CSCI 1515: Applied Cryptography

P. Miao

Spring 2023

These are lecture notes for CSCI 1515: Applied Cryptography taught at Brown University by Peihan Miao in the Spring of 2023.

These notes are taken by Jiahua Chen with gracious help and input from classmates and fellow TAs. Please direct any mistakes/errata to me via email, post a thread on Ed, or feel free to pull request or submit an issue to the notes repository.

Notes last updated March 23, 2023.

Contents

1 March 21, 2023 **2**

1.1 Secure Multi-Party Computation, *continued* 2

1.1.1 Feasibility Results 3

1.2 Oblivious Transfer 3

1.3 Yao's Garbled Circuit 3

1.3.1 Optimizations 5

2 March 23, 2023 **6**

2.1 Oblivious Transfer 6

2.1.1 OT Extension 7

2.2 Putting it Together: Semi-Honest 2PC 7

2.3 GMW 7

2.3.1 AND Gates 8

2.3.2 Complexities 8

2.3.3 Entire Protocol 9

§1 March 21, 2023

Survey results: generally that course is well paced, some mentioned too slow or too fast. Seems to be a healthy in-between.

We'll also be having a guest speaker! They are from Google and have implemented MPC in real life.

Last time, we mentioned Bilinear pairings. It was left unresolved whether the target group can be the same group as the domain groups. We should have them be different. Specifically, this allows us to do a *single* multiplication in the exponent. If they are all the same group, we can do arbitrary polynomials in the exponent, which is not desired.

To do an arbitrary n number of equations is called a multilinear map. There are no known secure constructions of which.

§1.1 Secure Multi-Party Computation, *continued*

To quickly recap, a two-party computation is a computation where two parties want to jointly compute a function $f(x, y)$ on their private inputs x, y—but they do not reveal to each what their inputs are.

In the multi-party case, there will be n parties $P_1, P_2, \dots, P_n$ with inputs $x_1, x_2, \dots, x_n$ wishing to jointly compute $f(x_1, x_2, \dots, x_n)$. We generally assume there are secure point-to-point channels, but some models assume broadcast channels. A single adversary can "corrupt" a subset of the parties, say t.

Here are properties we wish to attain in our protocol:

Correctness. The function is computed correctly.

Privacy. Only the output is revealed.

Independence of Inputs. Parties cannot choose their inputs depending on others' inputs.

Also with security guarantees:

Security with Abort. The adversary may "abort" the protocol. This prevents honest parties from receiving the output. This is the weakest model.

Fairness. If one party receives the output, then all parties will receive the output.

Guaranteed Output Delivery (GOD): Honest parties *always* receive output. Even if adversarial parties leave, the honest parties will simply continue the protocol.

§1.1.1 Feasibility Results

In the computational security setting, if we have a fundamental building block, a semi-honest oblivious transfer (OT), we can get semi-honest MPC for any function $t < n$. At a high level, using zero-knowledge proofs to enforce correctness of the protocol, we can convert any semi-honest MPC into a malicious MPC.

In terms of information-theoretic (IT) security. We can also get semi-honest and malicious MPC for any function with $t < \frac{n}{2}$. We call this an honest majority. This is a necessary bound, we cannot do any better than this.

§1.2 Oblivious Transfer

Definition 1.1 (Oblivious Transfer)

An oblivious transfer is a protocol in which a sender, with messages $m_0, m_1 \in \{0,1\}^l$ gives a choice to the receiver to receive either m_0, m_1.

Given a choice bit from the receiver $b \in \{0,1\}$, the receiver gets m_b and the sender also gets no information about the messeage transferred.

We'll learn about constructions of OT later, but we black-box its implementation until later.

Using a semi-honest OT, we can use Yao's Garbled Circuit to construct semi-honest 2PC for any function. We can also use the GMW compiler to compile this into a semi-honest MPC for any function. We'll focus on the first approach in this lecture, but we'll learn GMW in the following lectures.

§1.3 Yao's Garbled Circuit

Example 1.2 (Private Dating/AND Gate)

Alice and Bob want to figure out whether they want to go on a second date. Alice has single bit $x \in \{0,1\}$, and Bob also has single bit $y \in \{0,1\}$.

They want to compute a single AND gate.

Alice will *garble* circuit wires by generating some random l_0, l_1 for each wire corresponding to each bit possibility. We call these *labels*.

For each AND gate[1], she'll generate 4 ciphertexts,

$$\begin{gathered}\mathsf{Enc}_{\alpha_0}(\mathsf{Enc}_{\beta_0}(0))\\ \mathsf{Enc}_{\alpha_0}(\mathsf{Enc}_{\beta_1}(0))\\ \mathsf{Enc}_{\alpha_1}(\mathsf{Enc}_{\beta_0}(0))\\ \mathsf{Enc}_{\alpha_1}(\mathsf{Enc}_{\beta_1}(1))\end{gathered}$$

If we have some α_a, β_b, then we can decrypt $\mathsf{Enc}_{\alpha_a}(\mathsf{Enc}_{\beta_b}(\cdots))$ and all other ciphertexts will look like garbage (we gain no information). This is to say, we can only decrypt the ciphertext of the keys we know. The overarching idea is that we'll only know the right labels for our inputs.

Alice will send the circuit (the 4 encryptions) as well as the input label for x, α_a. Bob now needs to get the label correponding to his input wire, β_0, β_1. We can perform an oblivious transfer!

Bob has a choice bit and gets one of β_0, β_1 without Alice knowing his choice bit. Having attained β_b, Bob will try the encryption on all 4 ciphertexts with α_a, β_b, and sees which output is valid and returns that.

For Alice to learn this output, Bob will send the output back to Alice. In the semi-honest setting, Bob will honestly send the result back to Alice. In the malicious case, we might require Bob to provide some zero-knowledge proof in the end to prove that their plaintext result came from their circuit.

We'll generatlize this single-gate computation for arbitrary functions. We'll represent any arbitrary function as a boolean circuit consisting of only AND and XOR gates[2].

Every wire gets two labels, corresponding to a 0 bit or 1 bit. Each label is $\xleftarrow{\$} \{0,1\}^\lambda$. For each gate, we construct a 'mini' garbled table, where the encrypted message is the output 0 or 1 labels[3,4]. We vary the encryptions based on the gate we're trying to implement.

Using the garbled circuit, we can construct an arbitrary 2PC. We call the party who generates the circuit the 'garbler', and the other party the 'evaluator'.

Alice garbles the circuit, and sends it to Bob. Alice can easily send her own labels. For the labels corresponding to Bob's input, we run oblivious transfer for each input wire to get Bob's input bits without Alice knowing.

In the final output, we can encrypt plaintext $0, 1$. The other way is for Alice to send the final

[1] We can change the values depending on the different logic gates.

[2] Recall that any boolean circuit can be represented using only AND and XOR gates.

[3] *How will we know which is garbage?* Naïvely, we could just try every label. However, this is an exponential blowup for every gate we run the labels through. The solution is to attach a bitstring 'tag' (could just be a string of 0s) that indicates whether a decryption is indeed a label.

[4] One more subtle thing we should take care of! We show our ciphertexts in order of $00, 01, 10, 11$. This reveals information! We should take care to shuffle the ciphertexts everywhere.

random labels to Bob along with their corresponding bits.

§1.3.1 Optimizations

There are some optimizations we can make:

Point-and-Permute. For each wire, we'll randomly sample signal bits $\sigma_\alpha, \sigma_\beta$, and flip it for the other input. (Note that this doesn't reveal anything about α, β). In the circuit, we can indicate using the signal bit which ciphertext to decrypt.

We reduce Bob's computation complexity by at least a constant of 4, and saves communication complexity by half (we don't need to expand our garbled circuit size anymore).

Row Reduction. In this construction, there are 4 ciphertexts per gate. We can just hash the labels and XOR with the corresponding output label (this is not CPA-secure, but that is fine). From the 4 ciphertexts, we can set γ_0 to exactly the hash $H(\alpha_0 || \beta_0)$[5]. This is compatible with point-and-permute. We hide every row, which is fine. This gives us a $\frac{3}{4}$ space decrease.

Free XOR. Sample a global $\Delta \xleftarrow{\$} \{0,1\}^\lambda$. Every pair of labels differ by Δ. That is,

$$\begin{aligned} \alpha_1 &:= \alpha_0 \oplus \Delta \\ \beta_1 &:= \beta_0 \oplus \Delta \\ \gamma_1 &:= \gamma_0 \oplus \Delta \end{aligned}$$

and $\gamma_0 = \alpha_0 \oplus \beta_0$. To compute the output label, you just perform the XOR plainly.

This is to say, XOR is free. We don't need to send labels and Bob doesn't need to encrypt/decrypt.

We can also use *half-gates* which give us 2λ bits per AND gate + free XOR. A recent development, *slicing-and-dicing*, gives us around $\sim 1.5\lambda$ bits per AND gate + free XOR.

[5] Not really the $0, 0$ labels, but they can correspond to the signal bits.

§2 March 23, 2023

Some quick notes: if you submit homework after your alloted late allotment, we will still grade it, but it will not count toward your grade.

Additionally, we're seeing some responses that look like they were from chatbots like ChatGPT. They *will not* produce the correct responses and are definitely a violation of academic code. This has been placed in the syllabus.

§2.1 Oblivious Transfer

We saw last time how to construct Yao's garbled circuit using oblivious transfer, but we black boxed the implementation of OT.

We'll go over the implementation of semi-honest OT here. It will follow similarly to the Diffie-Hellman key exchange.

The sender will send $A = g^a$. The receiver will mask A^c with $c \in \{0, 1\}$ and $b \xleftarrow{\$} \mathbb{Z}_q$. a, b here are like Diffie-Hellman privates.

Then, the sender will compute $k_0 := H(B^a), k_1 := H\left(\left(\frac{B}{A}\right)^a\right)$. This means that k_c will be exactly $g^{ab} = A^b$ (whether $c = 0$ or $c = 1$). Then, k_0 and k_1 will be used to encrypt m_0, m_1 respectively.

Since only one will be the shared Diffie-Hellman key (and the other will require knowledge of a), the receiver will only be able to reveal one such message.

Doing out the algebra, we can conclude that the receiver can access the key.

Is this secure against a semi-honest receiver? If the key is $c = 0$, then the other key will be g^{ab-a^2}. g^{a^2} is difficult to compute, since the receiver only has $A = g^a$ and will need secret a to compute g^{a^2}.[6] If $c = 1$, then the other key will be g^{a^2+ab} which is hard again. So, for the receiver, it is computationally secure.

Is this secure against a semi-honest sender? g^b is a random mask on A^c, so the sender will not be able to distinguish between this.

[6] Formally, this security is guaranteed by the CDH assumption, that if we have g^α, g^β, it's computationally hard to determine $g^{\alpha\beta}$. If an adversary can derive g^{α^2} from g^α, they can also derive $g^{\alpha\beta}$. We can get $g^{\alpha^2}, g^{\beta^2}$, then we can get $g^{(\alpha+\beta)^2} = g^{\alpha^2+2\alpha\beta+\beta^2}$ and taking inverses we can peel off the α^2, β^2 exponents to get $g^{\alpha\beta}$.

§2.1.1 OT Extension

We used public-key operations to achieve our OT. Is it possible to construct OT only using symmetric-key primitives? Unlikely...

There are impossibility results that show that if we assume $\mathsf{P} \neq \mathsf{NP}$, it's not possible to construct an OT using symmetric-key primitives.

This makes OTs very difficult—since it takes an entire protocol (including expensive exponentiations) to transfer one bit. There has been current research in *extending* OT so we can use more bits.

An OT extension can extend $O(\lambda)$ OTs (with $O(\lambda)$ public-key operations) into a $\text{poly}(\lambda)$ bit OTs.

§2.2 Putting it Together: Semi-Honest 2PC

We can now construct our 2PC protocol. Alice, the garbler, will create the circuit with garbled inputs and wires (shuffling order of ciphertexts). Alice sends this circuit to Bob, and Bob will use OTs with his input bits to get the wire labels that he should use. Then, Bob runs these labels on the garbled circuit.

In the semi-honest case, Alice will generate this circuit correctly and Bob will follow the protocol correctly. *What could go wrong against malicious adversaries?*

- Alice could garble an incorrect gate, or give an entirely incorrect circuit.
- Alice could refuse to send the result (translate output label to bits) back to Bob, or send an incorrect result to Bob. If the outputs are not garbled, then Bob could similarly refuse to send this back to Alice.
- Alice and Bob could both cheat about their inputs.

§2.3 GMW

We can convert this into a MPC for any function with $t \leq n - 1$ (corrupted parties up to all but one).

Throughout the protocol, we keep the invariant that for each wire w, if the value of the wire is $v^w \in \{0, 1\}$, then the parties hold an additive secret share of v^w. Each party P_i holds a random share $v_i^w \in \{0, 1\}$ such that

$$\bigoplus_{i=1}^{n} v_i^w = v^w$$

and we keep this invariant throughout the entire circuit.

We need to be able to preserve this invariant throughout AND and XOR gates. The XOR case is easy, since XOR is completely commutative and associative, so each party can locally XOR their shares $c_i := a_i \oplus b_i$ for $c := a \oplus b$.

We'll wave our hands over the AND case, but we can do this. We'll proceede gate-by-gate for everyone to compute the result. Each party will publish their local shares, and everyone will XOR the result together to get the final result.

§2.3.1 AND Gates

We now finish addressing the AND gates. We have $\bigoplus_{i=1}^n a_i = a$ and $\bigoplus_{i=1}^n b_i = b$.

We want a set of $\{c_i\}$ s.t. $\bigoplus_{i=1}^n c_i = c = a \cdot b$ (multiplication of bits is AND). But

$$\begin{aligned} a \cdot b &= \left(\sum_{i=1}^n a_i\right) \cdot \left(\sum_{i=1}^n b_i\right) \pmod 2 \\ &= \left(\sum_{i=1}^n a_i \cdot b_i\right) \cdot \left(\sum_{i \neq j} a_i b_i\right) \pmod 2 \end{aligned}$$

The first sum is easy and computed locally, but the second sum requires parties to communicate. We do something called <u>resharing</u>.

Between P_i, P_j, we want random $r_i, r_j \in \{0,1\}$ such that $r_i + r_j = a_i \cdot b_j \pmod 2$. P_i will randomly sample $r_i \xleftarrow{\$} \{0,1\}$. We can use OT (!) to allow P_j to learn r_j such that $r_i + r_j = a_i \cdot b_j \pmod 2$ without revealing a_i or r_i.

P_i will be the sender, P_j is the receiver. P_j's choice bit is b_j. Then the messages will be

$$\begin{aligned} m_0 &= (a_i \cdot 0) - r_i \\ m_1 &= (a_i \cdot 1) - r_i \end{aligned}$$

such that r_i, r_j are two shares of $a_i \cdot b_j$.

§2.3.2 Complexities

What is the computational complexity for each party? Computational complexity is $O(\#\mathsf{AND} \cdot n)$ for each party. And for communication, every pair of parties needs to communicate for every AND gate, so $O(n^2 \cdot \#\mathsf{AND})$.

The round complexity is the depth of the circuit, *only counting AND gates* (we can ignore XOR gates).

§2.3.3 Entire Protocol

Here's our entire protocol:

We now have a secure multi-party computation scheme. How might we compare them?

- Yao's Garbled Circuit
  - Malicious security lower overhead
- Goldreich-Micali-Wigderson (GMW)
  - The number of OTs is $\#\mathsf{AND} \cdot n^2$.