University of Illinois, Urbana Champaign
CS 598DK Special Topics in Cryptography

Instructor: Dakshita Khurana
Scribe: Sanjeev Reddy, Tom Yurek
Date: September 18, 2019

LECTURE
7

Oblivious Transfer and Secure Computation

In this lecture, we continue to explore oblivious transfer and simulation-based approaches to constructing secure computation schemes. We also consider examples in which we construct two-party secure communication for circuits containing AND and XOR gates (gates that can be used to model any logical circuit when used in combination with NOT).

7.1 Recap: Oblivious transfer

An oblivious transfer communication is a protocol in which neither the sender nor receiver learn more about the other's input than is allowed. In particular, we study a 1-2 OT scheme in which the sender transmits 2 messages x_0, x_1, while the receiver submits a selection bit $b \in \{0,1\}$. The receiver should only learn x_b of the sender's input while learning nothing about x_{1-b}, while the sender should not learn anything about the receiver's selection (i.e. which of the 2 messages the receiver selected). Figure 7.1 illustrates a 1-2 oblivious transfer between sender Alice and receiver Bob.

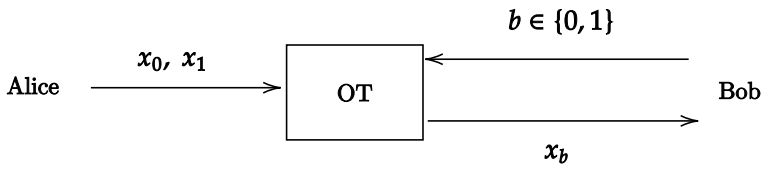


FIGURE 7.1: A 1-2 oblivious transfer scheme.

The security of an OT scheme can be defined via a **game-based** or **simulation-based** model. We will discuss the simulation-based security model in section 7.4. The game-based security definition is centered around honest-but-curious PPT participants Alice and Bob each playing one of two games, where each variation of their game requires them to submit a different input to the OT oracle. Broadly, the game-based security model is satisfied

if neither Alice nor Bob, given the protocol output, can guess which of the two games the other participant was playing with probability $P > \frac{1}{2} + negl(k)$. More specifically, the game-based notion of security is defined via two sub-definitions: **receiver security** and **sender security**.

DEFINITION 7.1. *Receiver security*: Receiver Bob plays one of two games. In Game 1, Bob sets his selection bit $b = 0$. In Game 2, Bob sets his selection bit $b = 1$. The OT scheme is **receiver secure** if sender Alice is not able to guess which of the two games Bob was playing with probability $P > \frac{1}{2} + negl(k)$. Note that this is equivalent to stating that Alice should not be able to guess Bob's b with probability $P > \frac{1}{2} + negl(k)$.

DEFINITION 7.2. *Sender security*: Sender Alice plays one of two games. As a precondition, receiver Bob keeps his selection bit b constant across both games. Alice keeps her input x_b constant across both games, but varies input x_{1-b} between Game 1 and Game 2. The OT scheme is **sender secure** if Bob is not able to guess which of the two games Alice was playing with probability $P > \frac{1}{2} + negl(k)$.

7.2 Ideal vs. real world

A sensitive multi-party computation is a protocol that operates on secrets from multiple sources and outputs the results of the computation to required participants. Naturally, each participant should not learn the other participants' secrets during the computation, and should only learn the output given *their* secret input. One way in which we can model this requirement is the **ideal-world** model.

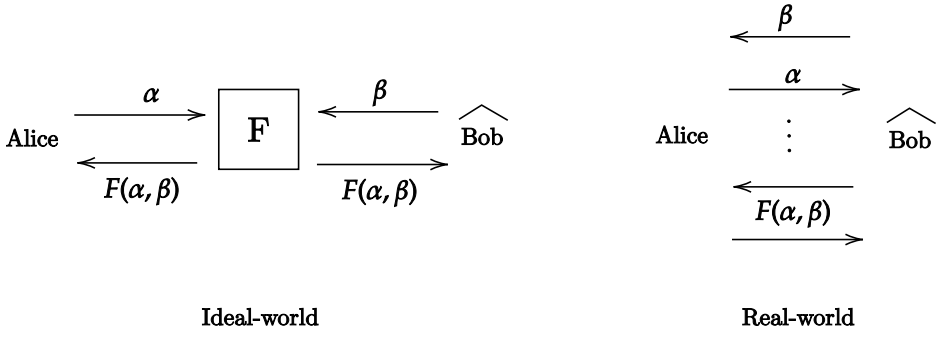


FIGURE 7.2: A comparison between ideal and real-world models.

In this setting, participants Alice and Bob use a trusted third-party oracle (known as an ideal functionality) $F(\alpha, \beta)$ to handle their secret inputs α and β respectively. F performs the protocol operations on Alice and Bob's inputs, and returns *only* the output of the operations to the participants. In this ideal setting, Alice and Bob learn the desired result of the multi-party computation, but learn nothing about the other's secret input, since the ideal functionality behaved as a third-party black box that handled all secrets independently

from the participants. An example of the ideal-world model can be seen in Figure 7.2.

We now consider an alternate (and more representative) model: the **real-world** model for multi-party computation. In this model, Alice and Bob interface directly with each other, as they would in a regular communication. There is no trusted third-party involved, and the participants themselves must perform a two-party protocol that computes $F(\alpha, \beta)$. Assuming Alice and Bob both follow the two-party protocol as they are supposed to, they will learn nothing except for their own secret and the result $F(\alpha, \beta)$. However, if one of them is adversarial—either honest-but-curious or outright malicious—that participant will learn more over the course of the communication than they are supposed to in the ideal-world setting. In other words, their view of the real-world will differ from their view of the ideal-world.

DEFINITION 7.3. *View of participant P*: The **view** of participant P in a communication protocol is the tuple (T, x, R), where T represents a transcript of the communication, x represents P's input to the communication, and R represents the randomness used by P [**?**].

Given this definition of a view and our understanding of the real and ideal-world settings, we can provide a view-based definition of security for a multi-party communication system.

DEFINITION 7.4. A system is secure if a participant $\widehat{P}$'s view in the real-world run of the system is computationally indistinguishable from $\widehat{P}$'s view in the ideal-world run of the system.

$$View_{\widehat{P}_{real}} \approx View_{\widehat{P}_{ideal}}$$

7.3 Simulators

As we observed earlier, an adversarial participant in a two-party protocol might obtain a different view in the real-world setting than they do in the ideal-world setting. Their transcript T of the communication will differ from the ideal-world transcript. This violates the security definition provided in 7.4. In order to remedy this issue, we introduce the concept of **simulators** to our ideal-world model.

A simulator is a PPT machine M that serves as an intermediary between participant $\widehat{P}_1$ and the ideal functionality. Importantly, M *simulates* participant P_2's behavior from the real-world transcript by running the protocol functionality usually run by P_2 in the two-party communication. M is a publicly available algorithm run by $\widehat{P}_1$, and has the ability to "rewind," or repeat parts of the protocol. Assuming $\widehat{P}_1$ is honest-but-curious, they should always respond honestly and correctly to any of the inputs from the simulator.

DEFINITION 7.5. *Simulator*: Given a participant P who runs protocol F with input x in a real-world two-party communication, PPT machine M **simulates** P in the ideal-world if:

$$Output_{P_{real}}(F(x)) \approx Output_M(F(x))$$

Figure 7.3 outlines an ideal-world model with a simulator. The simulator *extracts* β from Bob's transcript and forwards it to the ideal functionality. Similarly, it retrieves $F(\alpha, \beta)$ from the ideal functionality and forwards it to Bob over the course of their communication. The simulator does *not* learn any of Alice's secrets, as those are handled by the ideal functionality. Instead, it simulates a real-world conversation with Alice with bogus intermediary messages to satisfy Bob's real-world view of the protocol. Note that from Bob's perspective, the

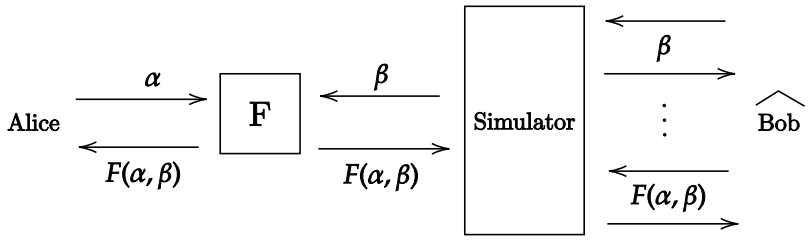


FIGURE 7.3: The ideal-world model with an "Alice" simulator. Although it is not pictured, note that Alice would also run a "Bob" simulator that emulates Bob's real-world transcript.

conversation he is conducting with the simulator is indistinguishable from a conversation he would have with Alice in the real-world. However, the ideal functionality prevents his exposure to any of Alice's secrets, thus satisfying the security definition 7.4 even in the case that Bob is an honest-but-curious adversary. This means that definition 7.4 can be revised to state that a system is secure if:

$$\exists \text{ simulator } S \text{ s.t. } View_{\widehat{P}_{real}} \approx View_{\widehat{P}_{ideal}}$$

7.4 Simulator model for oblivious transfer

We will now define a simulator model for oblivious transfer schemes, and use this model to define a **simulation-based** security model for OT. We will focus on a construction that places a "Bob" simulator (hereafter denoted as S) between Alice and the ideal OT functionality (denoted F_{OT}), although one could easily extend the model to provide Bob with an "Alice" simulator (i.e. a simulator that extracts Bob's selection bit b from his real-world transcript and outputs x_b).

S makes use of public key cryptography in conjunction with F_{OT} to ensure that Bob will only receive x_b. Given S has randomness r and runs a PKE scheme PKE, we define its behavior as follows:

1. Extract Alice's input messages (x_0, x_1) from her real-world transcript
2. Run $KeyGen_{\text{PKE}}$ twice to generate two public/private key pairs:
$$KeyGen_{\text{PKE}}(1^k, r) \rightarrow (pk_0, sk_0)$$
$$KeyGen_{\text{PKE}}(1^k, r) \rightarrow (pk_1, sk_1)$$
3. Encrypt x_0 and x_1 using pk_0 and pk_1 respectively:
$$Enc_{\text{PKE}}(pk_0, x_0, r) \rightarrow Enc(x_0)$$
$$Enc_{\text{PKE}}(pk_1, x_1, r) \rightarrow Enc(x_1)$$
4. Send the encrypted messages and private keys to F_{OT}
$$Enc(x_0), Enc(x_1), sk_0, sk_1 \rightarrow F_{OT}$$

Once F_{OT} is in possession of Alice's encrypted inputs and the corresponding private keys, it simply needs to wait to receive Bob's selection bit b. Given b, F_{OT} can select sk_b and decrypt the corresponding encryption:

$$Dec_{\text{PKE}}(sk_b, Enc(x_b)) \rightarrow x_b$$

Now, F_{OT} can forward x_b to Bob (or the simulator Bob is running). Figure 7.4 provides a visualization of this simulator model for oblivious transfer.

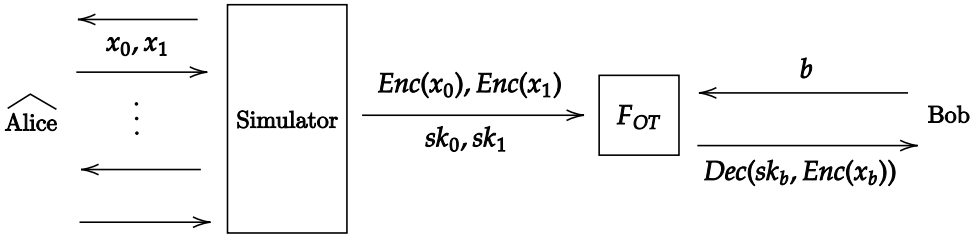


FIGURE 7.4: Simulator model for an OT scheme.

We are not done yet, though—what if the simulator itself behaves improperly? In order to strengthen the definition of the simulator, we assume that a corrupt Alice can learn something about Bob's selection—perhaps partial contents of the message he received—after the protocol has been completed (the means by which this is done are beyond the scope of this lecture). If Alice were to gain access to such information, she might be able to cross-check it against the inputs she presented in her view, and learn that the simulator corrupted or falsified her messages x_0 and x_1. Therefore, in order to ensure that the simulator does not misbehave (e.g. delivering random x_0 or x_1 values to F_{OT}), we extend the simulation-based security definition for OT to require indistinguishability of views as well as receiver selections between the real and ideal-world.

DEFINITION 7.6. *Simulation-based security model for OT*: An oblivious transfer scheme involving participants $\widehat{P_1}$ and P_2 is secure if:

$$\exists \text{ simulator } S \text{ s.t. } (View_{\widehat{P_1}_{real}}, selection_{P_{2_{real}}}) \approx (View_{\widehat{P_1}_{ideal}}, selection_{P_{2_{ideal}}})$$

7.5 Secure Computation of AND and XOR

In the previous lecture, we discussed the functionality $\mathcal{F}_{\text{XOR}}$ (shown in Figure 7.5) and how to realize it. The key insight is that for a two party XOR computation between Alice and Bob, both parties can learn the other's bit simply by looking at the output of the computation. Therefore, a secure protocol Π_{XOR} (Figure 7.6) can be achieved simply by having Alice and Bob exchange their bits.

$\mathcal{F}_{\text{XOR}}$

Inputs:
$\mathcal{F}_{\text{XOR}}$ receives a bit a from Alice and a bit b from Bob

Outputs:
$\mathcal{F}_{\text{XOR}}$ sends $a \oplus b$ to both Alice and Bob

FIGURE 7.5: $\mathcal{F}_{\text{XOR}}$, a functionality for computing the XOR of two bits

Π_{XOR}

Inputs:
Alice and Bob hold bits a and b respectively

Protocol:

- Alice sends a to Bob
- Bob sends b to Alice
- Alice and Bob both output $a \oplus b$

FIGURE 7.6: Π_{XOR}, a protocol for computing the XOR of two bits

THEOREM 7.7. *Π_{XOR} is indistinguishable from $\mathcal{F}_{XOR}$ from the perspective of an honest but curious adversary. Specifically,*

$$\exists \text{ simulator } S_1 \text{ s.t. } (View_{\widehat{P_1}_{real}}, outputP_{2real}) \approx (View_{\widehat{P_1}_{ideal}}, outputP_{2ideal}) \wedge$$
$$\exists \text{ simulator } S_2 \text{ s.t. } (View_{\widehat{P_2}_{real}}, outputP_{1real}) \approx (View_{\widehat{P_2}_{ideal}}, outputP_{1ideal})$$

Proof. Consider a PPT simulator S_A that is positioned between Alice and $\mathcal{F}_{\text{XOR}}$. Per the protocol, Alice sends a to S_A, which then forwards a to $\mathcal{F}_{\text{XOR}}$. S_A receives $a \oplus b$, computes $b = a \oplus b \oplus a$, and sends b to Alice. Alice will always send a and receive b in return, regardless of whether she is interacting with Bob or S_A. Thus the views are indistinguishable. Since Alice and Bob have identical roles in this protocol, S_B does the same thing as S_A and Bob's views are indistinguishable for the same reason. □

Of course, in practice, it is desirable to realize other boolean functions besides just XOR. For example, if we can also create a protocol that is indistinguishable from $\mathcal{F}_{\text{AND}}$, then we can securely compute any boolean circuit. However, accomplishing this is less straightforward, as viewing an output $a \wedge b$ while knowing a does not necessarily reveal b, and vice versa.

We have, however, recently introduced a tool that can help us securely realize $\mathcal{F}_{\text{AND}}$, Oblivious Transfer. The idea is that Alice can offer Bob one of two choices: $a \wedge 0$ or $a \wedge 1$. If Bob's bit $b = 0$ he should choose the former, and if $b = 1$ he should choose the later. After this, Bob should reveal the output to Alice, completing the protocol.

When it comes time to specify and prove the security of our protocol Π_{AND}, it would be cumbersome if we also needed to include protocol level details of our Oblivious Transfer

$\mathcal{F}_{\text{AND}}$

Inputs:
$\mathcal{F}_{\text{AND}}$ receives a bit a from Alice and a bit b from Bob

Outputs:
$\mathcal{F}_{\text{AND}}$ sends $a \wedge b$ to both Alice and Bob

FIGURE 7.7: $\mathcal{F}_{\text{AND}}$, a functionality for computing the AND of two bits

Π_{AND}

Inputs:
Alice and Bob hold bits a and b respectively

Protocol:

- Alice calculates $a \wedge 0$ and $a \wedge 1$ and gives these to as $\mathcal{F}_{\text{1-2 OT}}$ choices 0 and 1 respectively
- Bob sends b to $\mathcal{F}_{\text{1-2 OT}}$ and receives $a \wedge b$
- Bob sends $a \wedge b$ to Alice
- Alice and Bob both output $a \wedge b$

FIGURE 7.8: Π_{AND}, a protocol for computing the AND of two bits

subprotocol. Luckily, we don't need to. Earlier, we showed that we can securely realize the Oblivious Transfer functionality with an existing protocol. Since the protocol perfectly emulates the functionality, they are effectively interchangeable. So we instead specify our protocol Π_{AND} in Figure 7.8 by assuming it has access to an ideal functionality that realizes Oblivious Transfer. This idea is called *composability*.

DEFINITION 7.8. *C*omposability (informal): A protocol can be specified such that it relies on subprotocols that behave in a particular way. Instead of specifying the real world subprotocol, an ideal world functionality can be substituted in instead. The full protocol is secure in the real world if (in addition to the outer protocol being secure) there exists a real world protocol that emulates the functionality in a way that is indistinguishable from the perspective of the defined adversary.

THEOREM 7.9. *Π_{AND} is indistinguishable from $\mathcal{F}_{AND}$ from the perspective of an honest but curious adversary. Specifically,*

$$\exists \text{ simulator } S_1 \text{ s.t. } (View_{\widehat{P_1}_{real}}, outputP_{2real}) \approx (View_{\widehat{P_1}_{ideal}}, outputP_{2ideal}) \wedge$$
$$\exists \text{ simulator } S_2 \text{ s.t. } (View_{\widehat{P_2}_{real}}, outputP_{1real}) \approx (View_{\widehat{P_2}_{ideal}}, outputP_{1ideal})$$

Proof. First let us consider Alice's view of the protocol. Let S_A be a probabilistic polynomial time simulator for Alice. The protocol begins with Alice sending $a \wedge 0$ and $a \hat{} 1$ to $\mathcal{F}_{\text{OT}}$. Because we already know there exists a protocol Π_{OT} that emulates that functionality, we also know that there is a PPT extractor that can determine Alice's inputs to the functionality. So S_A can then extract $a \wedge 0$ and $a \hat{} 1$ from Alice. Because $a \hat{} 1 = a$, S_A determines a and sends it to $\mathcal{F}_{\text{AND}}$. S_A receives $a \hat{} b$ from $\mathcal{F}_{\text{AND}}$ and sends it to Alice.

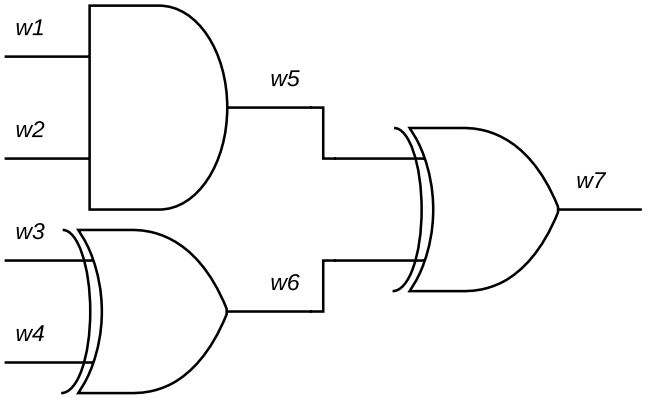
Bob's simulator S_B acts similarly. S_B is able to extract Bob's decision bit b from the Oblivious Transfer. S_B then simply sends b to $\mathcal{F}_{\text{AND}}$, receives $a \hat{} b$, and sends it to Bob.

Both of these simulators inherit the same properties of the underlying Oblivious Transfer protocol and are therefore Alice and Bob are similarly unable to distinguish between ideal world and real world executions. □

7.6 Achieving Arbitrary Computation

As mentioned earlier, XOR gates, AND gates and NOT gates are all that is needed to make any other logic gate and therefore perform any computation that is expressible as a *boolean circuit*.

Definition 7.10. Boolean Circuit (informal): a collection of logical gates that are connected by wires, where each wire represents a particular boolean value. Inputs to the circuit are placed onto input wires, these wires are fed through a series of logic gates, and eventually one or more outputs are produced.



Example 7.11.

In the above boolean circuit, w_1, w_2, w_3, and w_4 are input wires and w_7 is the output wire.

Example 7.11 demonstrates an example boolean circuit. Say that Alice's inputs are w_1 and w_3, while Bob's are w_2 and w_4. We could conceivably evaluate this circuit gate by gate to reveal the output, but that would reveal the intermediate values w_5 and w_6, which reveals some information about the inputs. What if we want to compute this circuit and reveal to Alice and Bob no more information than is implied by knowing the output w_7? We'll need to introduce a new technique due to GMW [**?**] which uses secret-shared bits.

Definition 7.12. t-of-N Secret Sharing: A scheme where a secret s is divided into N total shares such that possession of any t shares of s reveals s completely, but an adversary who only possesses $t-1$ or fewer shares learns no additional information about s.

In this scenario, we are concerned with 2-of-2 Secret bit Sharing, i.e. Alice and Bob both have a share of a given bit such that if the shares are combined, the bit is revealed. We realize such a scheme through the use of the XOR function. Specifically, given a bit b, we may create two shares b_0 and b_1 such that $b_0 \oplus b_1 = b$. Note that knowing either b_0 or b_1 tells you nothing about b if you do not know the other share.

In fact, this scheme has a very useful property: shares are *XOR-homomorphic*. In the above example circuit, say Alice performs 2-of-2 Secret Sharing on w_3 and Bob does the same for w_4 (and let's for example call the shares of w_3 by $[w_3]_a$ and $[w_3]_b$ for Alice and Bob's shares respectively). Then Alice and Bob can both locally calculate shares of w_6 by XOR'ing their shares of w_3 and w_4 together. This maintains the invariant that Alice and Bob's shares of a given wire should recreate that wire's value when XOR'ed together:

$$\begin{aligned}[w_6]_a \oplus [w_6]_b &= ([w_3]_a \oplus [w_4]_a) \oplus ([w_3]_b \oplus [w_4]_b) \\ &= ([w_3]_a \oplus [w_3]_b) \oplus ([w_4]_a \oplus [w_4]_b) = w_3 \oplus w_4 = w_6\end{aligned}$$

By using secret sharing, we can now evaluate a circuit consisting of many XOR gates without necessarily revealing all of the inputs. Of course, only a small fraction of computations are expressible using only XOR gates. However, calculating an AND gate is not quite as easy. In the example circuit, $w_5 = w_1 \wedge w_2$. If we expand this out, we get

$$\begin{aligned}w_5 &= ([w_1]_a \oplus [w_1]_b) \wedge ([w_2]_a \oplus [w_2]_b) \\ &= [w_1]_a \wedge [w_2]_a \oplus [w_1]_a \wedge [w_2]_b \oplus [w_1]_b \wedge [w_2]_a \oplus [w_1]_b \wedge [w_2]_b\end{aligned}$$

Clearly, $[w_1]_a \wedge [w_2]_a$ and $[w_1]_b \wedge [w_2]_b$ can be calculated locally by Alice and Bob respectively, but $[w_1]_a \wedge [w_2]_b$ and $[w_1]_b \wedge [w_2]_a$ present a challenge. We can't reveal these products either, as that would leak information about the inputs. Instead we have Alice sample a random bit r_1, calculate $r_1 \oplus [w_1]_a$ and invoke $\mathcal{F}_{\text{AND}}$ to send Bob $r_1 \oplus [w_1]_a \wedge [w_2]_b$ which does not reveal $[w_1]_a \wedge [w_2]_b$ to Bob. Bob likewise computes a random bit r_2 and invokes the functionality to give Alice $r_2 \oplus [w_1]_b \wedge [w_2]_a$. Then Alice and Bob can calculate their shares of w_5 as follows:

$$\begin{aligned}[w_5]_a &= ([w_1]_a \wedge [w_2]_a) \oplus (r_2 \oplus [w_1]_b \wedge [w_2]_a) \oplus r_1 \\ [w_5]_b &= ([w_1]_b \wedge [w_2]_b) \oplus (r_1 \oplus [w_1]_a \wedge [w_2]_b) \oplus r_2\end{aligned}$$

Then, when Alice and Bob's shares are XOR'ed together, the random bits cancel out and we're left with

$$[w_5]_a \oplus [w_5]_b = [w_1]_a \wedge [w_2]_a \oplus [w_1]_a \wedge [w_2]_b \oplus [w_1]_b \wedge [w_2]_a \oplus [w_1]_b \wedge [w_2]_b = w_5$$

Finally, we observe that the NOT gate simply flips its input, and this is trivially realizable over secret shared inputs. Since we now have the tools to evaluate both XORs and ANDs of secret shared bits without revealing them, we can use this to evaluate any computation expressible as a boolean circuit.

Acknowledgement

These scribe notes were prepared by editing a light modification of the template designed by Alexander Sherstov.