CMP_SC 8001 - Introduction to Secure Multiparty Computation

Fundamental MPC Protocols - Part 3

Wei Jiang

Department of Electrical Engineering and Computer Science
University of Missouri

Wei Jiang - http://faculty.missouri.edu/wjiang/
Outline

1. Constant-Round MPC
   - Protocol Overview
   - GC Generation Circuit
   - The BMR Protocol

2. Oblivious Transfer
   - Public Key-based OT
   - 1-out-on-N OT
   - Efficient OT Extension

3. Private Set Intersection
   - Oblivious PRF
   - Cuckoo Hashing
   - PSI from OPRF

Wei Jiang - http://faculty.missouri.edu/wjiang/
Outline

1. Constant-Round MPC
   - Protocol Overview
   - GC Generation Circuit
   - The BMR Protocol

2. Oblivious Transfer
   - Public Key-based OT
   - 1-out-on-N OT
   - Efficient OT Extension

3. Private Set Intersection
   - Oblivious PRF
   - Cuckoo Hashing
   - PSI from OPRF
After Yao’s (two-party) GC protocol was proposed, several SMC protocols appeared:

- Goldreich-Micali-Wigderson (GMW) (Goldreich, 2004; Goldreich et al., 1987)
- Ben-Goldwasser-Wigderson (BGW) (Ben-Or et al., 1988)
- Chaum-Crepeau-Damgård (CCD) (Chaum et al., 1988)

All of these protocols have a number of rounds linear in the depth of the circuit $C$ computing $F$.

BMR protocol (Beaver et al., 1990) runs in a constant (in the depth of $C$) number of rounds, while achieving security against any $t < n$ number of corruptions among $n$ parties.
The BMR protocol adapts the main idea of Yao’s GC to a multi-party setting.

GC is chosen as a starting point due to its round-efficiency.

The basic BMR idea is to perform a distributed GC generation, so that no proper subsets of all parties know GC secrets:
- the label assignment and correspondence.
Encoding GC Generation as a Circuit

- GC generation can be represented as a circuit $C_{\text{GEN}}$
- Using **MPC**, $C_{\text{GEN}}$ can be evaluated securely to produce GC
  - This is possible by first generating (in parallel) all wire labels independently, and
  - then independently generating garbled gate tables
Encoding GC Generation as a Circuit

- Because of parallel processing for all gates/wires, the GC generation is independent of the depth of the circuit.
- As a result, the GC generation circuit $C_{\text{GEN}}$ is constant-depth for all computed circuits $C$ (once the security parameter $\kappa$ is fixed).
- Even if the parties perform MPC evaluation of $C_{\text{GEN}}$ that depends on the depth of $C_{\text{GEN}}$, the overall BMR protocol will still have constant rounds overall.
The MPC output, the GC produced by securely evaluating CGEN, may be delivered to a designated player, say $P_1$, who will then evaluate it similarly to Yao’s GC.

But, the challenge is how to deliver the active input labels to $P_1$.

There are several ways how this may be achieved, depending on how exactly the MPC GC generation proceeded.

It is conceptually simplest to view this as part of the GC generation computation.
In concrete terms, the above approach can lead to high cost:

- requiring the garbled row encryption function (instantiated as a PRF or hash function) evaluation inside MPC

Several protocols were proposed, which allow the PRF/hash evaluation to be extracted from inside the MPC

- instead be done locally by the parties while providing the output of PRF/hash into the MPC

The underlying idea of such an approach is to assign different portions of each label to be generated by different players.
The \( C_{\text{GEN}} \) Circuit

- A wire \( w_a \)'s labels \( w_a^v \) are a concatenation of sub-labels \( w_{a,j}^v \), each generated by \( P_j \).

- Then, for a gate \( G_i \) with input labels \( w_{a}^v, w_{b}^v \) and the output label \( w_{c}^v \), the garbled row corresponding to input values \( v_a, v_b \) and output value \( v_c \) can simply be:

\[
e_{v_a, v_b} = w_{c}^v \bigoplus_{j=1 \ldots n} \left( F \left( i, w_{a,j}^v \right) \oplus F \left( i, w_{b,j}^v \right) \right)
\]

- \( F : \{0, 1\}^\kappa \mapsto \{0, 1\}^{n \cdot \kappa} \) is a PRG extending \( \kappa \) bits into \( n \cdot \kappa \) bits.
The $C_{\text{GEN}}$ Circuit

- The generation of the garbled table row is almost entirely done locally by each party.

- Each $P_j$ computes $F(i, w_{a,j}^v) \oplus F(i, w_{b,j}^v)$ and submits it to the MPC that simply xors all the values to produce the garbled row.

- **Security violation:**
  - The GC evaluator $P_1$ will reconstruct active labels.
  - The knowledge of its own contributed sub-labels allows it to identify which plaintext value the active label corresponds to.
Each player $P_j$ adds a **flip** bit $f_{a,j}$ to each wire $w_a$

The xor of the $n$ flip bits, $f_a = \bigoplus_{j=1}^{n} f_{a,j}$, determines which plaintext bit $v$ corresponds to the wire label $w_a$

The flip bits will be an additional input into the garbling MPC

With the addition of the flip bits, no subset of players will know the wire flip bit

Hence, this additional randomization can prevent the evaluator from inferring the plaintext value from its active sub-labels
BMR - Setting

Parameters:

- Boolean circuit $C$ implementing function $F$
- $F : \{0, k\}^\kappa \mapsto \{0, 1\}^{n \cdot \kappa + 1}$ is a pseudo-random generator (PRG)

Players:

- $P_1, \ldots, P_n$ with inputs $x_1, \ldots, x_n \in \{0, 1\}^k$
For each wire $w_i$ of $C$, each $P_j$ randomly chooses wire sub-labels, $w_{i,j}^b = (k_{i,j}^b, p_{i,j}^b) \in_R \{0, 1\}^{\kappa+1}$, such that $p_{i,j}^b = 1 - p_{i,j}^{1-b}$, and flip-bit shares $f_{i,j} \in_R \{0, 1\}$.

For each wire $w_i$, $P_j$ locally computes its underlying-MPC input:

$$l_{i,j} = [F(w_{i,j}^0), F(w_{i,j}^1), p_{i,j}^0, f_{i,j}]$$
For each gate $G_i$ of $C$, in parallel, all players participate in $n$-party MPC to compute the garbled table based on a GC generation function, $C_{\text{GEN}}$

The input of $C_{\text{GEN}}$: all players’ inputs $x_1, \ldots, x_n$ as well as pre-computed values $I_{i,j}$
Assume $G_i$ is a 2-input Boolean gate implementing function $g$, with input wires $w_a$, $w_b$ and output wire $w_c$.

2. Compute pointer bits:
   \begin{itemize}
   \item $p^0_a = \bigoplus_{j=1}^{n} p^0_{a,j}$ and $p^1_a = 1 - p^0_a$
   \item $p^0_b = \bigoplus_{j=1}^{n} p^0_{b,j}$ and $p^1_b = 1 - p^0_b$
   \item $p^0_c = \bigoplus_{j=1}^{n} p^0_{c,j}$, and $p^1_c = 1 - p^0_c$
   \end{itemize}

3. Similarly compute flip bits:
   \begin{itemize}
   \item $f_a = \bigoplus_{j=1}^{n} f_{a,j}$
   \item $f_b = \bigoplus_{j=1}^{n} f_{b,j}$
   \item $f_c = \bigoplus_{j=1}^{n} f_{c,j}$
   \end{itemize}
Create $G_i$’s garbled table: for each of $2^2$ possible combinations of $G_i$’s input values $v_a, v_b \in \{0, 1\}$, set

$$e_{v_a,v_b} = w_c^{v_c \oplus f_c} \bigoplus_{j=1 \ldots n} \left( F \left( i, w_{a,j}^{v_a \oplus f_a} \right) \oplus F \left( i, w_{b,j}^{v_b \oplus f_b} \right) \right)$$

where $w_c^0 = w_{c,1}^0 || \ldots || w_{c,n}^0$ and $w_c^1 = w_{c,1}^1 || \ldots || w_{c,n}^1$

Sort entries $e$ in the table: placing entry $e_{v_a,v_b}$ in position $(p_a, p_b)$

Output to $P_1$ the computed garbled tables, as well as active wire labels inputs of $C$, as selected by players’ inputs $x_1, \ldots, x_n$ and the flip bits $f_a, f_b, f_c$
Outline

1. Constant-Round MPC
   - Protocol Overview
   - GC Generation Circuit
   - The BMR Protocol

2. Oblivious Transfer
   - Public Key-based OT
   - 1-out-on-N OT
   - Efficient OT Extension

3. Private Set Intersection
   - Oblivious PRF
   - Cuckoo Hashing
   - PSI from OPRF
Oblivious Transfer (OT)

- Oblivious Transfer is an essential building block for secure computation protocols, and an inherently asymmetric primitive.
- Beaver (1996) showed a batched execution of OT only needs a small number of public key operations.
- Beaver’s construction was non-black-box in the sense that a PRF needed to be represented as a circuit and evaluated as MPC.
- Thus, Beaver’s result was mainly of theoretical interest.
- Ishai et al. (2003) proposed an extremely efficient batched OT which only required $\kappa$ of public key operations for the entire batch and two or three hashes per OT.
Public Key-based OT

Parameters:

- Two parties: Sender $S$ and Receiver $R$
- Input: $S$ has two secrets $x_0, x_1 \in \{0, 1\}^n$, and $R$ has a selection bit $b \in \{0, 1\}$
- Output: $\langle S, \bot \rangle, \langle R, x_b \rangle$

Security guarantee:

- Semi-honest
Public Key-based OT - Main Steps

1. $\mathcal{R}$ generates a public-private key pair $sk, pk$, and samples a random key $pk'$ from the public key space.
   - If $b = 0$, $\mathcal{R}$ sends $(pk, pk')$ to $S$.
   - If $b = 1$, $\mathcal{R}$ sends $(pk', pk)$ to $S$.

2. $S$ receives $(pk_0, pk_1)$ and sends back to $\mathcal{R}$ two encryptions $e_0 = Enc_{pk_0}(x_0)$ and $e_1 = Enc_{pk_1}(x_1)$.

3. $\mathcal{R}$ receives $e_0, e_1$ and decrypts the ciphertext $e_b$ using $sk$ ($\mathcal{R}$ is unable to decrypt the second ciphertext as it does not have the corresponding secret key).
The security of the construction assumes the existence of public-key encryption with the ability to sample a random public key without obtaining the corresponding secret key.

The scheme is secure in the semi-honest model.

$S$ only sees the two public keys sent by $R$, so cannot predict with probability better than $\frac{1}{2}$ which key was generated without the secret key.

$R$ sees two encryptions and has a secret key to decrypt only one of them.
Note that this semi-honest protocol provides no security against a malicious receiver.

\( R \) can simply generate two public-private key pairs \((sk_0, pk_0)\) and \((sk_1, pk_1)\) and send \((pk_0, pk_1)\) to \( S \).

Then it decrypts received ciphertexts to learn both \( x_1 \) and \( x_2 \).
Bob has two messages $m_0$ and $m_1$, and Alice has an index or a bit $b$, and Alice wants to retrieve $m_b$, without Bob learning $b$.

In addition, Bob wants to make sure that Alice only receives one of the two messages.

The following protocol was proposed by Even, Goldreich and Lempel, 1985.
EGL OT Protocol

1. Bob sends $N$, $e$, $x_0$ and $x_1$ to Alice, where $x_0$ and $x_1$ are randomly chosen from $\{1, \ldots, N-1\}$, where $e$ is the public key of RSA and Bob knows the private key $d$.

2. Alice randomly selects $k \in \{1, \ldots, N-1\}$, and sends $v = (k^e \mod N) \oplus x_b$ to Bob.

3. Bob computes $k_0 = (v \oplus x_0)^d \mod N$ and $k_1 = (v \oplus x_1)^d \mod N$, and sends $z_0 = m_0 \oplus k_0$ and $z_1 = m_1 \oplus k_1$ to Alice.

4. Alice computes $m_b = z_b \oplus k$. 
1-out-of-\(N\) OT

- Bob has \(n\) messages \(m_1, \ldots, m_n\), and Alice has an index \(i\) (\(1 \leq i \leq n\)), and Alice wants to retrieve \(m_i\), without Bob learning \(i\).
- In addition, Bob wants to make sure that Alice only receives one of the \(n\) messages.
1-out-of-$N$ OT

- $1-n$ OT can be built on top of a number of 1-2 OTs
- Suppose $n = 2^l$ and $m_i \in \{0, 1\}^s$
- The follow OT protocol was proposed by Naor and Pinkas 1999
NP OT Protocol

Bob prepares $l$ random pairs of keys

$$(K_0^0, K_1^1), (K_0^0, K_2^1), \ldots, (K_0^0, K_l^1)$$

where $K_j^b$ ($1 \leq j \leq l$ and $b \in \{0, 1\}$) is a $t$-bit key to a pseudo-random function $F_{K_j^b} : \{0, 1\}^s \rightarrow \{0, 1\}^s$. For any $1 \leq i \leq n$, let $\langle i_1, i_2, \ldots, i_l \rangle$ be the bits of $i$, and compute

$$y_i = m_i \oplus \bigoplus_{j=1}^{l} F_{K_j^b}(i)$$
NP OT Protocol

2 Via $l$ 1-2 OTs, the $j^{th}$ 1-2 OT is performed on $(K_j^0, K_j^1)$, Alice retrieves $K_1^{i_1}, K_2^{i_2}, \ldots, K_l^{i_l}$ for her index $i$ denoted by $\langle i_1, i_2, \ldots, i_l \rangle$.

3 Bob sends $y_1, y_2, \ldots, y_n$ to Alice

4 Alice retrieves

$$m_i = y_i \oplus \bigoplus_{j=1}^{l} F_{K_j^i}(i)$$
The first simple protocol requires one public key operation for both the sender and receiver for each selection bit.

As used in a Boolean circuit-based MPC protocol such as Yao’s GC, it is necessary to perform an OT for each input bit of the party executing the circuit.

For GMW, evaluating each AND gate requires an OT.

Hence, several works have focused on reducing the number of public key operations to perform a large number of OTs.
As discussed in Section 3.1, the GC protocol for computing a circuit $C$ requires $m$ OTs, where $m$ is the number of input bits provided by $P_2$.

Following the OT notation, we call $P_1$ (the generator in GC) the sender $S$, and $P_2$ (the evaluator in GC) the receiver $R$.

$S$’s input will be $m$ pairs of secrets $(x_1^0, x_1^1), \ldots, (x_m^0, x_m^1)$, and $R$’s input will be $m$-bit selection string $r = (r_1, \ldots, r_m)$. 

Our goal is to use a small number $k$ of base-OTs, plus only symmetric-key operations, to achieve $m \gg k$ effective OTs.

$k$ depends on the computational security parameter $\kappa$.

Below we describe the OT extension by Ishai et al. (2003) that achieves $m$ 1-out-of-2 OT of random strings, in the presence of semi-honest adversaries.
The IKNP Protocol

- Suppose the receiver $R$ has choice bits $r \in \{0, 1\}^m$
- $R$ chooses two $m$ by $k$ matrices ($m$ rows, $k$ columns) $T$ and $U$
- Let $t_j, u_j \in \{0, 1\}^k$ denote the $j$-th row of $T$ and $U$ respectively
- $T$ is chosen at random, and $U$ is derived as follows:

$$t_j \oplus u_j = r_j \cdot 1^k \overset{\text{def}}{=} \begin{cases} 1^k & \text{if } r_j = 1 \\ 0^k & \text{if } r_j = 0 \end{cases}$$
The IKNP Protocol

- The sender $S$ chooses a random string $s \in \{0, 1\}^k$, and the parties engage in $k$ instances of 1-out-of-2 string-OT:
  - with their roles reversed, to transfer to sender $S$ the columns of either $T$ or $U$
  - depending on the sender’s bit $s_i$ in the string $s$ it chose

- In the $i$-th OT, $R$ provides inputs $t^i$ and $u^i$, where these refer to the $i$-th column of $T$ and $U$ respectively

- $S$ uses $s_i$ as its choice bit and receives output $q^i \in \{t^i, u^i\}$
The IKNP Protocol

- Note that these are OTs of strings of length $m \gg k$, and the length of OT messages is easily extended, e.g.,
  - encrypting and sending the two $m$-bit long strings, and using OT on short strings to send the right decryption key

- Let $Q$ denote the matrix obtained by the sender, whose columns are $q^i$, and let $q_j$ denote the $j$-th row:

$$q_j = t_j \oplus [r_j \cdot s] \overset{\text{def}}{=} \begin{cases} t_j & \text{if } r_j = 0 \\ t_j \oplus s & \text{if } r_j = 1 \end{cases} \quad (3.2)$$
The IKNP Protocol

- Let $H$ be a Random Oracle (RO), and $S$ can compute two random strings $H(q_i)$ and $H(q_i \oplus s)$ of which $R$ can compute only one, via $H(t)$ of $R$’s choice.

- According to the previous equation, it is obvious that $t$ equals either $q_i$ or $q_i \oplus s$, depending on $R$’s choice bit $r$.

- Note that $R$ has no information about $s$, so intuitively it can learn only one of the two random strings $H(q_i)$ and $H(q_i \oplus s)$.

- Hence, each of the $m$ rows of the matrix can be used to produce a single 1-out-of-2 OT of random strings.
The IKNP Protocol

- Recall $S$ has $m$ pairs of secrets $(x_0^0, x_1^1), \ldots, (x_m^0, x_m^1)$, and $R$ has $m$-bit selection string $r = (r_1, \ldots, r_m)$.

- After $k$ OTs, $S$ has $q_1, \ldots, q_m$:

  $$q_j = \begin{cases} 
  t_j & \text{if } r_j = 0 \\
  t_j \oplus s & \text{if } r_j = 1 
  \end{cases}$$

- $S$ and $R$ perform the following steps
The IKNP Protocol

1. For $j \in \{1, \ldots, m\}$, $S$ computes:
   - $e_j^0 = H(q_j) \oplus x_j^0$
   - $e_j^1 = H(q_j \oplus s) \oplus x_j^1$

   Sends $\left\{ (e_j^0, e_j^1) \right\}_{j \in \{1, \ldots, m\}}$ to $R$

2. $R$ receives $\left\{ (e_j^0, e_j^1) \right\}_{j \in \{1, \ldots, m\}}$ from $S$, and computes:
   - $x_j^{r_j} = e_j^{r_j} \oplus H(t_j)$, for $j \in \{1, \ldots, m\}$
Selecting Values for $k$

- The parameter $k$ determines the number of base OTs and the overall cost of the protocol.
- The IKNP protocol sets $k = \kappa$. 
Outline

1. Constant-Round MPC
   - Protocol Overview
   - GC Generation Circuit
   - The BMR Protocol

2. Oblivious Transfer
   - Public Key-based OT
   - 1-out-on-N OT
   - Efficient OT Extension

3. Private Set Intersection
   - Oblivious PRF
   - Cuckoo Hashing
   - PSI from OPRF
Private Set Intersection

- The goal of private set intersection (PSI) is to enable a group of parties to jointly compute the intersection of their input sets, without revealing any other information about those sets (other than upper bounds on their sizes).

- Although protocols for PSI have been built upon generic MPC (Huang et al., 2012a), more efficient custom protocols can be achieved by taking advantage of the structure of the problem.

- We present a PSI protocol of Pinkas et al. (2015), which heavily uses Oblivious PRF (OPRF) as a subroutine.
Oblivious PRF

- OPRF is an MPC protocol which allows two players $P_1$ and $P_2$ to evaluate a PRF $F$ where
  - $P_1$ holds the PRF key $k$, and $P_2$ holds the PRF input $x$
  - $P_2$ gets $F_k(x)$

- We now describe the Pinkas-Schneider-Segev-Zohner (PSSZ) construction (Pinkas et al., 2015) building PSI from an OPRF

- For concreteness, we describe the parameters used in PSSZ when the parties have roughly the same number $n$ of items
The protocol relies on Cuckoo hashing (Pagh and Rodler, 2004) with 3 hash functions.

To assign $n$ items into $b$ bins using Cuckoo hashing, first choose random functions $h_1, h_2, h_3 : \{0, 1\}^* \rightarrow [b]$ and initialize empty bins $B[1, \ldots, b]$.

To hash an item $x$, first check to see whether any of the bins $B[h_1(x)], B[h_2(x)], B[h_3(x)]$ are empty.

If so, then place $x$ in one of the empty bins and terminate.
Otherwise, choose a random \( i \in \{1, 2, 3\} \), evict the item currently in \( B[h_i(x)] \) and replace it with \( x \), and then recursively try to insert the evicted item.

If this process does not terminate after a certain number of iterations, then the final evicted element is placed in a special bin called the stash.
The PSSZ Protocol

- First, the parties choose 3 random hash functions $h_1, h_2, h_3$ suitable for 3-way Cuckoo hashing

- Suppose $P_1$ has input set $X$ and $P_2$ has input set $Y$, where $|X| = |Y| = n$

- $P_2$ maps its items into $1.2n$ bins using Cuckoo hashing and a stash of size $s$
  - At this point, $P_2$ has at most one item per bin and at most $s$ items in its stash

- $P_2$ pads its input with dummy items so that each bin contains exactly one item and the stash contains exactly $s$ items
The parties then run 1.2n + s instances of OPRF where P_2 plays the receiver and uses each of its 1.2n + s items as input to OPRF.

Let \( F(k_i, \cdot) \) denote the PRF evaluated in the \( i \)-th OPRF instance, and each \( k_i \in_R \{0, 1\}^\kappa \) only known to \( P_1 \).

Then \( P_2 \) learns the following:

- \( F(k_i, y) \): if \( P_2 \) mapped item \( y \) to bin \( i \) via Cuckoo hashing.
- \( F(k_{1.2n+j}, y) \): if \( P_2 \) mapped \( y \) to position \( j \) in the stash.
On the other hand, $P_1$ can compute $F(k_i, \cdot)$ for any value

$P_1$ computes sets of candidate PRF outputs:

$$H = \{ F(k_{hi}(x), x) | x \in X \text{ and } i \in \{1, 2, 3\} \}$$

$$S = \{ F(k_{1.2n+j}, x) | x \in X \text{ and } j \in \{1, \ldots, s\} \}$$

$P_1$ randomly permutes elements of $H$ and $S$ and sends them to $P_2$ who can identify the intersection of $X$ and $Y$ as follows.
The PSSZ Protocol

- If $P_2$ has an item $y$ mapped to a hashing bin, it checks whether its associated OPRF output is in $H$.
- If $P_2$ has an item $y$ mapped to the stash, it checks whether the associated OPRF output is present in $S$. 
The PSSZ Protocol - Security

- Intuitively, the protocol is secure against a semi-honest $P_2$ by the PRF property.
- For an item $x \in X - Y$, the corresponding PRF outputs $F(k_i, x)$ are pseudorandom.
- Similarly, if the PRF outputs are pseudorandom even under related keys, then it is safe for the OPRF protocol to instantiate the PRF instances with related keys.
The protocol is correct as long as the PRF does not introduce any further collisions, i.e.,

\[ F(k, x) = F(k', x'), \text{ for } x \neq x' \]

We must carefully set the parameters required for the PRF to prevent such collisions.
Acknowledgment

- Chapter 3: Fundamental MPC Protocols, A Pragmatic Introduction to Secure Multi-Party Computation
  https://securecomputation.org/