins have to be padded to avoid information leakage

In total $O(n \log n / \log\log n)$ comparison circuits

- Per comparison: $O(\sigma \log n)$ sym-crypto & comm
- Total: $O(n (\sigma \log n) \log n / \log\log n)$ sym-crypto & comm
- SCS circuit [HuangEvansKatz12]: $3n\sigma \log n$ sym-crypto & comm
Improvements Circuit-based PSI

PSI on $n = 65,000$ elements of $\sigma=32$-bit length for 128-bit security on Gbit LAN
OT-Phasing
OT-Phasing

Use permutation hashing in OT+Hashing protocol [PSZ14]

Further protocol optimizations:

Use 3 hash functions for the hashing-to-bins routine
  • decreases number of bins by factor 2

Generate only one random string per bin
  • decreases client's work for larger sets
Improvements OT-based PSI

PSI on varying set sizes of different length for $128$-bit security on Gbit LAN
PSI on $n=16$ mio elements of different length for 128-bit security on Gbit LAN
Conclusion [PSSZ15]

Today’s most efficient PSI protocol is based on OT
- Only factor 3 slower than currently used (insecure) solutions
- Throughput for $\sigma=32$ bit:
  - Local: 80k elements/s
  - Cloud: 28k elements/s (Amazon EC2 m3.medium, US – EU)

More efficient and scalable Circuit-based PSI

Code is online on GitHub [http://encrypto.de/PSI](http://encrypto.de/PSI)
EXERCISE 2

Turn Diffie-Helman-based PSI protocol from [Meadows86] into protocol for private set intersection cardinality (PSI-CA) that computes $|X \cap Y|$, i.e., the number of elements that are in the intersection.
EXERCISE 3

Design PSI protocol based on symmetric-key cryptography that uses a trusted third party that does not learn the inputs X or Y in the clear.
Literature


