Private Set Intersection

Benny Pinkas
Bar-Ilan University

(mostly based on joint work with Thomas Schneider, Gil Segev and Michael Zohner)
Protocols for Specific Problems

• Generic protocols can securely compute any functionality
  – Often, the best way to securely compute a function is to represent it as a circuit and apply a generic protocol
  – This is usually the most efficient solution in terms of development time
  – This approach utilizes all improvements that are applied to generic protocols
  – Still, sometimes it is required to achieve better performance than offered by generic protocols
Private Set Intersection (PSI)

Client

Input: \( X = x_1 \ldots x_n \)
Output: \( X \cap Y \) only

Server

Input: \( Y = y_1 \ldots y_n \)
Output: nothing

Other variants exist (e.g., both parties learn output; client learns size of intersection; compute some other function of the intersection, etc.)
Applications

• PSI is a very natural problem
  – Matching
    • Testing human genomes [BBC+11]
    • Proximity testing [NTL+11]
  – Intersection of suspect lists
    • Botnet detection [NMH+10]
    • Contact list discovery (TextSecure, Secret, Medium)
  – Measuring conversion rates for online advertising (Facebook)
This talk

• Survey the major results
• Suggest optimizations based on new observations

• Present new schemes

• Compare the performance of all schemes
  – On the same platform
  – Using the best optimizations that we have
Implementations?

• Generic circuits seem too large for the job
  – More about that later

• PSI is equivalent to oblivious transfer
  – We’ll see PSI protocols based on OT
  – Given PSI we can implement OT:
    – OT: Alice’s input is a bit $b$, Bob’s input is two bits $x_0, x_1$. Alice should learn $x_b$.
    – Implement OT by computing PSI where
      • Alice uses the input set $(b0, b1)$
      • Bob uses the input set $(0x_0, 1x_1)$
A naïve PSI protocol

• A naïve solution:
  – Have A and B agree on a “cryptographic hash function” \( H() \)
  – B sends to A: \( H(y_1), \ldots, H(y_n) \)
  – A compares to \( H(x_1), \ldots, H(x_n) \) and finds the intersection

• Does not protect B’s privacy if inputs do not have considerable entropy

• This is the algorithm used by all applications we are aware of
Preliminaries

• We only consider semi-honest (passive) adversaries

• Why discuss only semi-honest?
  – There are PSI protocols secure against malicious adversaries [FNP04, JL09, HN10, CKT10, FHNP13]
  – These protocols are much less efficient
  – None of them was implemented
PSI secure against malicious adversaries [FHNLP]
Preliminaries – the random oracle model

• In the random oracle model (ROM) a specific function is modeled (in the analysis) as a random function
  – This analysis is very problematic
  – In the theory of crypto, ROM proofs are considered heuristic

• We describe protocols that are based on the ROM
  – There are PSI protocols in the standard model [FNP04], but they are less efficient.
  – We use OT extension
    • Can be based on a non-ROM assumption
    • But the random-OT variant in ROM is even more efficient
Public-key based Protocols
PSI based on Diffie-Hellman

• The Decisional Diffie-Hellman assumption
  – Agree on a group $G$, with a generator $g$.
  – The assumption: for random $a,b,c$
    cannot distinguish $(g^a, g^b, g^{ab})$ from $(g^a, g^b, g^c)$
PSI based on Diffie-Hellman

• The protocol [M86, HFH99, AES03]:

\[ \alpha \]

\[ x_1, \ldots, x_n \]

\[ \beta \]

\[ y_1, \ldots, y_n \]

\[ (H(x_1))^\alpha, \ldots, (H(x_n))^\alpha \]

\[ \text{in parallel} \]

\[ (H(y_1))^\beta, \ldots, (H(y_n))^\beta \]

\[ ((H(y_1))^\beta)^\alpha, \ldots, ((H(y_n))^\beta)^\alpha \]

\[ \text{in parallel} \]

\[ ((H(x_1))^\alpha)^\beta, \ldots, ((H(x_n))^\alpha)^\beta \]

Comparing the two lists

(H is modeled as a random oracle. Security based on DDH)

**Implementation**: very simple; can be based on elliptic-curve crypto; minimal communication.

**What else could we want?**

Secure Computation and Efficiency
Bar-Ilan University, Israel 2015
PSI based on Blind RSA [CT10]

• There is also a PSI protocol based on an RSA variant

• The performance is similar to that of DH based protocols, but
  – In RSA only the owner of the private key does all the hard work ⇒ no advantage in the two parties working in parallel
  – Cannot be based on elliptic curve crypto
PSI based on Blind RSA [CT10]

- Bob chooses an RSA key pair \((N,e),d\)
- Alice chooses random \(r_1,\ldots,r_n\)
  computes \(x_1 \cdot (r_1)^e, \ldots, x_n \cdot (r_n)^e\), and sends to Bob.
- Bob computes and sends
  - \(H((y_1)^d), \ldots, H((y_n)^d)\)
  - \((x_1 (r_1)^e)^d, \ldots, (x_n (r_n)^e)^d\), which equal \((x_1)^d \cdot r_1, \ldots, (x_n)^d \cdot r_n\)
- Alice divides by \(r_i\), applies \(H()\) and compares the lists.
PSI based on Oblivious Polynomial Evaluation [FNP04] (short version)

- (Advantage: proof in the standard model, no ROM)
- Implemented based on additively homomorphic encryption (Paillier, El Gamal).
- Alice generates the polynomial
  \[ P(x) = (x-x_1)(x-x_2) \cdots (x-x_n) = a_n x^n + \cdots + a_1 x + a_0 \]
- Alice sends additively homomorphic encryptions
  \[ E(a_0), E(a_1), \ldots, E(a_n) \]
- \( \forall y_i \) Bob uses these to evaluate and send \( E(P(y_i) \cdot r_i + y_i) \)
- Implementation: \( O(n^2) \) exps. Can be reduced to \( O(n \log \log n) \) using hashing. Too inefficient.
Generic Protocols
A circuit based protocol

• There are generic protocols for implementing any functionality expressed as a Binary circuit
  – GMW, Yao,…

• A naïve circuit uses $n^2$ comparisons of words

• Can we do better?
A circuit based protocol [HEK12]

- A circuit that has three steps
  - **Sort:** merge two sorted lists using a bitonic merging network [Bat68]. Uses \( n \log(2n) \) comparisons.
A circuit based protocol [HEK12]

• A circuit that has three steps
  – **Sort**: merge two sorted lists using a bitonic merging network [Bat68]. Uses $n \log(2n)$ comparisons.
  – **Compare**: compare adjacent items. Uses $2n$ equality checks.
  – **Shuffle**: Randomly shuffle results using a Waxman permutation network [W68], using $\sim n \log(n)$ swappings.

  – **Overall** uses $L \cdot (3n \log n + 4n)$ AND gates. ($L$ is input length)
  • (2/3 of the AND gates are for multiplexers)
Improving Circuit Based PSI

• Initial implementation was done using Yao’s protocol
• GMW uses two OTs per gate; Yao uses four symmetric encryptions.
  – Yao was considered much more efficient.
  – OT extension makes GMW faster than Yao.
Recall the evaluation of multiplication gates in GMW

- Input: $P_1$ has $a_1, b_1$, $P_2$ has $a_2, b_2$.
- $P_1$ outputs $a_1 b_2 + a_j b_2 + s_{1,2}$. $P_j$ outputs $s_{1,2}$.
- $P_j$:
  - Chooses a random $s_{1,2}$
  - Computes the four possible outcomes of $a_1 b_2 + a_2 b_1 + s_{1,2}$, depending on the four options for $P_i$’s inputs.
  - Sets these values to be its input to a 1-out-of-4 OT implemented using two 1-out-of-2 OT2
Improving Circuit Based PSI

• Note that in the PSI circuit $2/3$ of the AND gates are for multiplexers
  – A single bit chooses between two 32 bit inputs
  – For the GMW protocol, instead of independently implementing the OTs for each gate use OTs with inputs that are 32 bit long.
  – It is also possible to implement GMW using random-OT, which is more efficient than regular OT.
Performance of Circuit Based PSI

• We will see that circuit based PSI performs unfavorably compared to other protocols

• The main advantage of circuit based PSI is that it can be used to compute any variant of PSI
  – This can be done by a programmer. Other PSI protocols require a cryptographer in order to apply any change to the computed function.
PSI based on OT

• OT extension is extremely efficient

• Design simple protocols based on OT

• Use OT extension and hashing based constructions to maximize their performance
First step: Private equality test

- Private equality test
  - Input: Alice has $x$, Bob has $y$. Each is $s$ bits long.
  - Output: is $x=y$?
Private equality test

• Alice input: 001    Bob input: 011
Private equality test

• Alice input: 001   Bob input: 011.
• Random OTs

Alice

Bob

<table>
<thead>
<tr>
<th>R_{0,0}</th>
<th>R_{0,1}</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_{1,0}</td>
<td>R_{1,1}</td>
</tr>
<tr>
<td>R_{2,0}</td>
<td>R_{2,1}</td>
</tr>
</tbody>
</table>
Private equality test

- Alice input: 001   Bob input: 011
- Random OTs

Bob sends \( R_{0,0} \oplus R_{1,1} \oplus R_{2,1} \)

Alice computes \( R_{0,0} \oplus R_{1,0} \oplus R_{2,1} \), and compares.

Inputs of length \( s \). Random strings of length \( \lambda \).
Private equality test

• Correctness?

• Security?

• Efficiency?
  – For inputs of length $s$, run $s$ random OTs of $\lambda$ bits strings
  – Bob sends a single $\lambda$ bits string to Alice
  – OTs can be implemented very efficiently using OT extension
Private set inclusion

• Input: Alice has $x$, Bob has $y_1,...,y_n$
• Output: is $x$ in $\{y_1,...,y_n\}$?

• Run $n$ Private Equality Tests in parallel.
  – Alice’s OT choices for all $y_1,...,y_n$ are the same
  – Run only $s$ random OTs of seeds
  – Use a pseudo-random generator to generate from each seed $n$ strings of length $\lambda$ bits (for the corresponding locations in all columns)
  – Send $\lambda n$ bits from Bob to Alice
Private set intersection

• Input: Alice has \{x_1,\ldots,x_n\}, Bob has y_1,\ldots,y_n
• Output: Intersection of \{x_1,\ldots,x_n\} and \{y_1,\ldots,y_n\}

• Run \(n\) Private Set Inclusion protocols
  
  ▸ Total communication is \(n^2 \lambda\) bits
  
  ▸ Communication can be further reduced via hashing
Hashing

• Suppose each party uses a random hash function $H()$, (known to both) to hash its $n$ items to $n$ bins.
  – Then obviously if Alice and Bob have the same item, both of them map it to the same bin.
  – Each bin is expected to have $O(1)$ items
  – The items mapped to the bin can be compared using private equality tests, with $O(\lambda)$ communication.
  – Overall only $O(n\lambda)$ communication.

• The problem
  – Some bins have more items
  – Must hide how many items were mapped to each bin
Hashing

• Solution
  – Pad each bin with dummy items
  – so that all bins are of the size of the most populated bin

• Mapping $n$ items to $n$ bins
  – The expected size of a bin is $O(1)$
  – The maximum size of a bin is whp $O(\log n)$
  – Communication increases by $O(\log n)$ to be $O(n\lambda \log n)$ 😞
Hashing

• Mapping $n$ items to about $n / \ln n$ bins
  – The expected size of a bin is $\approx O(\ln n)$
  – The maximum size of a bin is (whp) the same
  – This is ideal, since we cannot hope to pay less than the expected cost
Other hashing schemes

- Power of two hashing (balanced allocations)
- Cuckoo hashing

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Total #OTs</th>
<th>OT comm.</th>
<th>Overall Comm. (MB) for $n=2^{18}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>No hashing</td>
<td>$ns$</td>
<td>$n^2\lambda$</td>
<td>327,808</td>
</tr>
<tr>
<td>Simple hashing</td>
<td>3.7ns</td>
<td>$n\lambda$</td>
<td>475</td>
</tr>
<tr>
<td>Balanced hashing</td>
<td>2.9ns $\ln\ln n$</td>
<td>$2n\lambda$</td>
<td>939</td>
</tr>
<tr>
<td>Cuckoo hashing</td>
<td>$(2(1+\epsilon)n+\ln n)s$</td>
<td>$(2+\ln n)n\lambda$</td>
<td>276</td>
</tr>
</tbody>
</table>
Input length

• The protocol performs an OT for each bit in the representation of the input items
• Reducing input length ⇒ reducing overhead!
Hashing: can inputs be shorter?

• When mapping \( n \) items to \( n/\ln n \) bins each bin has \( O(\ln n) \) items.
  
  – Birthday paradox: Can hash down input values to \( O(\ln \ln n) \) bits, and expect no collisions in a bin!
  
  – \( N=2^{20} \Rightarrow \ln \ln n = 2.6. \) Wow!!!

  – Unfortunately, to obtain an error probability of \( 2^{-s} \) in the birthday paradox, one needs to represent each item using \( s+\ln \ln n \) bits.

  – For reasonable error probabilities we gain nothing 😞
Permutation based Hashing
[ANS,PSSZ15]

• Hash the values in the bins to a shorter representation while ensuring that different values map to different hashes.
  – Assume we have $2^b$ bins. Input length is $|x| > b$.
  – $x = x_Lx_R$, where $|x_L| = b$.
  – $f$ is a random function whose range is $[1, 2^b]$.
  – $x$ is mapped to bin $x_L \oplus f(x_R)$.
  – Store in that bin the value $x_R$. 
Permutation based Hashing
[ANS,PSSZ15]

• Hash the values in the bins to a shorter representation while ensuring that different values map to different hashes.
  – Assume we have $2^{20}$ bins. Input length is $|x| = 32$.
  – $x = x_L x_R$, where $|x_L| = 20$.
  – $f$ is a random function whose range is $[1,2^{20}]$.
  – $x$ is mapped to bin $x_L \oplus f(x_R)$.
  – Store in that bin 12 bits.
Permutation based Hashing
[ANS,PSSZ15]

• Hashing is Feistel like
  – x is mapped to bin $x_L \oplus f(x_R)$.
  – Store in the bin the value $x_R$.

• If $x,x'$ are mapped to the same bin and store there the same value, then $x=x'$, since
  – Same value: $x_R = x'_R$
  – Same bin: $x_L \oplus f(x_R) = x'_L \oplus f(x'_R)$
Permutation based Hashing

• Great savings!
  – Assume $|x|=32$ and $2^b=2^{20}$ bins.
  – Permutation-based hashing stores in a bin the value $x_R$ of length 12 bits (instead of 32 bits).
  – The overhead of the protocol is reduced to about $12/32 = 37.5\%$ of original cost!
  – Will see performance results in a minute
Generic Computation + Permutation Based Hashing [PSSZ15]

- PSI based on generic secure computation + permutation based hashing
  - Alice maps her inputs to bins (using Cuckoo hashing)
  - Bob maps his inputs to bins
  - They both use permutation-based hashing to reduce the length of their input representations
  - For each bin, they evaluate a circuit that simply compares the elements mapped to it by both parties
Generic Computation + Permutation Based Hashing [PSSZ15]

• Advantages
  – SCS circuits compare all input elements to each other. The new circuits work independently on each bin and use shorter representations.
  – For representation length $\sigma$, the entire new circuit has $n\sigma\log n$ non-xor gates, and a depth of only $\log \sigma$. (SCS has $O(n\sigma')\log n$ gates, and depth $O(\log n \log \sigma')$.)
  – The depth affects number of communication rounds...
  – The circuit is very regular: this reduces memory footprint and enables easy parallelization.
Experiments

• No previous “fair” comparison of all protocols

• We used two desktops in a LAN and cloud settings
  – Inputs are 32 bit long
  – Statistical security parameter $\lambda=40$
  – Symmetric security parameter of 128 bits
## Experiments: run time msec (for $2^{16}$ items)

<table>
<thead>
<tr>
<th>Protocol</th>
<th>local</th>
<th>cloud</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naïve insecure hashing</td>
<td>48</td>
<td>560</td>
</tr>
<tr>
<td>DH ECC</td>
<td>51,400</td>
<td>162,000</td>
</tr>
<tr>
<td>Sorting circuit</td>
<td>47,700</td>
<td>225,500</td>
</tr>
<tr>
<td>Perm-based hash circuit</td>
<td>10,500</td>
<td>42,500</td>
</tr>
<tr>
<td>Perm-based hash + OT</td>
<td>442</td>
<td>3000</td>
</tr>
</tbody>
</table>

Secure Computation and Efficiency
Bar-Ilan University, Israel 2015
Experiments: run time msec (for $2^{16}$ items)

<table>
<thead>
<tr>
<th>Protocol</th>
<th>local</th>
<th>cloud</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naïve insecure hashing</td>
<td>48</td>
<td>560</td>
</tr>
<tr>
<td>DH ECC</td>
<td>51,400</td>
<td>162,000</td>
</tr>
<tr>
<td>Sorting circuit</td>
<td>47,700</td>
<td>225,500</td>
</tr>
<tr>
<td>Perm-based hash circuit</td>
<td>10,500</td>
<td>42,500</td>
</tr>
<tr>
<td>Perm-based hash + OT</td>
<td>442</td>
<td>3000</td>
</tr>
</tbody>
</table>

For $n=2^{20}$ items run time of insecure hashing is **710msec**, and of the Perm-based hash + OT based protocol **4500msec**. Ratio of about **6.3**

For $n=2^{24}$ items the ratio is about **3.4**
Experiments: run time msec (for $2^{16}$ items)

<table>
<thead>
<tr>
<th>Protocol</th>
<th>local</th>
<th>cloud</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naïve insecure hashing</td>
<td>48</td>
<td>560</td>
</tr>
<tr>
<td>DH ECC</td>
<td>51,400</td>
<td>162,000</td>
</tr>
<tr>
<td>Sorting circuit</td>
<td>47,700</td>
<td>225,500</td>
</tr>
<tr>
<td>Perm-based hash circuit</td>
<td>10,500</td>
<td>42,500</td>
</tr>
<tr>
<td>Perm-based hash + OT</td>
<td>442</td>
<td>3000</td>
</tr>
</tbody>
</table>

The permutation-based hashing circuit is about 4-5 times faster than sorting based circuits. Still, circuits are slower than other solutions.
The Diffie-Hellman protocol is slow, but is as far the easiest to implement.

<table>
<thead>
<tr>
<th>Protocol</th>
<th>local</th>
<th>cloud</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naïve insecure hashing</td>
<td>48</td>
<td>560</td>
</tr>
<tr>
<td>DH ECC</td>
<td>51,400</td>
<td>162,000</td>
</tr>
<tr>
<td>Sorting circuit</td>
<td>47,700</td>
<td>225,500</td>
</tr>
<tr>
<td>Perm-based hash circuit</td>
<td>10,500</td>
<td>42,500</td>
</tr>
<tr>
<td>Perm-based hash + OT</td>
<td>442</td>
<td>3000</td>
</tr>
</tbody>
</table>
Communication in MB ($2^{16}$ items)

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Communication in MB (items)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naïve insecure hashing</td>
<td>0.55</td>
</tr>
<tr>
<td>DH ECC</td>
<td>4.5</td>
</tr>
<tr>
<td>Sorting circuit</td>
<td>3,300</td>
</tr>
<tr>
<td>Permutation based circuit</td>
<td>1,050</td>
</tr>
<tr>
<td>Perm-based hash + OT</td>
<td>6.5</td>
</tr>
</tbody>
</table>

The Diffie-Hellman protocol has the best communication. The Perm. based hash + OT protocol is pretty close.
Conclusions

• Set intersection can be efficiently applied to very large input sets

• Different settings require different protocols
  – Run time
  – Communication
  – Generality
  – Development time

• Nice combination of crypto/hashing/systems research.