Session 8: Constructions for Specific Functions of Interest

Benny Pinkas
Bar Ilan University
Extending OT [IKNP]
- Is fully simulatable
- Depends on a non-standard security assumption “correlation robust” functions
  - Modern hash families should have this property
- Security against malicious adversaries is based on the “cut and choose” approach
  - Increases the overhead by a factor of $s$ to reduce cheating probability to $2^{-s}$. 
How efficient is Yao’s protocol?

- Example: the millionaires problem – comparing two N bit numbers

- What’s the overhead?
  - Circuit size is linear in N
  - N oblivious transfers
Other applications

- Two parties. Two large data sets.
- Example applications
  - Computing the Max?
  - Mean?
  - Median?
  - Intersection?
How efficient is **generic secure computation**?

- If the circuit is not too large then generic secure two-party computation is efficient
  - AES (key and plaintext are known to Alice and Bob, respectively) [PSSW09]
    - About 33,000 gates
    - 7/60/1114 sec for semi-honest/covert/malicious

- If the circuit is large: we currently need ad-hoc solutions.
Secure Computation of the Median

k^{th}-ranked element (e.g. median)

- **Inputs:**
  - Alice: $S_A$
  - Bob: $S_B$
  - Large sets of **unique** items ($\in D$).

- **Output:**
  - $x \in S_A \cup S_B$ s.t. $x$ has $k-1$ elements smaller than it.

- **The median**
  - $k = (|S_A| + |S_B|) / 2$

- **Motivation:**
  - Basic statistical analysis of distributed data.
  - E.g., histogram of salaries.
Some information is always revealed

- The $k^{th}$-ranked element reveals some information.
- Suppose $S_A = x_1, \ldots, x_{1000}$ (sorted)
  - Median of $S_A \cup S_B = x_{400}$
- Party A now learns that $S_B$ contains at least 200 elements smaller than $x_{400}$
- But she shouldn’t learn more
Using a generic solution...

- The Problem:
  - The size of a circuit for computing the $k^{th}$ ranked element is at least linear in $k$.
  - For the median, $k$ is in the same order as the size of the inputs.
  - Generic constructions using circuits [Yao,...] have communication complexity which is linear in the circuit size, and therefore in $k$. 
An (insecure) two-party median protocol

$S_A$  
\begin{array}{ccc}
L_A & m_A & R_A \\
\end{array}

$m_A < m_B$

$S_B$  
\begin{array}{ccc}
L_B & m_B & R_B \\
\end{array}

$L_A$ lies below the median, $R_B$ lies above the median. $|L_A| = |R_B|$

New median is same as original median!

Recursion $\rightarrow$ Need $\log n$ rounds  
(assume each set contains $n=2^i$ items)
A Secure two-party median protocol

Secure comparison (e.g. a small circuit)

A finds its median \( m_A \)

B finds its median \( m_B \)

\( m_A < m_B \)

YES

A deletes elements \( \leq m_A \).
B deletes elements \( > m_B \).

NO

A deletes elements \( > m_A \).
B deletes elements \( \leq m_B \).
An example

Median found!!
Proof of security

A

median

B

m_A > m_B

m_A < m_B

m_A < m_B

m_A > m_B

m_A < m_B
Proof of security

- This is a proof of security for the case of semi-honest adversaries.

- Security for malicious adversaries is more complex.
  - The protocol must be changed to ensure that the parties’ answers are consistent with some input.
  - Also, the comparison of the medians must be done by a protocol secure against malicious adversaries.
Now, compute the median of two sets of size $k$.

Size should be a power of 2.

median of new inputs = $k^{th}$ element of original inputs
Hiding size of inputs

- Can search for \(k^{th}\) element without revealing size of input sets.
- However, \(k=n/2\) (median) reveals input size.
- Solution: Let \(S=2^i\) be a bound on input size.

Median of new datasets is same as median of original datasets
A Protocol secure against malicious adversaries

- The parties can choose arbitrary inputs to the comparisons.
- For example,
  - In Step 1 claim that $m_A = 100$, and be told that $m_A < m_B$ (therefore A must remove all items $\leq m_A$).
  - In step 2 claim that $m_A = 10$...

- We change the protocol so that even if input values are chosen adaptively during the protocol, they correspond to a valid input that can be sent to the TTP.
Protocol secure against malicious adversaries

- The modified protocol:
  - Initialize bounds $L_A = L_B = -\infty$, $U_A = U_B = \infty$.
  - Each comparison protocol must be secure against malicious parties and verify that
    - $L_A < m_A < U_A$
    - $L_B < m_B < U_B$
  - If the verification succeeds, then
    - If $m_A \geq m_B$ then set $U_A = m_A$ and $L_B = m_B$
    - Otherwise set $L_A = m_A$ and $U_B = m_B$

The bounds ensure that $m_A$ and $m_B$ are consistent with previous inputs.
Implementing the secure computation

- The bounds $L_A, L_B, U_A, U_B$ must not be revealed to any party, but rather be internal values of the secure computation.
- The secure computation is run in phases, where each phase must pass updated values of $L_A, L_B, U_A, U_B$ to the next phase.
- Can be implemented using reactive computation
- Or, in a simpler way…
Implementing the secure computation

- The circuit is composed of layers.
- Each layer provides an external output, and has internal wires going into the next layer.
Implementing reactive computation

- In each layer
  - provide A with
    - Shares of $L_A, U_A, L_B, U_B$
    - MACs of B’s shares of these values
  - provide B with
    - Shares of $L_A, U_A, L_B, U_B$
    - MACs of A’s shares of these values

- In the next level
  - A and B inputs these values.
  - The circuit checks the MACs, and reconstructs $L_A, U_A, L_B, U_B$ from shares.
Proof of security

- Must show that for every adversary $A'$ in real model there is a simulator $A''$ in the ideal model, etc…
- The operation of $A'$ in the real model can be visualized as following a path in a binary tree.
Proof of security

- If A’ does not provide a legitimate input to a comparison (namely \( m_A \not\in (L_A, U_A) \)) then the simulator aborts.
- Assume that the random input of A’ is known to the simulator, and therefore A’ is deterministic.
 Proof of security

- The simulator runs the protocol with A’, rewinding over all execution paths in the tree.
- Learns the inputs of A’ to all comparisons. The inputs to the leaves correspond to the sorted input of A’.
- The simulator sends this input to TTP. Based on the result, it simulates the real execution path with A’.
The multi–party case

- **Input:** Party $P_i$ has set $S_i$, $i=1..n.$
  
  (all values $\in [a,b]$, where $a$ and $b$ are known)

- **Output:** $k^{th}$ element of $S_1 \cup ... \cup S_n$

- **Protocol (binary search):** Set $m = (b-a)/2$. Repeat:
  
  - $P_i$ uses the following input for a secure computation:
    
    $L_i = \# $ elements in $S_i$ smaller than $m$.
    
    $B_i = \# $ times $m$ appears in $S_i$.

  - The following is computed securely:
    
    - If $\Sigma L_i \geq k$, set $b=m$, $m=(m-a)/2$, else
    - If $\Sigma (L_i + B_i) \geq k$, stop. $k^{th}$ element is $m$.
    - Otherwise, set $a=m$, $m = m+(b-m)/2$. 

Secure Computation and Efficiency
Bar–Ilan University, Israel 2011
Conclusion

- Efficient secure computation of the median.
  - Two-party: $\log k$ rounds $\times O(\log D)$
  - Multi-party: $\log D$ rounds $\times O(\log D)$
  - Very close to the communication complexity lower bound of $\log D$ bits.

- Malicious case is efficient too.
  - Do not use generic tools.
  - Instead, implement simple consistency checks.
Private matching and set intersection

M. Freedman, K. Nissim and B. Pinkas, Efficient Private Matching and Set Intersection, Eurocrypt'04.
The Scenario

Input:

\[ X = x_1 \ldots x_n \]

Output:

\[ X \cap Y \text{ only} \]

- Shared interests (research, music)
- Credit rating
- Sharing intelligence between agencies (IARPA)
- Dating
- Genetic compatibility, etc

\[ Y = y_1 \ldots y_n \]

nothing
Implementation by a circuit?

- **Trivial circuit compares each** $(x_i, y_j)$ **pair**
  - $O(n^2)$ circuit size

- **A more advanced circuit:**
  - Sort the union of the two sets, using a sorting network.
  - If $x_i = y_j$ these two values will become adjacent.
  - Scan and search for identical adjacent values
  - $O(n \log n)$ circuit size (with huge constant [AKS])
Basic tool: Additively homomorphic encryption

- Public key encryption, such that
  - Given $E(x)$ it is possible to compute, without knowledge of the secret key, the value of $E(c \cdot x)$, for every $c$.
  - Given $E(x)$ and $E(y)$, it is possible to compute $E(x+y)$.

- We will use the notation
  - $E(x) \cdot E(y) = E(x+y)$
  - $E(x)^c = E(c \cdot x)$

- Applications
  - Voting
  - Many cryptographic protocols, such as keyword search, oblivious transfer...
Background on homomorphic encryption

- “Standard” public key encryption schemes support Homomorphic operations with relation to multiplication
  - **RSA**
    - Public key: N, e. Private key: d.
    - $E(m) = m^e \mod N$
    - $E(m_1) \cdot E(m_2) = E(m_1 \cdot m_2)$
  - **El Gamal**
    - Public key: p (or a similar group), $y = g^x$. Private key: x.
    - $E(m) = (g^r, y^r m)$
    - $E(m_1) \cdot E(m_2) = E(m_1 \cdot m_2)$
Background on additively homomorphic encryption

- **Modified El Gamal**
  - $E(m) = (g^r, y^r g^m)$
  - $E(m_1) \cdot E(m_2) = (g^r, y^r g^{m_1 + m_2}) = E(m_1 + m_2)$
  - Decryption reveals $g^{m_1 + m_2}$
  - Computing $m_1 + m_2$ is possible if $m_1 + m_2$ is small

- **Paillier’s cryptosystem**
  - Based on composite residuocity classes
  - Works in the group $\mathbb{Z}_{n^2}^*$, where $n = pq$
  - “Public-Key Cryptosystems Based on Composite Degree Residuosity Classes”, Pascal Paillier, Eurocrypt’99.
The protocol (semi-honest case)

- Client (C) defines a polynomial of degree $n$ whose roots are her inputs $x_1, \ldots, x_n$

$$P(y) = (x_1 - y)(x_2 - y) \cdots (x_k - y) = a_0 + a_1 y + \cdots + a_k y^k$$

- C sends to server (S) homomorphic encryptions of polynomial’s coefficients

$$\text{Enc}(a_0), \ldots, \text{Enc}(a_k)$$
The protocol

Note that
- \( \text{Enc}(P(y)) = \text{Enc}(a_0 + a_1 \cdot y^1 + \ldots + a_k \cdot y^k) = \text{Enc}(a_0) \cdot \text{Enc}(a_1) \cdot y \cdot \text{Enc}(a_2) \cdot y^2 \cdot \ldots \cdot \text{Enc}(a_k) \cdot y^k \)
- Therefore \( \forall y \), server can compute \( \text{Enc}(P(y)) \)

The operation of the server
- \( \forall y_j \), choose random \( r_j \) and compute \( \text{Enc}(r_j \cdot P(y_j) + y_j) \)
- This equals \( \text{Enc}(y_j) \) if \( y_j \in X \), and is random otherwise.
- S sends (permuted) results back to C
- C decrypts and learns \( X \cap Y \)
Variants of the basic protocol

- The server computes $\text{Enc}(r_j \cdot P(y_j) + 1)$
  - This equals $\text{Enc}(1)$ if $y_j \in X$, and is random otherwise.
  - The client decrypts, counts the number of 1’s and learns $|X \cap Y|$.

- A different variant enables to compute whether $|\text{intersection}| > \text{threshold}$.
Security (semi-honest)

- **Client’s privacy**
  - Server only sees semantically-secure enc’s
  - We can simulate server’s view by sending it enc’s of arbitrary values.

- **Server’s privacy**
  - Client can simulate her view in the protocol, given the output $X \cap Y$ alone:
    - Compute the enc’s of items in $X \cap Y$ and of random items, and receive them in random order.
Communication is $O(n)$
- C sends $n$ coefficients
- S sends $n$ evaluations of polynomial

Computation
- Client encrypts and decrypts $n$ values
- Server:
  - $\forall y \in Y$, computes $\text{Enc}(r \cdot P(y) + y)$, using $n$ exponentiations
  - Total of $O(n^2)$ exponentiations
Inputs typically from a “small” domain of $D$ values. Represented by $\log D$ bits (say, 20)

Use Horner’s rule to compute polynomial:
- $P(y) = a_0 + y(a_1 + \ldots y(a_{n-1} + ya_n) \ldots)$ instead of $P(y) = a_0 + a_1 y + a_{n-1} y^{n-1} + a_n y^n$
- Now, exponents are only $\log D$ bits
- Overhead of exponentiation is linear in $|\text{exponent}|$

→ Improvement by factor of $|\text{modulus}|/\log D$, e.g., $1024/20 \approx 50$
C uses $H(\cdot)$ to hash inputs to $B$ bins ($H$ indep. of inputs)
Let $M$ bound max # of items in a bin.
Client defines $B$ polynomials of deg $M$.
Each poly encodes $x$’s mapped to its bin.
C sends B polynomials and H to server.

For every y, S computes H(y) and evaluates the corresponding poly (of degree M)

\[ \forall y \in Y, \quad i \leftarrow H(y), \quad r \leftarrow \text{rand} \]
\[ \text{Enc}( r \cdot P_i(y) + y ) \]
Overhead with Hashing

- Communication: $B \cdot M$
- Server: $n \cdot M$ short exp’s, $n$ full exp’s
  \[
  (P(y)) \quad (r \cdot P_i(y) + y)
  \]
- How large should $M$ be?
- Simple hashing:
  - If the number of bins is $B=n$, then $M=O(\log n)$
  - Therefore
    - Communication $O(n \log n)$
    - Server computation $O(n \log n)$
  - Can do better…
Balanced allocations [ABKU]:

- \( H = (h_1, h_2) \): Choose two bins, map item \( y \) to the less occupied bin among \( h_1(y), h_2(y) \).

- It was shown that for \( B = n/\ln \ln n \) the maximum bin size is \( M = O(\ln \ln n) \).

- Communication is \( BM = O(n) \).

- Server: \( n \ln \ln n \) short exp, \( n \) full exp. (in practice \( \ln \ln n \leq 5 \)).

- Client must check results in two bins.
Overhead with Hashing

- Cuckoo hashing [Pagh–Rodler]
- Map $n$ items to $B \approx 2n$ bins of size $M=1$, or to a small stash (of size, say, 3).
- Each item $y$ is found in either $h_1(y)$ or $h_2(y)$, or in the stash.
  - Details of the construction are omitted
- Communication, and server work are only $O(n)$
Actual run times

- Asymptotic run time of server with random hashing/balanced allocations/Cuckoo hashing is \(\frac{n\log n}{n\log \log n}/n\), respectively.
- For \(n=10,000\), actual run times were 48/69/65 seconds, respectively.
- ????????
Actual run times – what happened?

- Server computes \( E(r \cdot P(y) + y) \)
- The overhead of multiplying by \( r \) is independent of the degree. It is also a full exponentiation.
  - Experiments showed the overhead of evaluating \( E(r \cdot P(y) + y) \) to be linear in \( d + 6.5 \)
- In the different methods S evaluates 1/2/3 polynomials, of degree \( \log n / \log \log n / O(1) \).
- Simple hashing is better since it evaluates a single polynomial.
- The other schemes are better only for larger values of \( n \).