Cryptography and Security Protocols **IST-UL, 2024/2025**

Secure computation

Lecturer: João Ribeiro

Introduction

In these notes we will study the cryptographic task of *secure (multiparty) computation*. The setting of secure computation is the following: n parties $P_1, \ldots, P_n$ hold private inputs $x_1, \ldots, x_n$ and each party P_i wishes to compute a joint function $f_i(x_1, \ldots, x_n)$ of all inputs. For simplicity, assume that every pair of parties is connected by private authenticated channels.[1] If we enforce no requirements beyond correctness, then there is a simple solution – each party P_i sends their input x_i to all other parties, and then each party locally computes the desired joint function.

However, we care about settings where parties' inputs may be sensitive information that they do not want to reveal to other parties (beyond what is inherently revealed by their corresponding outputs). For example, imagine that two billionaires want to find out who is richer, but do not want to reveal anything beyond that (in particular, no billionaires learn the other billionaire's net worth, only whether it is greater than theirs). Or, imagine that two hospitals want to cross their private databases to figure out a list of common patients, without revealing any other information about their databases (beyond the unavoidable fact that patients outside the intersection are not in both databases).

1 Secure 2-party computation

In these notes we will focus on secure *2-party* computation (secure 2PC), where two parties, Alice and Bob, with private inputs x and y, respectively, want to interact in order to compute (possibly randomized) functions $f_A(x, y)$ and $f_B(x, y)$, respectively, without revealing any extra information about the other party's input (we formalize this below). Secure 2PC can be naturally generalized to a setting with $n \geq 2$ parties. You will see something about this in the problem set.

Before we discuss concrete secure computation protocols we must first formalize our design goals. In particular, we must formalize the notion that "each party learns nothing about the other party's input beyond what can be inferred from their own input and output". Also, we need to formalize the *adversarial model*. We will consider the weakest non-trivial form of security, in the sense that we will only require that the informal privacy notion described above holds under the assumption that both parties always follow the protocol, but may record intermediate steps of the protocol to try and extract additional information about the other party's input. We call this these parties

[1] There are many interesting problems related to the "synchrony" of these channels. But that is beyond the scope of our discussion.

honest-but-curious (these parties are also sometimes called *semi-honest* or *passive*). This notion bears some similarity to the notion of honest-verifier zero-knowledge proofs that we saw before. Our formal definition will follow the same principles.

Definition 1 (Secure protocol) *We say that an interactive protocol $\langle A(x), B(y)\rangle$ between two PPT algorithms A and B with inputs $x \in \mathcal{X}$ and $y \in \mathcal{Y}$, respectively,* securely computes the function[2] *$f(x,y) = (f_A(x,y), f_B(x,y))$ if the following properties hold:*

1. ***Correctness:*** *With probability 1, at the end of the interaction A holds $f_A(x,y)$ and B holds $f_B(x,y)$.*
2. ***Security:*** *For every $x \in \mathcal{X}$ and $y \in \mathcal{Y}$ there exist PPT simulators $\mathcal{S}_A$ and $\mathcal{S}_B$ such that*
   (a) *$\mathcal{S}_A(x, f_A(x,y))$ is computationally indistinguishable from $\mathsf{view}_A\langle A(x), B(y)\rangle$;*
   (b) *$\mathcal{S}_B(y, f_B(x,y))$ is computationally indistinguishable from $\mathsf{view}_B\langle A(x), B(y)\rangle$.*

   As when we discussed interactive proofs and zero-knowledge proofs, $\mathsf{view}_A\langle A(x), B(y)\rangle$ (resp. $\mathsf{view}_B\langle A(x), B(y)\rangle$) denotes the full view of party A (resp. B) in the protocol $\langle A(x), B(y)\rangle$.

We emphasize that depending on the function f that we wish to securely compute we may have to allow, say, A to learn some information about B's input y. For example, if $x, y \in \{0,1\}$ and $f_A(x,y) = f_B(x,y) = x + y \pmod 2$, then A can actually recover y from x and the output $f_A(x,y)$, and likewise for B. This means that the trivial protocol where A sends x to B and B sends y to A, and both locally compute $x + y \pmod 2$, securely computes this f.

For other functions there are properties of the other party's input that must remain hidden. For example, consider $x, y \in \{0,1\}$ and $f(x,y) = (\perp, x \cdot y)$ (in other words, A gets no output and B gets the AND of x and y). If B has input $y = 0$ and learns output $f_B(x,y) = 0$, then $(y, f_B(x,y))$ reveals no information about x.

This discussion raises some natural questions:

> *Do we need computational assumptions to securely compute some functions f?*

The answer turns out to be "yes". For example, it is possible to show that there is no "statistically secure" 2-party protocol for securely computing the AND function $f(x,y) = (x \cdot y, x \cdot y)$. More generally, unconditionally secure n-party computation of even simple functions is impossible unless there is an "honest majority" (i.e., the adversary is only allowed to get the view of $t < n/2$ parties in the protocol). So we need computational assumptions for secure 2PC. Interestingly, as you will see in the problem set, general multiparty computation with an "honest majority" can be achieved unconditionally.

> *For which functions do we have secure 2PC protocols? And under which computational hardness assumptions?*

[2] Sometimes we will also call f a "functionality".

Very surprisingly, the answer to this question is "all efficiently computable functions, under mild computational hardness assumptions"! And, in fact, this is true even if we allow a large subset of corrupted parties to deviate arbitrarily from the protocol. This was proved by Goldwasser, Micali, and Wigderson [GMW87].

2 Oblivious transfer

We will begin by studying a very specific secure computation problem called *oblivious transfer*. This corresponds to the following setting: A holds two inputs (x_0, x_1) and B holds a bit $b \in \{0,1\}$. After the interactive protocol, B should hold x_b. More formally, oblivious transfer (OT) is the problem of securely computing $f((x_1, x_2), b) = (\perp, x_b)$. Intuitively, A should not learn anything about b (the input chosen by B), and B should not learn anything about x_{1-b} (the input he did not choose). This problem was introduced by Rabin [Rab81].

2.1 Oblivious transfer from public-key encryption with "oblivious" key-generation

On our way to devising secure protocols for any efficiently computable function, let's design a secure protocol for OT. Our protocol will be based on CPA-secure PKE schemes that admit an additional "oblivious" key-generation algorithm. Intuitively, this algorithm allows someone to generate a "fake" public key that (1) looks like a real public key, and (2) no-one, not even the party that generated the fake public key, can decrypt ciphertexts encrypted using the fake public key.

Informally, armed with this special PKE scheme, we can implement OT as follows:

1. Bob, with input $b \in \{0,1\}$, generates a real public key/secret key pair (pk, sk) using the real key generation algorithm and a fake public key pk^* using the oblivious key generation algorithm. Then, he sets $pk_b = pk$ and $pk_{1-b} = pk^*$ and sends (pk_0, pk_1) to Alice.
2. Alice, with input (x_0, x_1), computes the encryptions $c_i = \mathsf{Enc}(pk_i, x_i)$ and sends (c_0, c_1) to Bob.
3. Bob decrypts c_b using sk, thus obtaining x_b.

Intuitively, this protocol is secure because (1) Alice cannot tell the difference between pk_b (the real public key) and pk_{1-b} (the fake public key), and (2) Bob gets no information about x_{1-b} from c_{1-b}, because it was encrypted using the fake public key.

Let's formalize things.

Definition 2 (PKE with oblivious key generation) *We say that a PKE scheme* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ admits oblivious key generation *if there exists a deterministic algorithm* OGen *that receives* $(1^n, r)$ *as input, with* $r \in \{0,1\}^\ell$ *the randomness used to generate its output, running in time* $\mathrm{poly}(n)$*, satisfying the following properties:*

- *For any PPT adversary $\mathcal{A}$ there exists a negligible function $\varepsilon(n)$ such that for $r \leftarrow \{0,1\}^\ell$ and $pk^* = \mathsf{OGen}(1^n, r)$ and $(pk, sk) \leftarrow \mathsf{Gen}(1^n)$ we have*

$$|\Pr[\mathcal{A}(1^n, pk^*) = 1] - \Pr[\mathcal{A}(1^n, pk) = 1]| \leq \varepsilon(n).$$

- *For any PPT adversary $\mathcal{B}$ there exists a negligible function $\varepsilon(n)$ such that for $r \leftarrow \{0,1\}^\ell$, $pk^* = \mathsf{OGen}(1^n, r)$, any two messages m_0 and m_1, and $b \leftarrow \{0,1\}$ we have*

$$\Pr[\mathcal{D}(1^n, r, pk^*, \mathsf{Enc}(pk^*, m_b)) = b] \leq \frac{1}{2} + \varepsilon(n).$$

  Note that $\mathcal{D}$ receives the randomness r used to generate pk^* as input too.

Using this definition, a more formal description of the protocol described above is as follows:

1. Bob, with input $b \in \{0,1\}$, samples $(pk, sk) \leftarrow \mathsf{Gen}(1^n)$ and also samples $r \leftarrow \{0,1\}^\ell$ and computes $pk^* = \mathsf{OGen}(1^n, r)$. Then, he sets $pk_b = pk$ and $pk_{1-b} = pk^*$ and sends (pk_0, pk_1) to Alice.
2. Alice, with input (x_0, x_1), computes the encryptions $c_i = \mathsf{Enc}(pk_i, x_i)$ and sends (c_0, c_1) to Bob.
3. Bob computes $x_b = \mathsf{Dec}(sk, c_b)$.

Theorem 1 *The protocol described above securely computes OT.*

Proof: To prove this theorem we must design appropriate simulators for the views of both parties. We describe the simulators and sketch arguments that their outputs are computationally indistinguishable from the corresponding views. We leave formalizing these arguments as homework (these require only standard techniques you have seen several times before).

Alice's view in the protocol is $(x_0, x_1, pk_0, pk_1, \perp)$. Consider the PPT simulator $\mathcal{S}_A(x_0, x_1, \perp)$ that works as follows:

1. Generate $(pk_i, sk_i) \leftarrow \mathsf{Gen}(1^n)$ independently for each $i \in \{0,1\}$.
2. Output $(x_0, x_1, pk_0, pk_1, \perp)$.

Fix (x_0, x_1) and b. The only difference between $\mathcal{S}(x_0, x_1, \perp)$ and Alice's view is that in Alice's view pk_b is generated using OGen, while $\mathcal{S}_A$ generates it using Gen. So, intuitively, the claim follows if Alice cannot efficiently distinguish between a real public key and a fake public key. It is not hard to show that a PPT adversary that distinguishes between these two distributions with non-negligible advantage can be used to construct a PPT adversary that breaks the first property of Definition 2.

Bob's view in the protocol is $(b, pk, r, pk^*, c_0, c_1, x_b)$, where $pk^* = \mathsf{OGen}(1^n, r)$ and $c_i = \mathsf{Enc}(pk_i, x_i)$ for $i \in \{0,1\}$ with $pk_b = pk$ and $pk_{1-b} = pk^*$. Consider the PPT simulator $\mathcal{S}_B(b, x_b)$ that works as follows:

1. Generate $(pk, sk) \leftarrow \mathsf{Gen}(1^n)$ and $r \leftarrow \{0,1\}^\ell$ and $pk^* = \mathsf{OGen}(1^n, r)$.
2. Set $c_b = \mathsf{Enc}(pk, x_b)$ and $c_{1-b} = \mathsf{Enc}(pk^*, 0)$.
3. Output $(b, pk, r, pk^*, c_0, c_1, x_b)$.

Note that the only difference between the simulator's output and Bob's view is that in Bob's view c_{1-b} is an encryption of x_{1-b} under pk^*, while in the simulator's output it is an encryption of 0 under pk^*. Intuitively, security holds because, by the second property of Definition 2, an efficient Bob cannot distinguish between encryptions of x_{1-b} and of 0 under a fake public key pk^*, *even if Bob knows the randomness r used to generate pk^**. ■

2.1.1 Some examples of PKE schemes with oblivious key generation

We have seen how to securely compute OT assuming we have access to a PKE scheme with oblivious key generation. We now discuss some examples of PKE schemes with this property.

ElGamal encryption. Recall that the public key in ElGamal encryption is $pk = (G, q, g, h = g^x)$, where $x \leftarrow \mathbb{Z}_q$, and the secret key is (G, q, g, x). We can obliviously sample a fake public key pk^* by sampling h uniformly at random from G, *in a way that does not reveal its discrete logarithm with respect to the generator g*. The procedure for doing this depends on the underlying cyclic group G. When $G = \mathbb{Z}_p^*$, we can do this simply by picking a uniformly random element of $\{0, 1, \ldots, p-1\}$.

In this case, real and fake public keys are identically distributed (and so the first property of Definition 2 is trivially satisfied). This also means that the second property of Definition 2 follows directly from the security of ElGamal PKE under DDH.

Regev encryption. Recall that the public key in Regev encryption is $pk = (A, As + e \pmod q)$, where $A \leftarrow \mathbb{Z}_q^{m \times n}$, $s \leftarrow \mathbb{Z}_q^n$, and $e \leftarrow \chi^{\otimes n}$ for some noise distribution χ. We can obliviously sample a public key by sampling $z \leftarrow \mathbb{Z}_q^m$ and setting $pk^* = (A, z)$. I.e., z is now a uniformly random vector in $\mathbb{Z}_q^m$ independent of A. The fact that no efficient adversary can distinguish between pk and pk^* follows directly from the decision-LWE assumption. The fact that no efficient adversary with knowledge of pk^* follows from part of the hybrid argument we used to show CPA-security of Regev encryption. Namely, Regev encryption using a uniformly random z produces ciphertexts that are statistically close to uniformly distributed (in terms of statistical distance/total variation distance). Note that the randomness used to generate pk^* is embedded in pk^* itself.

3 Some applications of OT

OT is deceptively powerful. We now work through some concrete applications.

3.1 Securely computing ANDs

Let's use OT to securely compute the AND functionality $f(\alpha, \beta) = (\perp, \alpha \cdot \beta)$ for $\alpha, \beta \in \{0, 1\}$. Note that when, say, $\beta = 0$, Bob should not learn anything about Alice's input α. The protocol is as follows:

1. Alice sets $x_0 = 0$ and $x_1 = \alpha$.
2. Bob sets $b = \beta$.
3. Alice and Bob run an OT protocol with inputs (x_0, x_1) and b, respectively.

Correctness is easy to verify. After this protocol, Bob holds $x_b = x_\beta$. If $\beta = 0$, then

$$x_b = x_0 = 0 = \alpha \cdot 0 = \alpha \cdot \beta.$$

If $\beta = 1$, then

$$x_b = x_1 = \alpha = \alpha \cdot 1 = \alpha \cdot \beta.$$

Security is also not hard to establish, assuming we have access to PPT simulators $\mathcal{S}_A^{\mathrm{OT}}$ and $\mathcal{S}_B^{\mathrm{OT}}$ for OT. More precisely, Alice's view is simply her view in the OT protocol with inputs $(x_0, x_1) = (0, \alpha)$ and $b = \beta$. So her view is computationally indistinguishable from $\mathcal{S}_A^{\mathrm{OT}}(0, \alpha)$. Likewise, Bob's view is simply his view in the OT protocol with inputs $(x_0, x_1) = (0, \alpha)$ and $b = \beta$. Therefore, his view is computationally indistinguishable from $\mathcal{S}_B^{\mathrm{OT}}(\beta)$.

3.2 The billionaires' problem

Suppose that two billionaires, Alice and Bob, each hold inputs x and y from the set $\{1, \ldots, M\}$, representing their (private) net worth. They wish to securely compute $f(x, y) = (\mathbf{1}_{\{x \geq y\}}, \mathbf{1}_{\{x \geq y\}})$, where $\mathbf{1}_{\{x \geq y\}} = 1$ if and only if $x \geq y$. In words, they want to securely learn who is richer.

Before we describe a protocol for this problem we need to set up some things. Alice first constructs the vector

$$v_x^A = (0, \ldots, 0, 1, 0, \ldots, 0) \in \{0, 1\}^M,$$

where the 1 occurs in the x-th coordinate. Bob constructs the vector

$$v_y^B = (0, \ldots, 0, 1, 1, \ldots, 1) \in \{0, 1\}^M,$$

where the 1's occur from the y-th coordinate onwards. Note that

$$\mathbf{1}_{\{x \geq y\}} = \langle v_x^A, v_y^B \rangle = \sum_{i=1}^{M} (v_x^A)_i \cdot (v_y^B)_i \pmod 2.$$

A natural first attempt would be to have Alice and Bob securely compute each "AND" $(v_x^A)_i \cdot (v_y^B)_i$, and then have them compute the sum locally. But a moment's though reveals that this does not work, as Bob will learn x this way. So we must proceed differently.

3.2.1 Securely computing secret-shared ANDs

Although the natural first attempt does not work, it sets us off in the right direction. It turns out that one thing that does work is to securely compute *secret-shared ANDs*.

More precisely, we will have Alice and Bob securely compute the functionality $f(\alpha, \beta) = (\gamma, \delta)$, where γ and δ are sampled uniformly at random from $\{0, 1\}$ conditioned on $\gamma + \delta = \alpha \cdot \beta \pmod 2$. In other words, after running the protocol, Alice and Bob hold shares of a 2-out-of-2 additive secret sharing of $\alpha \cdot \beta$.

It turns out that we can securely compute secret-shared ANDs using OT! Consider the following protocol:

1. Alice samples $\gamma \leftarrow \{0, 1\}$ and sets $x_0 = \gamma$ and $x_1 = \gamma + \alpha$.[3]
2. Bob sets $b = \beta$.
3. Alice and Bob run an OT protocol with inputs (x_0, x_1) and b, respectively.
4. Alice outputs γ and Bob outputs $\delta = x_b$.

To see that this protocol is correct, note that the output Bob gets from OT is

$$\delta = x_b = (1 - b)x_0 + bx_1 = x_0 + b(x_1 - x_0) = \gamma + \beta(\gamma + \alpha - \gamma) = \gamma + \alpha \cdot \beta \pmod 2.$$

Therefore, γ and δ are each uniformly distributed over $\{0, 1\}$, and also $\gamma + \delta = \alpha \cdot \beta \pmod 2$, as desired. Security of this protocol follows analogously to the security of the AND protocol above, assuming access to PPT simulators for the underlying OT protocol.

3.2.2 Solving the billionaires' problem through secret-shared ANDs

We now use secret-shared ANDs to solve the billionaires' problem. Consider the following protocol:

1. Alice and Bob construct their respective vectors v_x^A and v_y^B.
2. For each $i \in \{1, \ldots, M\}$, Alice and Bob run the secure protocol for computing the secret-shared AND of $(v_x^A)_i$ and $(v_y^B)_i$. Denote the resulting additive shares by γ_i and δ_i.
3. Alice computes $\gamma = \sum_{i=1}^{M} \gamma_i \pmod 2$ and sends γ to Bob.
4. Bom computes $\delta = \sum_{i=1}^{M} \delta_i \pmod 2$ and sends δ to Alice.
5. Alice and Bob output $\gamma + \delta \pmod 2$.

[3] We may see Alice's inputs to the OT protocol as an additive secret sharing of her input α.

Correctness follows from the linearity of additive secret sharing. Indeed,

$$\gamma + \delta = \sum_{i=1}^{M} \gamma_i + \sum_{i=1}^{M} \delta_i = \sum_{i=1}^{M} (\gamma_i + \delta_i) = \sum_{i=1}^{M} [(v_x^A)_i \cdot (v_y^B)_i] = \mathbf{1}_{\{x \geq y\}} \pmod 2.$$

Establishing security of this protocol is left as homework.

4 The GMW protocol – General secure 2-party computation from oblivious transfer and secret sharing

We have seen how OT allows us to securely compute some interesting functionalities. We will now see that, in fact, OT allows us to securely compute *any efficiently computable function*!

For simplicity, we will focus on the setting where Alice and Bob hold inputs $x \in \{0,1\}^n$ and $y \in \{0,1\}^n$, respectively, and wish to compute $f(x,y) = (g(x,y), g(x,y))$ for some arbitrary function $g : \{0,1\}^\ell \times \{0,1\}^\ell \to \{0,1\}$. Every such function g can be represented as an *arithmetic circuit*. This is a directed acyclic graph G where vertices represent gates and edges represent wires. There are input wires, intermediate wires, and an output wire. Each of the input wires carries an input bit. The output wire carries the final output $g(x,y)$. The intermediate wires carry outputs of gates to be input into the following gates. There are two types of gates that can be used. The MULT gate receives two inputs $\alpha, \beta \in \{0,1\}$ and outputs $\alpha \cdot \beta$. The ADD gate receives two inputs $\alpha, \beta \in \{0,1\}$ and outputs $\alpha + \beta \pmod 2$.

Writing g as an arithmetic circuit suggests a natural approach: securely computing $g(x,y)$ "gate-by-gate", just like we would process the circuit computation. However, it is not immediate how to do this appropriately, because in this process Alice and Bob (1) should not learn the output of any intermediate gate in the circuit by themselves, and (2) should still have enough information at the end to learn the final output of the circuit together. This suggests the use of secret sharing!

Concretely, Alice and Bob set things up as follows:

1. Alice computes additive secret sharings (x_i^A, x_i^B) of each input x_i and sends $x_1^B, \ldots, x_\ell^B$ to Bob.
2. Bob computes additive secret sharings (y_i^A, y_i^B) of each input y_i and sends $y_1^A, \ldots, y_\ell^A$ to Alice.

After this step, Alice and Bob know an additive share of each input to the circuit. We now securely process the circuit gate-by-gate as follows. Suppose that Alice and Bob respectively hold additive shares (α^A, α^B) and (β^A, β^B) of inputs α and β to some gate $h : \{0,1\}^2 \to \{0,1\}$ in the circuit. Then, they follow a secure protocol to obtain an additive secret sharing (γ, δ) of the gate output $h(\alpha, \beta)$. More precisely, after this protocol Alice holds γ, Bob holds δ, and γ and δ are uniformly random over $\{0,1\}$ conditioned on $\gamma + \delta = h(\alpha, \beta) \pmod 2$.

Because of this invariant, in the end Alice and Bob hold additive secret shares of the output $g(x,y)$. Each of them sends their share to the other party, and they reconstruct the output.

Given the above, we must design secure protocols for each type of gate in the circuit.

4.1 Securely computing addition and multiplication gates

ADD gates. Securely computing the ADD gate in a secret-shared fashion is easy. Suppose that Alice holds additive shares α^A and β^A while Bob holds additive shares α^B and β^B, such that $\alpha^A + \alpha^B = \alpha \pmod 2$ and $\beta^A + \beta^B = \beta \pmod 2$. Recall that their goal is to end up with additive shares γ and δ, respectively, such that $\gamma + \delta = \alpha + \beta \pmod 2$.

This can be accomplished without any interaction between Alice and Bob, exploiting the linearity of additive secret sharing. More precisely, Alice computes $\gamma = \alpha^A + \beta^A \pmod 2$ and Bob computes $\delta = \alpha^B + \beta^B \pmod 2$. Note that each of γ and δ is uniformly distributed over $\{0, 1\}$, and that

$$\gamma + \delta = (\alpha^A + \beta^A) + (\alpha^B + \beta^B) = (\alpha^A + \alpha^B) + (\beta^A + \beta^B) = \alpha + \beta \pmod 2.$$

MULT gates. Securely computing the MULT gate in a secret-shared fashion is not as easy. It is reminiscent of the secret-shared AND functionality we showed how to compute securely based on OT.

Suppose that Alice holds additive shares α^A and β^A while Bob holds additive shares α^B and β^B, such that $\alpha^A + \alpha^B = \alpha \pmod 2$ and $\beta^A + \beta^B = \beta \pmod 2$. Recall that their goal is to end up with additive shares γ and δ, respectively, such that $\gamma + \delta = \alpha \cdot \beta \pmod 2$.

Let's build some intuition before attempting to design a protocol. Note that

$$\alpha \cdot \beta = (\alpha^A + \alpha^B) \cdot (\beta^A + \beta^B) = \alpha^A \cdot \beta^A + \alpha^A \cdot \beta^B + \alpha^B \cdot \beta^A + \alpha^B \cdot \beta^B \pmod 2.$$

Let's look at each of the terms in the rightmost sum. Alice can compute $\alpha^A \cdot \beta^A$ on her own. Likewise, Bob can compute $\alpha^B \cdot \beta^B$ on his own. The term $\alpha^A \cdot \beta^B$ cannot be computed locally, but it is an AND of Alice's input α^A and Bob's input β^B. So we can run a secret-shared AND protocol with inputs α^A and β^B so that Alice and Bob get additive shares γ^{AB} and δ^{AB}, respectively, such that $\gamma^{AB} + \delta^{AB} = \alpha^A \cdot \beta^B \pmod 2$. Analogously, they can run a secret-shared AND protocol with inputs β^A and α^B, respectively, so that Alice and Bob get additive shares γ^{BA} and δ^{BA}, respectively, such that $\gamma^{BA} + \delta^{BA} = \alpha^B \cdot \beta^A \pmod 2$.

Finally, Alice sets her additive share to be

$$\gamma = \alpha^A \cdot \beta^A + \gamma^{AB} + \gamma^{BA} \pmod 2.$$

Analogously, Bob sets his additive share to be

$$\delta = \alpha^B \cdot \beta^B + \delta^{AB} + \delta^{BA} \pmod 2.$$

Note that each of γ and δ is uniformly distributed over $\{0, 1\}$, because, say, γ^{AB} (resp. δ^{AB}) is uniformly random over $\{0, 1\}$ and independent of the other terms in the sum. Correctness is also easy to verify.

We now discuss the security of this protocol. Let's focus on Alice's view – the argument for Bob's view is analogous. Alice gets as input (α^A, β^A) and receives γ as output. This γ is uniformly random over $\{0, 1\}$ from Alice's perspective, but is also correlated with Bob's output δ so that $\gamma + \delta = \alpha \cdot \beta \pmod 2$. We will try to be quite explicit about things:

- Alice's view in the protocol with input (α^A, β^A) is

$$(\alpha^A, \beta^A, \gamma^{AB}, \gamma^{BA}, V_A^{\mathrm{OT}}((\gamma^{AB}, \gamma^{AB} + \alpha^A), \beta^B), V_A^{\mathrm{OT}}((\gamma^{BA}, \gamma^{BA} + \beta^A), \alpha^B), \gamma),$$

  where all sums are mod 2, $V_A^{\mathrm{OT}}((x_0, x_1), b)$ denotes Alice's view in the OT protocol with Alice's inputs (x_0, x_1) and Bob's input b, γ^{AB} and γ^{BA} are independent and uniformly random over $\{0, 1\}$, and $\gamma = \alpha^A \cdot \beta^A + \gamma^{AB} + \gamma^{BA} \pmod 2$.

- The simulator $\mathcal{S}(\alpha^A, \beta^A, \gamma)$ must simulate this view when the output is γ. Let $\mathcal{S}_A^{\mathrm{OT}}(x_0, x_1)$ be the simulator of Alice's view in the OT protocol with Alice's input (x_0, x_1). Then the simulator works as follows:

  1. Sample γ^{AB} and γ^{BA} uniformly at random from $\{0, 1\}$ conditioned on $\gamma^{AB} + \gamma^{BA} = \gamma + \alpha^A \cdot \beta^A \pmod 2$.
  2. Generate simulated views $V_{AB} \leftarrow \mathcal{S}_A^{\mathrm{OT}}(\gamma^{AB}, \gamma^{AB} + \alpha^A)$ and $V_{BA} \leftarrow \mathcal{S}_A^{\mathrm{OT}}(\gamma^{BA}, \gamma^{BA} + \beta^A)$.
  3. Output $(\alpha^A, \beta^A, \gamma^{AB}, \gamma^{BA}, V_{AB}, V_{BA}, \gamma)$.

Establishing computational indistinguishability between the simulator's output and Alice's view follows by a hybrid argument where, say,

- The first hybrid is the simulator's output.
- In the second hybrid we replace V_1 in the simulator's output and $V_A^{\mathrm{OT}}((\gamma^{AB}, \gamma^{AB} + \alpha^A)$. The two hybrids are computationally indistinguishable by the security of the underlying OT protocol (homework: formalize this).
- In the third hybrid we replace V_2 by $V_A^{\mathrm{OT}}((\gamma^{BA}, \gamma^{BA} + \beta^A)$. This hybrid is distributed exactly like Alice's true view. It is computationally indistinguishable from the second hybrid again by the security of the OT protocol.

4.2 Security of the GMW protocol

We sketch the full simulator for the GMW protocol, assuming access to PPT simulators for ADD and MULT gates.

The simulator $\mathcal{S}_A(x, g(x, y))$ works as follows:

1. For each $i \in \{1, \ldots, \ell\}$, sample $x_i^A \leftarrow \{0, 1\}$ and set $x_i^B = x_i + x_i^A \pmod 2$.
2. For each $i \in \{1, \ldots, \ell\}$, sample $y_i^A \leftarrow \{0, 1\}$. These are Alice's additive secret shares of Bob's input y.
3. Do the following for each gate in the circuit (appropriately sorted). Suppose that Alice holds inputs α^A and β^A for that gate.
   (a) If the gate is an ADD gate, simply set Alice's output to $\alpha^A + \beta^A \pmod 2$.

(b) If the gate is a MULT gate, sample Alice's supposed output $\gamma \leftarrow \{0,1\}$ and run the simulation strategy outlined in the previous section.

4. At the end Alice holds z^A, which is an additive share of the circuit output. Compute $z^B = z^A + g(x,y) \pmod 2$ (the simulator knows how to compute this because he knows the final output $g(x,y)$) and give z^B to Alice.

The simulator for Bob is analogous.

One can prove that the output of the full simulator is computationally indistinguishable from the corresponding party's view in the protocol via a standard hybrid argument, where we replace each gate of the circuit by the corresponding simulator for that gate one by one.

4.3 Security against "active" adversaries?

The GMW protocol securely computes an efficiently computable function against honest-but-curious parties ("efficiently computable" in the sense of being represented by an arithmetic circuit with a polynomial number of gates in the security parameter), using OT and secret sharing. It is an easy exercise to show that the protocol is easily broken if we allow one of the parties to be dishonest and deviate arbitrarily from the protocol. We call such adversaries *active*.

Goldreich, Micali, and Wigderson [GMW87] also showed that it is possible to "compile" any protocol for computing a given functionality secure against honest-but-curious parties into a protocol for the same functionality secure against active adversaries.

5 A note on composability

In these notes we discussed secure protocols for computing some simple functionalities. In the real world, we usually want to compose several of these protocols together (sequentially and in parallel) to arrive at a secure protocol for a more complex task. Ensuring that protocols "compose nicely", in the sense that the relevant security properties are preserved by composition is far from trivial.

Protocol composability, broadly speaking, is an active area of research. There are various general frameworks for designing protocols that remain secure when composed. A notable example is Canetti's universal composability framework [Can01].

6 Further reading

If you want to learn more about secure multiparty computation, this book by Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen, volume 2 of Oded Goldreich's book on the foundations of cryptography, Daniel Escudero's survey on secret-sharing-based MPC are good starting points, and these lecture notes on cryptographic computing by Claudio Orlandi. This short and sweet paper by Maurer also provides great insight.

References

[Can01] Ran Canetti. Universally composable security: a new paradigm for cryptographic protocols. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, pages 136–145, 2001.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play ANY mental game. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, page 218–229, New York, NY, USA, 1987. Association for Computing Machinery.

[Rab81] Michael O. Rabin. How to exchange secrets with oblivious transfer, 1981. https://eprint.iacr.org/2005/187.