

CS 858: Computation on Encrypted Data

Lecturer: Mohammad Hajiabadi

Contents

1 Preliminaries	3
2 Multiplication Gates	7
3 Security of Shamir's Secret Sharing	8
4 Secret Sharing on Monotone Functions	8
5 Semi-Honest Security for MPC	10
6 Recap	11
7 Semi-Honest Security	13
8 To think about for next lecture	16
9 Oblivious Transfer	16
9.1 Secure AND from OT	17
9.2 (1, 4)-OT	17
9.3 General Secure Computation (GMW)	18
9.3.1 AND Gates	19
9.3.2 XOR Gates	20
9.3.3 NOT Gates	21
10 Equivalence of Semi-Honest Security Definitions	22
11 Building Secure OT from Secure AND	23
12 Additively Homomorphic Encryption	24
12.1 Building OT from additively homomorphic encryption	26
13 OT from the Decisional Diffie-Hellman assumption	27
13.1 1-bit A-HE from DDH	27
13.2 Simpler way to make OT from DDH	28
14 CPA secure PKE from 2 round OT	29
15 Security of OT against outsiders	30
16 Question for later	31

17 Public-Key Operations vs Symmetric-Key Operations	32
17.1 Motivating Example	32
18 OT Extensions	32
18.1 A Simplified Warmup	33
18.2 History of OT Extensions	34
18.3 High-Level Overview	34
18.4 Phase 1: OT Correlations	34
18.5 Phase 2: OT Extensions from OT Correlations	36
18.6 Purpose of the Random Oracle in OT Correlations	36
18.7 Adaptive OT Extensions	37
19 Private Set Intersection (PSI)	38
19.1 Building OT from PSI	40
19.2 Designing PSI	40
19.2.1 Building PSI from Asymmetric AND	40
19.2.2 Designing PSI (First Attempt)	40
19.2.3 Designing PSI (Fixed)	41
20 Oblivious Pseudorandom Functions (OPRFs)	41
20.1 PSI from OPRFs	42
20.2 Building OPRFs from DH	42
21 2-round MPC using Garbling Circuit	43
21.1 Garbled Table for AND and XOR gate	43
21.2 Garbled Table for Large Circuit	45
21.3 Garbled Circuit Definition	45
21.4 Building a Garbled Circuit without Cryptography	46
21.5 Information Theoretic Garbled Circuits	47
21.6 Garbling a Family of Functions	48
21.7 Garbled Circuits for Complicated Functions	49
21.8 Randomized Encodings	51
21.9 Secure MPC with Constant Round Complexity	54
21.10 Secure Computation Conclusion	56
22 Zero Knowledge Proofs	58
22.1 Zero-Knowledge for NP-Complete Problems	62

1 Preliminaries

We use λ for denoting the security parameter.

We start by revisiting the notions of computational/statistical indistinguishability, central notions in cryptography used as metrics of closeness between distributions.

Deterministic/Randomized (Probabilistic) Algorithms. A deterministic algorithm is one that maps any input deterministically to a fixed output. One may imagine a deterministic algorithm A as a function f with a corresponding input-output space. A probabilistic (randomized) algorithm is one that, in addition to its deterministic input, it also takes in random coins (bits), and outputs a value that depends on both its input and randomness. We use $A(x; r)$ to denote the output of a randomized algorithm A on a deterministic input x and randomness r . When omitting r , $A(x)$ denotes the random variable/distribution obtained by choosing r uniformly at random, and outputting $A(x; r)$. The acronym PPT stands for probabilistic polynomial time.

Definition 1 (Distinguishing Advantage). For two distributions DD and DD' and a (possibly randomized) algorithm/adversary $\mathcal{A}$ that always outputs 0/1, we define the advantage of $\mathcal{A}$ in distinguishing DD from DD' as

$$\text{Dist}_{\mathcal{A}}(DD, DD') := \left| \Pr_{x \leftarrow DD}[\mathcal{A}(x) = 1] - \Pr_{x \leftarrow DD'}[\mathcal{A}(x) = 1] \right|.$$

The above definition captures the difference between DD and DD' in $\mathcal{A}$'s perspective. That is, how similar the behavior of $\mathcal{A}$ is with respect to the two distributions.

It is easy to see that if DD and DD' are identically distributed, then the above difference is always zero, for any (possibly randomized, and non-efficient) adversary $\mathcal{A}$.

Definition 2 (Statistical/Computational Indistinguishability). Let $DD := \{DD_{\lambda}\}_{\lambda \in \mathbb{N}}$ and $DD' := \{DD'_{\lambda}\}_{\lambda \in \mathbb{N}}$ be two ensembles of distributions, indexed by a security parameter λ . (For example, think of DD_{λ} as the output of a pseudorandom number generator on a security parameter 1^{λ} and a uniformly random input.) We say DD and DD' are statistically (resp., computationally) indistinguishable/close if for any (resp., for any PPT) adversary $\mathcal{A}$, $\text{Dist}_{\mathcal{A}}(DD_{\lambda}, DD'_{\lambda})$ is a negligible function of λ . We use $\cong^c$ for computational indistinguishability and $\cong^s$ for statistical indistinguishability.

Another View of Statistical Closeness. We now give another characterization of statistical closeness, which does not involve adversaries. Roughly speaking, two distributions are statistically close if any event holds with almost the same probability under a random sample of the first and second distribution.

Definition 3 (Statistical Distance). Let DD and DD' be two distributions over a universe S . The statistical distance between DD and DD' is defined as

$$\Delta(DD, DD') := \frac{1}{2} \sum_{a \in S} |\Pr[DD = a] - \Pr[DD' = a]|,$$

where $DD = a$ denotes the value assigned to a under the probability mass function (PMF) of DD .

We now prove some basic facts about statistical distance, and relate it to the other characterizations. In particular, we show that

Lemma 1.1. *For any distributions DD and DD' , the following conditions hold.*

1. *If DD and DD' are identically distributed (i.e., having the same PMF), then $\Delta(DD, DD') = 0$. If DD and DD' are disjoint distributions (i.e., for no a , both $DD(a) > 0$ and $DD'(a) > 0$), $\Delta(DD, DD') = 1$.*
2. $0 \leq \Delta(DD, DD') \leq 1$.
3. *For any set S ,*

$$\left| \Pr_{x \leftarrow DD}[x \in S] - \Pr_{x \leftarrow DD'}[x \in S] \right| \leq \Delta(DD, DD').$$

Moreover, for some set S_g

$$\left| \Pr_{x \leftarrow DD}[x \in S_g] - \Pr_{x \leftarrow DD'}[x \in S_g] \right| = \Delta(DD, DD').$$

4. **(Relation to distinguishing advantages)** *Assuming $\delta := \Delta(DD, DD')$, for any (randomized) $\mathcal{A}$, $\text{Dist}_{\mathcal{A}}(DD, DD') \leq \delta$. Moreover, the maximum value δ is achieved by some (not necessarily efficiently computable) adversary $\mathcal{A}$.*

Proof. Part (1) is trivial.

Part 2: $\Delta(DD, DD') \geq 0$ because we are adding non-negative integers. Moreover, since probability values are non-negative,

$$\frac{1}{2} \sum_{a \in S} |\Pr[DD = a] - \Pr[DD' = a]| \leq \frac{1}{2} \sum_{a \in S} (\Pr[DD = a] + \Pr[DD' = a]) = 1.$$

Part 3: Fix distributions DD and DD' . Define

$$U[S] := \left| \Pr_{x \leftarrow DD}[x \in S] - \Pr_{x \leftarrow DD'}[x \in S] \right|.$$

Suppose without loss of generality that $\Pr_{x \leftarrow DD}[x \in S] \geq \Pr_{x \leftarrow DD'}[x \in S]$, and so $U[S] = \Pr_{x \leftarrow DD}[x \in S] - \Pr_{x \leftarrow DD'}[x \in S]$.

We define a set S_g such that $U[S] \leq U[S_g]$ and that $U[S_g] = \Delta(DD, DD')$. This will complete the proof.

Define $S_g = \{a \mid DD(a) > DD'(a)\}$. First, we claim $U[S] \leq U[S_g]$. This is because

- If there exists $a \in S \setminus S_g$ removing a from S will increase the value of $U[S]$.
- If there exists $a \in S_g \setminus S$, adding a to S will increase the value of $U[S]$.

We now show $U[S_g] = \Delta(DD, DD')$. Let S_d be the dual of S_g , namely, $S_d = \{a \mid DD(a) \leq DD'(a)\}$. By how S_g and S_d are defined, we may remove the absolute values to get

$$\delta = \Delta(DD, DD') = \underbrace{\frac{1}{2} \sum_{a \in S_g} (\Pr[DD = a] - \Pr[DD' = a])}_{f_0} + \underbrace{\frac{1}{2} \sum_{a \in S_d} (\Pr[DD' = a] - \Pr[DD = a])}_{f_1}.$$

Note that $f_0 = U[S_g] = \Delta(DD, DD')$. Also, $f_0 = f_1$ because

$$f_0 - f_1 = \sum_{a \in S_g} (\Pr[DD = a] - \Pr[DD' = a]) - \sum_{a \in S_d} (\Pr[DD' = a] - \Pr[DD = a]) = 1 - 1 = 0.$$

Thus, $U[S_g] = \Delta(DD, DD')$.

Part 4: We first show that for any (randomized) $\mathcal{A}$, $\text{Dist}_{\mathcal{A}}(DD, DD') \leq \delta$. Fix (a possibly randomized) adversary $\mathcal{A}$. Define $S = \{x \mid \Pr[\mathcal{A}(x) = 1] > 0\}$. We have $\Pr_{x \leftarrow DD}[\mathcal{A}(x) = 1] = \Pr_{x \leftarrow DD}[x \in S]$. The same statement holds for DD' . Thus,

$$\text{Dist}_{\mathcal{A}}(DD, DD') = \left| \Pr_{x \leftarrow DD}[x \in S] - \Pr_{x \leftarrow DD'}[x \in S] \right| \leq \Delta(DD, DD'),$$

where the last inequality follows by Part 3.

For the proof of the other statement, let S_g be as in Part 3, and define $\mathcal{A}$ as follows: $\mathcal{A}(x)$ outputs 1 iff $x \in S_g$. By Part 3, $\text{Dist}_{\mathcal{A}}(DD, DD') = \delta$. □

We now use the notion of computational indistinguishability to define a number of cryptographic primitives.

Definition 4 (Pseudorandom Generators). A function $G: \{0, 1\}^\lambda \rightarrow \{0, 1\}^{\lambda+\ell}$ for $\ell > 0$ is a pseudorandom generator if its output on a uniformly random input is computationally random (aka, pseudorandom). Formally, $\{G(DU_\lambda)\} \cong^c \{G(DU_{\lambda+\ell})\}$, where DU_i denotes the uniform distribution over i -bit strings.

Note that the above two distributions, $\{G(DU_\lambda)\}, \{G(DU_{\lambda+\ell})\}$, are not statistically close. Indeed, the statistical distance between the two distributions is at least one half (why?), and hence by Part 4 of Lemma 1.1, for some adversary $\mathcal{A}$, $\text{Dist}_{\mathcal{A}}(DD, DD') \geq 1/2$. Thus, pseudorandom generators are objects that produce outputs which are not random in a statistical or information-theoretic sense, yet no polynomial time adversary can tell them apart from random.

Definition 5 (Public-Key Encryption). A public-key encryption scheme for plaintext bits is given by (G, E, D) , where the key generation algorithm is a PPT algorithm that outputs a pair of a public key and a secret key $(pk, sk) \leftarrow G(1^\lambda)$, the encryption algorithm takes in a public key a plaintext bit $b \in \{0, 1\}$ and randomness r and outputs $c := E(pk, b; r)$, and the decryption algorithm D takes in a secret key sk and a ciphertext c and outputs a bit b . We require for all b, r and (pk, sk) and c generated as above, $D(sk, E(pk, b; r)) = b$. The definition of semantic/ chosen-plaintext attack (CPA) stipulates $(pk, E(pk, 0)) \cong^c (pk, E(pk, 1))$, where $(pk, *) \leftarrow G(1^\lambda)$.

Notice that the statistical distance between $(pk, E(pk, 0))$ and $(pk, E(pk, 1))$ is one (the distributions are disjoint), and hence they can be distinguished with advantage 1 by non-efficient adversaries. Thus, they are computationally close, but statistically far.

Definition 6 (Private-Key Encryption). A private-key encryption scheme (G, E, D) is defined similarly to a PKE except that $pk = sk$. For security, we present the following (weak) definition.¹ For any $t \in \text{poly}(\lambda)$ and for any two distributions DD and DD' over t -tuple messages,

$$(m_1, \dots, m_t, m'_1, \dots, m'_t, E(k, m_1), \dots, E(k, m_t)) \cong^c (m_1, \dots, m_t, m'_1, \dots, m'_t, E(k, m'_1), \dots, E(k, m'_t)),$$

where $k \leftarrow G(1^\lambda)$, $(m_1, \dots, m_t) \leftarrow DD$ and $(m'_1, \dots, m'_t) \leftarrow DD'$.

Note that there exists an inefficient adversary $\mathcal{A}$, defined as follows, that achieves a large advantage in distinguishing the two distributions of Definition 6. Given an input

$$(m_1, \dots, m_t, m'_1, \dots, m'_t, c_1, \dots, c_t),$$

where all of m_i and m'_i are uniformly random, if there exists a key k such that $D(k, c_i) = m_i$ for all $i \in [t]$, then output 1; else, output 0. Given a sample from the left distribution, $\mathcal{A}$ always output 1 (because such a key always exists), while given a sample from the right distribution, $\mathcal{A}$ outputs 1 with probability at most $\frac{2^{|k|}}{2^t}$ (why?). Thus, if t is sufficiently larger than $|k|$, the statistical distance between the two distributions becomes almost 1 (i.e., $1 - \text{negl}(\lambda)$).

¹The standard CPA security definition is defined as a game and gives the adversary an adaptive power.

2 Multiplication Gates

Recall that in the last lecture, we discussed computation of secret sharing on addition and multiplication gates, where each party i has a share $\text{sh}_{j,i}$ of the secret s_j .

We want to maintain secret sharing on the gates. In addition it was simple, we noted that the sum of the shares was the share of the sum. However, for multiplication, it is not as simple.

Suppose we have two $(t+1, n)$ -shares $p(i)$ and $q(i)$ for polynomials p, q that have degree t , where $p(0) = a, q(0) = b$. If we try to multiply the polynomials, we end up with a $(2t+1, n)$ -share of ab (for a polynomial pq of degree $2t$).

We will use interaction to make a $(2t+1, n)$ -share of $c = ab$ to a $(t+1, n)$ -share of c .

Since we assumed $2t+1 \leq n$, we can just assume that we have a polynomial h that represents a (n, n) -share of c , and party i knows $\beta_i = h(i)$, and we want to construct $(t+1, n)$ -shares of c .

Claim 2.1. There exists fixed $r_1, r_2, \dots, r_n \in \mathbb{F}$ known to all parties such that

$$c = r_1\beta_1 + r_2\beta_2 + \dots + r_n\beta_n.$$

If we show this claim, then we are done since . . .

Proof. Suppose $c_0 = c$ and let $h(x) = c_0 + c_1x + c_2x^2 + \dots + c_{n-1}x^{n-1}$. Then we have

$$\begin{pmatrix} 1^0 & 1^1 & \dots & 1^{n-1} \\ 2^0 & 2^1 & \dots & 2^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ n^0 & n^1 & \dots & n^{n-1} \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{n-1} \end{pmatrix} = \begin{pmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_n \end{pmatrix}$$

The leftmost matrix is called *Vandermonde Matrix* (denoted V_n), it is known to be invertible, and so we can have a unique solution for $c_0, c_1, \dots, c_{n-1}$:

$$\begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{n-1} \end{pmatrix} = V_n^{-1} \begin{pmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_n \end{pmatrix}.$$

We can write $c = c_0 = r_1\beta_1 + r_2\beta_2 + \dots + r_n\beta_n$ where r_i is the sum of 1st row of V_n^{-1} . □

So, let's recap: (look at the slides ...)

1. Party i has $p(i)$ and $q(i)$ for some degree- t

So, as long as $2t+1 \leq n$, $(t+1, n)$ -shares will be perfectly secure. We assumed this, and now we're going to prove the security of Shamir's secret sharing.

3 Security of Shamir's Secret Sharing

So that's the setup:

1. An Adversary $\mathcal{A}$ gives two secrets s_0 and s_1 and $(t - 1)$ -sized T where T is a sequence of distinct indices between 1 and n .
2. We sample a bit $b \leftarrow \{0, 1\}$ uniformly at random, and share s_b ,
3. The adversary gets the shares of s_b at the indices in T , and outputs a guess b' .
4. The adversary wins if $b' = b$.

Let's call the output of this experiment $\text{Shamir}_{t,n}^{\mathcal{A}}$, and denote the success of $\mathcal{A}$ as $\text{Shamir}_{t,n}^{\mathcal{A}} = 1$ (and 0 otherwise).

The scheme is perfectly secure if that adversary wins with probability at most $1/2$.

Theorem 3.1. *Shamir's secret sharing is perfectly secure. That is,*

$$\Pr[\text{Shamir}_{t,n}^{\mathcal{A}} = 1] \leq \frac{1}{2}.$$

over all adversaries $\mathcal{A}$.

This theorem follows from this lemma:

Lemma 3.2. *Let $T = (i_1, i_2, \dots, i_{t-1})$ and $O = (o_1, o_2, \dots, o_{t-1})$, where $1 \leq i_1 < i_2 < \dots < i_{t-1} \leq n$. Then the set*

$$P_s = \{p : p \text{ is a degree-}t \text{ polynomial and } p(i_j) = o_j \text{ for all } j\}$$

Then, $|P_s| = |\mathbb{F}|$, and for all $x \in \mathbb{F}$, there exists $p \in P_s$ such that $p(0) = x$.

Proof. Left as an exercise for you. □

Note that this lemma shows that a distribution over polynomials $p \in P_s$ is uniform, the probability that $p(0) = x$ is $1/|\mathbb{F}|$, and that's why the theorem holds.

4 Secret Sharing on Monotone Functions

Suppose we have n parties. Any string $x \in \{0, 1\}^n$ denotes a subset of parties. In other words, we have a bijection $\mathcal{M} : \{0, 1\}^n \rightarrow \mathcal{P}(\{p_1, p_2, \dots, p_n\})$ where

$$\mathcal{M}(x) = \{p_i : x_i = 1\}$$

where p_i denotes the i -th party.

Define $x \leq y$ for $x, y \in \{0, 1\}^n$, $x_i \leq y_i$ for all $1 \leq i \leq n$.

A function $F : \{0, 1\}^n \rightarrow \{0, 1\}$ is called *monotone* if $x \leq y$ implies $F(x) \leq F(y)$.

Definition 7 (Secret Sharing for a monotone function). Let $F : \{0, 1\}^n \rightarrow \{0, 1\}$ be a monotone function. Let **Gen** be a share-generation algorithm where $(sh_1, sh_2, \dots, sh_n) \leftarrow \text{Gen}(1^n, F, a)$ (where a denotes a secret).

We say that a subset $S \subseteq \{p_1, p_2, \dots, p_n\}$ of parties is *authorized* if $F(\mathcal{M}^{-1}(S)) = 1$.

Let **Rec** be a reconstruction algorithm.

We say that a secret sharing scheme defined by algorithms **Gen** and **Rec** is *correct* if for all authorized S ,

$$\text{Rec}(S, \{sh_i\}_{p_i \in S}) = a.$$

So, we are secret sharing a secret a according to a monotone function (access-structure).

How does this definition apply to Shamir's secret sharing? Well, the monotone function just checks if the number of parties is at least t . That is,

$$F(x) = \begin{cases} 1 & \text{if } \sum_{i=1}^n x_i \geq t \\ 0 & \text{otherwise} \end{cases}.$$

Why monotone functions? Since we want adding more colluding parties to an already authorized set to not change the result.

Now, note that every circuit C over $\{\wedge, \vee\}$ gates is monotone (adding more bits to 1's doesn't decrease the result).

We want to construct a secret sharing scheme for a circuit C over $\{\wedge, \vee\}$ of polynomial size. First, let's define the size of a formula.

Definition 8 (Size of a formula). The size of a formula $\phi(x_1, x_2, \dots, x_n)$ is the number of operands in ϕ .

For example, the size of the formula

$$x_1 \wedge (x_2 \vee x_3 \vee x_4)$$

is 4.

Now, we want to construct a secret sharing scheme for a formula φ .

1. If $\varphi = \varphi_0 \vee \varphi_1$, then you secret share a according to φ_0 , and secret share a according to φ_1 .
2. If $\varphi = \varphi_0 \wedge \varphi_1$, you sample $r \leftarrow \{0, 1\}$ uniformly at random, and secret share r according to φ_0 , and secret share $a \oplus r$ according to φ_1 .

This is a recursive construction.

Now, consider the class of circuits where the indegree of each gate is 2, and the outdegree of each gate is 1, and every input of a gate in level i is an output of a gate in level $i - 1$ (where level 0 corresponds to the input wires).

The gates are only $\wedge$ and $\vee$.

It can be shown that this class of circuits is equivalent to the whole class of circuits that have $\wedge$ and $\vee$ gates.

Now, can we construct a polynomial-size circuit for Shamir's secret sharing (i.e. that outputs 1 only when the number of 1's in the input is at least t)? This question is open.

5 Semi-Honest Security for MPC

We want to define semi-honest security. The idea is that if you have MPC where t parties are corrupt, whatever the t parties learn by sharing information with each other will not leak anything about the inputs of the other parties.

Before we define this formally, we need to discuss some terminology.

We denote a protocol by Π , and a *transcript* of the protocol is the sequence of messages sent by two parties.

Note that the transcript is a random variable determined by the randomness in the messages of the parties, but if we fix some random inputs, it will be deterministic.

6 Recap

Let's talk about secret sharing for a bit. Suppose we have a circuit $C : \{0, 1\}^n \rightarrow \{0, 1\}$ that is monotone (recall the definition of monotone).

We talked about sharing a secret s according to a circuit C . Suppose C takes inputs $x_1, x_2, \dots, x_n$, and note that for any x where $C(x) = 1$, we have to have $\text{Reconst}(\{(i, \text{sh}_i) : x_i = 1\}) = s$ (correctness).

We also need security; which means that for any x where $C(x) = 0$, and any two secrets s and s' , suppose that $(\text{sh}_1, \text{sh}_2, \dots, \text{sh}_n) \leftarrow \text{Share}(s)$ and $(\text{sh}'_1, \text{sh}'_2, \dots, \text{sh}'_n) \sim \text{Share}(s')$, then the $(s, s', \{(i, \text{sh}_i) : x_i = 1\})$ and $(s, s', \{(i, \text{sh}'_i) : x_i = 1\})$ are identical.

Recall that we shared a secret using a top-down approach, where we considered $\varphi = \varphi_0 \wedge \varphi_1$ or $\varphi = \varphi_0 \vee \varphi_1$, each with specific shares.

Now, we will talk about polynomial size circuit/formula (the definition of size according to operands or operations doesn't matter since they're "polynomially" equivalent). Also, note that we can define formulas in the standard inductive way. So, let's stick to the number of operations/gates

Definition 9 (Polynomial Size Formula Circuit). A *polynomial-sized formula circuit* is a formula $C : \{0, 1\}^n \rightarrow \{0, 1\}$ such that

1. $|C| = \text{poly}(n)$, where $|C|$ denotes the number of gates (or the size of the circuit)
2. The fan-out of each gate is 1.

We note that in some cases, if the second condition is not satisfied, sometimes the size of the shares can be exponential.

Claim 6.1. Any polynomial-sized formula $\varphi : \{0, 1\}^n \rightarrow \{0, 1\}$ has an associated polynomial-sized circuit-formula $C : \{0, 1\}^n \rightarrow \{0, 1\}$ where $C(x) = \varphi(x)$ for all $x \in \{0, 1\}^n$.

Also, any polynomial-sized circuit $C : \{0, 1\}^n \rightarrow \{0, 1\}$ has an associated polynomial-sized formula $\varphi : \{0, 1\}^n \rightarrow \{0, 1\}$ where $\varphi(x) = C(x)$ for all $x \in \{0, 1\}^n$.

Proof. The transition from a formula to a polynomial-sized circuit formula is just by simple induction over the shape of the formula. And polynomiality is preserved.

Now, we note if the fan-out of a gate is allowed to be 2, just thinking purely syntactically, we can have a recursive formula as

1. $\varphi_{0,0} = x_1, \varphi_{0,1} = x_2,$
2. $\varphi_{1,0} = \varphi_{0,0} \wedge \varphi_{0,1}, \varphi_{1,1} = \varphi_{0,0} \vee \varphi_{0,1},$
3. $\varphi_{2,0} = \varphi_{1,0} \wedge \varphi_{1,1}, \varphi_{2,1} = \varphi_{1,0} \vee \varphi_{1,1},$

$$4. \varphi_{3,0} = \varphi_{2,0} \wedge \varphi_{2,1}, \varphi_{3,1} = \varphi_{2,0} \vee \varphi_{2,1},$$

$$5. \vdots$$

the size of the formula syntactically will double each time.

However, since we will show that the fact that the fan-out of each circuit is 1, we can convert any circuit to a formula with polynomial size.

PROOF NEEDED

□

Another class of circuits that we needed to talk about are called NC^1 circuit:

Definition 10 (NC^1 Circuit). An NC^1 circuit is a circuit $C : \{0, 1\}^n \rightarrow \{0, 1\}$ such that

1. $|C| = \text{poly}(n)$
2. the depth of C is $O(\log n)$ (the depth is the longest path from an input wire to the output wire)

Claim 6.2. Let the set of all NC^1 circuits be denoted by NC^1 , and Polynomial size formula-circuits as PFORM . Then, $\text{NC}^1 = \text{PFORM}$.

Proof. **PROOF NEEDED**

□

There's also a kind of circuits constructed by Andrew Yao called the *Yao's Garbled Circuits*.

7 Semi-Honest Security

First, we present the definition for two parties, and then we can generalize it to multiple parties (the generalization is simple).

Remember that the BGW protocol doesn't work for two parties, we have to have at least $n \geq 2t + 1$.

The main principle is that in each party P_i , we care about, after the protocol ends, what does the party have? The view in then protocol can be simulated from its input and output.

For this, let's introduce some definitions:

Definition 11. Let Π be a communication protocol between two parties P_0 and P_1 ;

1. The transcript $\text{tr}^\Pi(n, x_0, x_1)$ of the protocol Π is the sequence of messages sent back and forth between $P_0(1^n, x_0)$ and $P_1(1^n, x_1)$ on security parameter n .
2. The view of a party P_i in the protocol Π , $\text{view}_i^\Pi(n, x_0, x_1)$ for $i \in \{0, 1\}$ is the tuple $(x_i, r_i, \text{tr}^\Pi(n, x_0, x_1))$ where $r_i \in \{0, 1\}^{p(n)}$ (for some polynomial p) is a factor of randomness for each party (sampled uniformly at random).

Now, we can define semi-honest security.

Definition 12 (Semi-honest security). Let P_0 and P_1 be two parties, where P_0 has input x_0 and P_1 has input x_1 , and P_0 wants to learn $f_0(x_0, x_1)$ and P_1 wants to learn $f_1(x_0, x_1)$.

A two-party protocol Π is semi-honest secure for a functionality

$$(x, y) \mapsto (f_0(x, y), f_1(x, y))$$

if there exists a PPT simulation algorithm Sim such that for any $i \in \{0, 1\}$ and any two inputs $(x_0, x_1) \in \{0, 1\}^2$, we have

$$\text{Sim}(i, x_i, f_i(x_0, x_1)) \approx \text{view}_i^\Pi(n, x_0, x_1)$$

where $\approx$ denotes statistical or computational indistinguishability.

Note that the simulation algorithm Sim here is meant to capture the following idea.

- No party i , that knows i, x_i , and $f_i(x_0, x_1)$ can deviate from the protocol.

For example, let's assume party P_0 is the malicious party. Then, party P_0 knows $0, x_0$, and $f_0(x_0, x_1)$. Then, any simulation algorithm Sim that party P_0 can come up with that takes input $(0, x_0, f_0(x_0, x_1))$ should know no more than its view, $\text{view}_0^\Pi(n, x_0, x_1)$.

This seems like an abstract definition. Let's define a more concrete definition. Consider the following experiment:

1. The adversary Adv submits (x_0, x_1) and $i \in \{0, 1\}$
2. We sample a random bit $b \leftarrow \{0, 1\}$.

3. If $b = 0$, we sample $v \leftarrow (\text{Sim}(n, i, x_i, f_i(x_0, x_1)), f_i(x_0, x_1))$;
if $b = 1$, we sample $v \leftarrow (\text{view}_i^\Pi(n, x_0, x_1), f_i(x_0, x_1))$.
4. We give v to Adv , and Adv outputs b' .
5. We say that the output of the experiment is 1 if $b = b'$, and we denote this by $\text{PrivK}_{\text{Adv}, \Pi} = 1$ (otherwise, the output is 0).

This introduces the following definition:

Definition 13 (Alternative Definition of Semi-Honest Security). We say that a protocol Π is *semi-honest computationally secure* if for all PPT adversaries Adv ,

$$\Pr[\text{PrivK}_{\text{Adv}, \Pi} = 1] \leq \frac{1}{2} + \text{negl}(n).$$

If Adv is not computationally bounded, then we call it *statistically secure*. We call the protocol *semi-honest perfectly secure* if for all adversaries (not necessarily PPT) Adv ,

$$\Pr[\text{PrivK}_{\text{Adv}, \Pi} = 1] = \frac{1}{2}.$$

The experiment basically mimicks the first definition of semi-honest security.

Let's consider a simpler definition that doesn't have a simulation algorithm.

1. Suppose $f_0 = f_1 = f$ (both parties want to compute the same function).
2. The adversary Adv submits (x_1, y_1) and x such that $f(x, x_1) = f(x, y_1)$.
3. We sample $b \leftarrow \{0, 1\}$. If $b = 0$, we sample $v \leftarrow \text{view}_0^\Pi(x, x_1)$. If $b = 1$, we sample $v \leftarrow \text{view}_0^\Pi(x, y_1)$.
4. v is given to Adv , and Adv outputs b' .
5. We say that the output of the experiment is 1 if $b = b'$, and we denote this by $\text{PrivK2}_{\text{Adv}, \Pi} = 1$ (otherwise, the output is 0).

Same as the definition above, the protocol is *semi-honest computationally secure* if for all PPT adversaries Adv ,

$$\Pr[\text{PrivK2}_{\text{Adv}, \Pi} = 1] \leq \frac{1}{2} + \text{negl}(n).$$

In this definition, the adversary is considered to be in the place of the first party P_0 , and is trying to distinguish between two inputs of the second party P_1 ; the protocol will be considered secure when the adversary can't distinguish between the two inputs.

Now, we note that 1-round MPC cannot satisfy the above definition.

Claim 7.1. 1-round MPC cannot satisfy the above definition for any non-trivial f .

We call a function f non-trivial if there exists $x_0, y_0, x_1, y_1 \in \{0, 1\}$ such that

1. $f(x_0, x_1) = f(x_0, y_1)$, and
2. $f(y_0, x_1) \neq f(y_0, y_1)$.

An example of a trivial function is the exclusive OR, since

$$\begin{aligned} x_0 \oplus x_1 = x_0 \oplus y_1 &\implies x_1 = y_1 \\ y_0 \oplus x_1 \neq y_0 \oplus y_1 &\implies x_1 \neq y_1 \end{aligned}$$

and so such choice of x_0, y_0, x_1, y_1 would be impossible.

The definition of a non-trivial function a bit weird, but it motivates the claim that a 1-round MPC is not secure. We will see why the definition looks this way.

In particular, suppose we're doing 1-round MPC. The setup is as follows:

1. Two parties P_0 and P_1 have inputs x_0 and x_1 respectively, and they want to compute $f(x_0, x_1)$. However, party P_0 is malicious, and wants to reveal information about x_1 .
2. P_0 sends a message m_0 to P_1 and P_1 sends a message m_1 to P_0 simulatenously, meaning that m_0 only depends on x_0 and m_1 only depends on x_1 .
3. The malicious party P_0 changes its input from x_0 to y_0 , and differentiates between two inputs x_1 and y_1 of party P_1 .

If the protocol is correct, P_0 only from (x_0, r_0, m_1) can learn $f(x_0, x_1)$ but P_0 can change its input after the protocol since m_1 only depends on x_0 .

This means that from (y_0, r_0, m_1) , P_0 can learn $f(y_0, x_1)$ as well.

So, we need a function f where,

1. to attack, we can submit x_0 as input for P_0 and inputs x_1, y_1 where $f(x_0, x_1) = f(x_0, y_1)$ and
2. changing the input from x_0 to y_0 , we should be able to differentiate between x_1 and y_1 , meaning $f(y_0, x_1) \neq f(y_0, y_1)$.

This is exactly the description of a non-trivial function.

This motivates the following attack (that illustrates how we can break this definition of security): let $x_0, y_0, x_1, y_1 \in \{0, 1\}$ be those that satisfy the conditions of the non-trivial function f .

1. The adversary **Adv** submits x_0 and (x_1, y_1) (noting that $f(x_0, x_1) = f(x_0, y_1)$)
2. We sample $b \leftarrow \{0, 1\}$. If $b = 0$, we sample $v \leftarrow \text{view}_0^\Pi(x_0, x_1)$. If $b = 1$, we sample $v \leftarrow \text{view}_0^\Pi(x_0, y_1)$.
3. The adversary gets the view $v = (x_0, r, (m_0, m_1))$. Supposing the input to the second party is x , the adversary can compute $f(y_0, x)$ solely from m_1 (which only depends on x).
4. Since the values of $f(y_0, x_1)$ and $f(y_0, y_1)$ are not equal, the adversary submits $b' = 0$ if $f(y_0, x) = f(y_0, x_1)$ and $b' = 1$ if $f(y_0, x) = f(y_0, y_1)$.

This attack will succeed with probability 1 since the function is non-trivial.

8 To think about for next lecture

In Section 7, we defined what it means for a two-party MPC protocol to be secure. To prove security of the BGW scheme, we need to extend this definition to include n -party protocols, where t -out-of- n parties behave honestly (i.e., do not collude), but the remaining $n - t$ parties are corrupt (i.e. collude with each other). Our extended security definition must capture the following points:

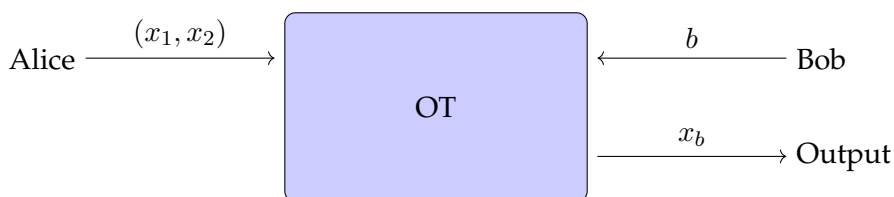
- All n parties follow the protocol; that is, all n parties follow the protocol exactly as it is written.
- $n - t$ of the parties collude with each other; that is, $n - t$ of the parties share their views with each other, even though they are supposed to be kept private.
- Even after sharing their views with each other, the $n - t$ honest-but-curious parties still are not able to determine anything else, other than the final output of the protocol.

To model this with a security experiment, we give the adversary access to the views of each of the corrupt parties (these may be real views or simulated views, depending on the challenge bit) to mimic what one of those parties would be able to figure out. If the adversary is still unable to distinguish whether the set of views is real or simulated, then this means that the set of real views does not allow any other information to be computed (otherwise the adversary would be able to use this information to distinguish this set of views from a simulated set of views for which it is impossible to learn anything meaningful), and hence the protocol is secure.

In the next lecture, we will use this extended security definition to prove that the BGW scheme is secure. In order to do this, we need to write a simulation which will output 'views' for the corrupted parties with the correct distribution. This is mentioned a lecture early so that we have time to think about it before being handed the solution.

9 Oblivious Transfer

In this section, we'll discuss oblivious transfer (OT) protocols. In an oblivious transfer protocol, Alice has two secret values, x_0 and x_1 . Bob wants to learn one of x_0 and x_1 (indicated by a bit b), but he doesn't want Alice to know the value of b . On the other hand, Alice is happy to share x_b with Bob, but she doesn't want Bob to learn the value of x_{1-b} . Pictorially, speaking, the situation looks like this:



The security requirements are exactly what is stated above:

- **Security for Alice:** Bob should not learn anything about x_{1-b} .
- **Security for Bob:** Alice should not learn anything about b .

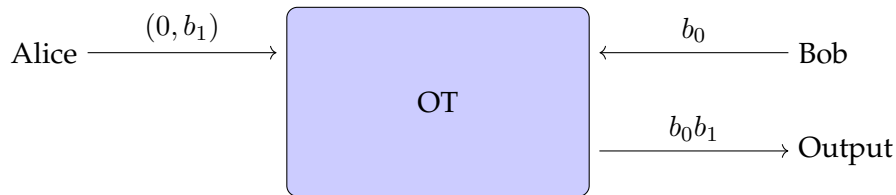
There exist secure OT protocols, and we'll talk about some later in the course, but for now, suppose one exists and treat it like a blackbox. For the remainder of the lecture, we'll show how, using this blackbox OT protocol, we can construct secure 2PC for any circuit.

9.1 Secure AND from OT

First, we'll start by constructing a secure two-party AND protocol using our OT blackbox. Suppose Alice has bit b_1 and Bob has bit b_0 , and they want to compute $b_0 \wedge b_1$, without learning anything that can't be deduced from the output $b_0 \wedge b_1$. Notice the following:

- The only way to obtain an output of 1 is if $b_0 = b_1 = 1$. So, if the output is 1, Alice and Bob can both deduce each others' secret bit.
- If $b_0 = 1$, then the output will be 1 regardless of the value of b_1 . So, in this case, Bob cannot deduce anything about Alice's bit. The same holds in reverse.

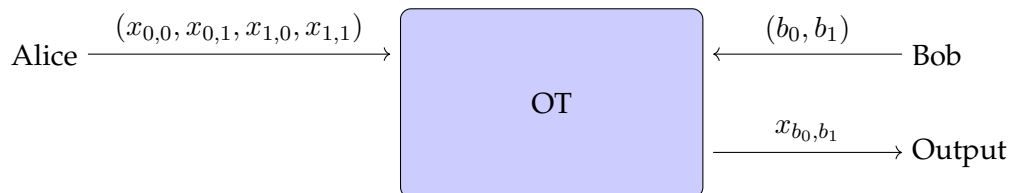
Based on these two facts, we only need to keep the Alice and Bob's inputs secret if they are not both 1. So, we can use our OT blackbox as follows:



If $b_0 = 0$, then the OT protocol will return Alice's first input, 0. Regardless of the value of b_1 , this is the correct output, since for all x , $0 \wedge x = 0$. Otherwise, if $b_0 = 1$, then the OT protocol will return Alice's second input, b_1 . Again, regardless of the value of b_1 , this is the correct output, since for all x , $1 \wedge x = x$.

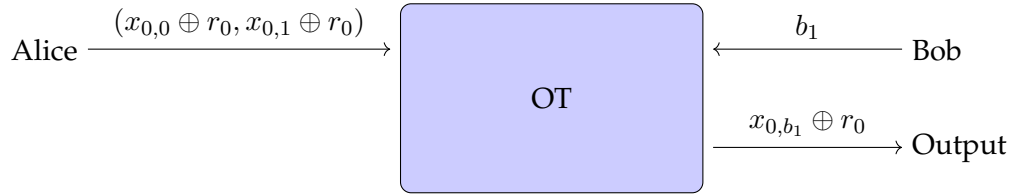
9.2 (1, 4)-OT

Now, we'll construct a secure (1, 4)-OT protocol; that is, an OT protocol in which Alice has 4 inputs, and Bob specifies the one he wants using two bits. Just like our (1, 2)-OT protocol, Alice will not learn either of Bob's bits, and Bob will only learn one of Alice's inputs. Ultimately, we want an OT protocol that looks like this:



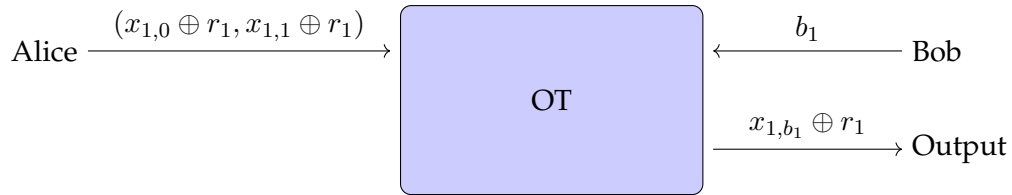
Naively, Alice and Bob could simply run the blackbox OT protocol twice (first using $x_{0,0}$, $x_{0,1}$, and b_0 , and next using $x_{1,0}$, $x_{1,1}$, and b_1), but then Bob will learn both x_{0,b_0} and x_{1,b_1} . To fix this, we can introduce randomness. Suppose Alice chooses random values r_0 and r_1 . Then, we can construct a 3-round OT protocol as follows:

Round 1:



After this round, Bob knows $x_{0,b_1} \oplus r_0$. Since r_0 is chosen randomly, this completely masks the value of x_{0,b_1} , so Bob learns nothing about any of Alice's inputs.

Round 2:



After this round, Bob knows both $x_{0,b_1} \oplus r_0$ and $x_{1,b_1} \oplus r_1$. Since r_0 and r_1 are independently chosen random values, they not only mask the values x_{0,b_1} and x_{1,b_1} , but they also ensure that the two masked values are independent of each other. So, Bob still learns nothing about any of Alice's inputs.

Round 3:



After this round, Bob knows $x_{0,b_1} \oplus r_0$ and $x_{1,b_1} \oplus r_1$, and r_{b_0} . Using this information, he can now unmask the value of x_{b_0,b_1} by XOR-ing the appropriate value with r_{b_0} . However, he still cannot learn anything about x_{1-b_0,b_1} since r_0 and r_1 are independent. So, this is a secure $(1, 4)$ -OT protocol.

9.3 General Secure Computation (GMW)

The goal now is to show how two parties can securely evaluate any circuit, using the tools we developed above. Suppose Alice and Bob have a circuit $C : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}$ which uses AND, XOR, and NOT gates. Suppose further that Alice has input $x \in \{0, 1\}^n$, Bob has input

$x' \in \{0, 1\}^n$, and that they want to jointly evaluate $C(x, x')$ such that Bob learns the final output value, and neither of them learn anything about each others' input values. We can do this using the BGW protocol, which was first developed by Goldreich, Micali, and Wigderson, in their 1987 paper "How to play ANY mental games." [Goldreich et al. \[1987\]](#)

At a high level, the protocol will consist of three steps:

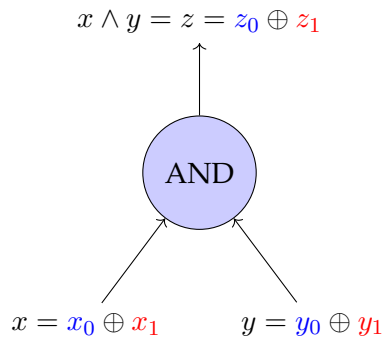
1. **Initial secret sharing phase.** To do this, both of Alice and Bob will split each of their input bits into two random shares (i.e., r and $b \oplus r$ for bit b and random r). They will keep the share $r \oplus b$ for themselves, and give the other share r to the other party. At the end of this phase, they will each control one share of each input wire.
2. **Computing on secret-shared data.** In this step, Alice and Bob will use OT to compute the outputs of AND, XOR, and NOT gates. For each gate, they each control one share of each input wire, and at the end of the computation, they will each control one share of the output wire of the gate. By keeping this invariant, we ensure that the last step will can be completed. It is worth noting that having only Bob learn the output of OT is useful here, since it ensures that Alice will not know his share out the output wire of each gate.
3. **Pooling together the final shared data.** Since Alice and Bob will each control one share of the final output wire, they can simply XOR their shares together to compute the final result. Since Bob is supposed to learn the final value but Alice isn't, Alice will send her share over to Bob so that he can perform the final XOR by himself, without Alice knowing his share.

In the extended case of n parties, we will want our protocol to be secure under the collusion of $n - 1$ honest-but-curious parties.

To define the protocol, we need to show how to compute AND, XOR, and NOT gates, for secret-shared values. From here on, we will adopt the convention that all of Alice's shares are written in [blue](#), and all of Bob's shares are written in [red](#), for ease of readability.

9.3.1 AND Gates

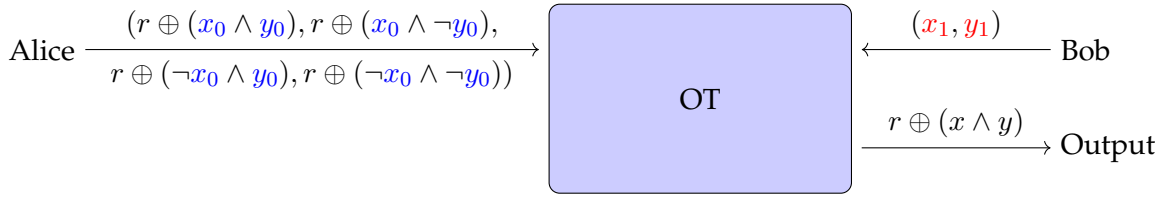
Suppose Alice and Bob want to compute an AND gate, where they each have control of one share of each of the input bits. In other words, they want to compute the following gate:



Let's see how Alice and Bob can do this. First, consider Bob's share of x_1 of x . Notice that if $x_1 = 0$, then $x = x_0$, and if $x_1 = 1$, then $x = \neg x_0$. The same goes for y . So, the value of $x \wedge y$ can be determined from Alice's shares, based on the following table:

Bob's shares (x_1, y_1)	Output Value $x \wedge y$
$(0, 0)$	$x_0 \wedge y_0$
$(0, 1)$	$x_0 \wedge \neg y_0$
$(1, 0)$	$\neg x_0 \wedge y_0$
$(1, 1)$	$\neg x_0 \wedge \neg y_0$

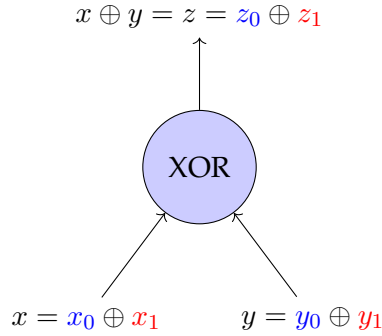
As a naive first attempt, we could use $(1, 4)$ -OT to solve obtain the output value. This way, Bob's share of the output value can simply be z , and Alice can set her share to 0. Alice learns nothing about Bob's shares... but Bob learns z , which he can use to deduce information about Alice's input shares. So, this naive attempt is insecure. However, the idea can be salvaged by adding randomness. Suppose Alice samples a random value r , and consider the following $(1, 4)$ -OT scheme:



Bob receives the masked output $r \oplus (x \wedge y)$, and Alice keeps r as her share of the output. Clearly, these are additive shares of $x \wedge y$, and since r is chosen randomly, neither Alice nor Bob can deduce anything about each others' share. So, this is a secure protocol for jointly computing AND gates.

9.3.2 XOR Gates

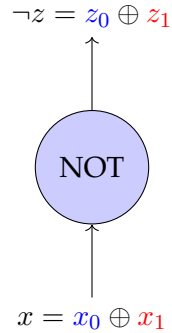
Now, suppose Alice and Bob want to compute an XOR gate, where they each have control of one share of each of the input bits. In other words, they want to compute the following gate:



Jointly computing this gate is trivial, since XOR is both associative and commutative. Alice and Bob can each locally add their shares of x and y to compute their share of z ; that is, Alice computes $z_0 = x_0 \oplus y_0$, and Bob computes $z_1 = x_1 \oplus y_1$. Since the shares of z were computed entirely locally using only local inputs, this is secure.

9.3.3 NOT Gates

Finally, suppose Alice and Bob want to compute a NOT gate, where they each have control of one share of each of the input bits. In other words, they want to compute the following gate:



Jointly computing this gate is also trivial. Alice can locally compute her share of z as $z_0 = \neg x_0$, and Bob can locally compute his share of z as $z_1 = x_1$ (this was a very easy computation for him). These are additive shares of z , since

$$z_0 \oplus z_1 = \neg x_0 \oplus x_1 = \neg(x_0 \oplus x_1) = \neg x = z.$$

Again, since the shares of z were computed entirely locally using only local inputs, this is secure.

10 Equivalence of Semi-Honest Security Definitions

Recall that we have the following definitions of semi-honest security for the two-party case. In both cases, let λ be a security parameter, and let P_0 and P_1 be two parties, where P_0 has input x_0 and P_1 has input x_1 , and both parties would like to compute the output $f(x, y)$. Suppose further that $f : X \times Y \rightarrow Z$ is such that for all $x \in X, y \in Y, z \in Z$ with $f(x, y) = z$, there exists $x' \in X$ such that $f(x', y) = z$.

Definition 14 (Semi-honest security (indistinguishability definition)). Fix y , and let x and x' be such that $f(x, y) = f(x', y)$. We say that a 2PC protocol Π satisfies semi-honest security if for $i \in \{0, 1\}$,

$$(\text{view}_i^\Pi(\lambda, x, y), f(x, y)) \cong (\text{view}_i^\Pi(\lambda, x', y), f(x', y)).$$

Definition 15 (Semi-honest security (simulation-secure definition)). Let Π be a 2PC protocol that evaluates f , and consider the following experiment:

$\text{Exp}_\Pi^{\text{sim-secure}}(\mathcal{A})$	$ \begin{aligned} &1: (x_0, x_1) \leftarrow \$\mathcal{A}() \\ &2: i \in \{0, 1\} \leftarrow \$\mathcal{A}() \\ &3: b \leftarrow \$\{0, 1\} \\ &4: \text{if } b = 0 : v \leftarrow (\text{Sim}(\lambda, i, x_i, f(x_0, x_1)), f(x_0, x_1)) \\ &5: \text{if } b = 1 : v \leftarrow (\text{view}_i^\Pi(\lambda, x_{1-i}, x_i), f(x_0, x_1)) \\ &6: b' \leftarrow \$\mathcal{A}(v) \\ &7: \text{return } \llbracket b' = b \rrbracket \end{aligned} $
---	--

We say that Π satisfies semi-honest security if for all PPT adversaries $\mathcal{A}$,

$$\left| \Pr[\text{Exp}_\Pi^{\text{sim-secure}}(\mathcal{A}) \implies 1] - \frac{1}{2} \right| \leq \epsilon,$$

where ϵ is negligible.

In this section we will prove that these two definitions are equivalent. However, there is a small caveat in that for one of the implications, we also require that f is *reverse sampleable*.

Definition 16 (Reverse sampleable). We say a function $f : X \times Y \rightarrow Z$ is reverse sampleable in its first input if for all $x \in X$ and $z \in Z$, there exists a PPT algorithm RS_0 such that $\text{RS}_0(x, z) = y$, where $f(x, y) = z$. Similarly, we say a function $f : X \times Y \rightarrow Z$ is reverse sampleable in its second input if for all $y \in Y$ and $z \in Z$, there exists a PPT algorithm RS_1 such that $\text{RS}_1(y, z) = x$, where $f(x, y) = z$.

As it turns out, most function we'll be considering in this course satisfy this property. For example, consider an OT protocol OT such that $\text{OT}(b, (m_0, m_1)) = (m_b, \perp)$. To show that this is reverse sampleable in the first input, we must argue that given (m_0, m_1) and $\perp$, there exists x such that $\text{OT}(x, (m_0, m_1)) = ([\text{anything}], \perp)$. Any choice of x works. To show that this is reverse sampleable in the second input, we must show that given (b, m_b) , there exists x such that $\text{OT}(b, x) = (m_b, \perp)$. We can take $x = (m_b, m_b)$.

Proof of equivalence:

First, we'll show that Definition 14 implies 15 (with a small caveat). For this, suppose that f is reverse sampleable in both inputs, and suppose that Π is a 2PC protocol such that for $i \in \{0, 1\}$,

$$(x, r, \text{tr}^\Pi(\lambda, x, y), f(x, y)) \cong (x', r, \text{tr}^\Pi(\lambda, x', y), f(x', y)).$$

For each $i \in \{0, 1\}$, we need to construct an algorithm Sim as in Definition 15, that outputs a value v indistinguishable from the view of party i . We'll show a simulator for $i = 0$; the case where $i = 1$ is identical (apart from changing a few variable names). Our simulator works as follows:

```

Sim( $\lambda, 0, y, f(x, y)$ )
-----
1 :  $x' \leftarrow \$ \text{RS}_0(x, f(x, y))$ 
2 :  $v \leftarrow \$ \text{view}_0^\Pi(\lambda, x', y)$ 
3 : return  $v$ 

```

By Definition 14, this is indistinguishable from $\text{view}_0^\Pi(\lambda, x, y)$, hence Definition 15 is satisfied.

Next, we need to show that Definition 15 implies Definition 14. For this direction, we do not need that f is reverse sampleable, and once again we will only show one case, this time for $i = 1$ (and note that the other case is identical up to variable names). Suppose there exists a simulator Sim for Π , as in Definition 15. Since $f(x, y) = f(x', y)$, then

$$\text{Sim}(\lambda, 0, y, f(x, y)) = \text{Sim}(\lambda, 0, y, f(x', y)).$$

But

$$\text{Sim}(\lambda, 0, y, f(x, y)) \cong \text{view}_0(\lambda, x, y)$$

and

$$\text{Sim}(\lambda, 0, y, f(x', y)) \cong \text{view}_0(\lambda, x', y)$$

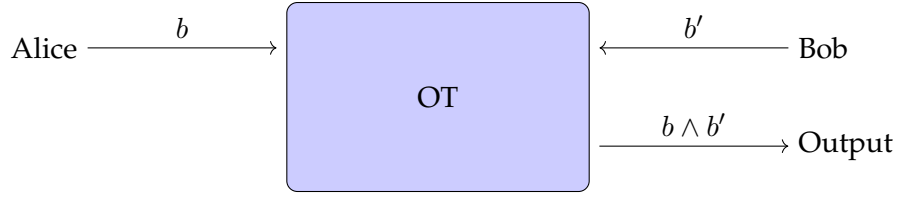
by Definition 15, hence

$$\text{view}_0(\lambda, x, y) \cong \text{view}_0(\lambda, x', y)$$

and Definition 14 is satisfied.

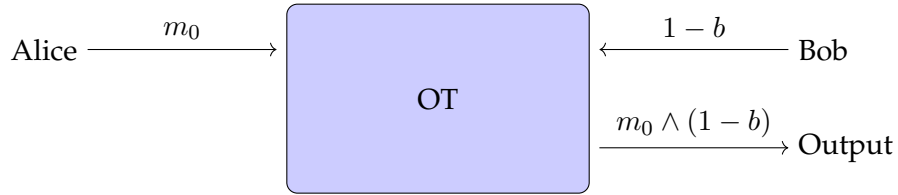
11 Building Secure OT from Secure AND

Suppose we have a protocol that allows Alice (with input bit a) and Bob (with input bit b) to evaluate $a \wedge b$ in such a way that Alice learns nothing, and Bob learns $a \wedge b$, but neither of them learn any more information about each others' input bits than their output leaks. Pictorially, that would be this:



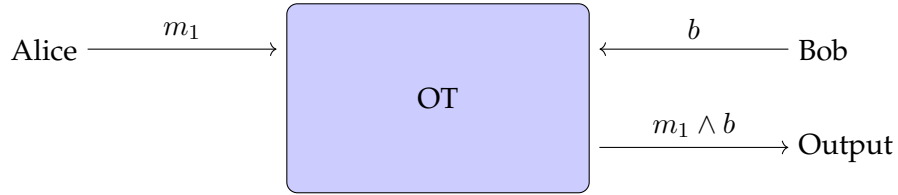
In this section, we will show how to implement a secure OT protocol using this AND protocol as a blackbox. For this, suppose Alice has input bits m_0 and m_1 , and Bob has input bit b . We can implement OT with two calls to the AND blackbox, as follows:

Round 1:



After this round, Bob learns $m_0 \wedge (1 - b)$. If $b = 0$, Bob learns m_0 , which he is supposed to learn in this case, so this is okay. If $b = 1$, then Bob receives back the bit 0 (regardless of the value of m_0), which tells him nothing useful.

Round 2:



After this round, Bob learns $m_1 \wedge b$. If $b = 0$, then he receives back the bit 0 (regardless of m_1), which tells him nothing useful. If $b = 1$, Bob learns m_1 , which he is supposed to learn in this case, so this is okay.

Preview of homework: Show how to use a symmetric AND protocol (i.e. both parties learn $a \wedge b$) to secure OT.

12 Additively Homomorphic Encryption

Recall the definition of a public key encryption scheme:

Definition 17. Public key encryption A public key encryption scheme Π consists of a key space $\mathcal{K}$, a probability distribution Σ , a message space $\mathcal{M}$, a ciphertext space $\mathcal{C}$, and a tuple of algorithms $(\text{KGen}, \text{Enc}, \text{Dec})$, where

- $\text{KGen}(\lambda)(\text{pk}, \text{sk})$ is a probabilistic key generation algorithm, which takes as input a security parameter λ and outputs a tuple $(\text{pk}, \text{sk}) \in \mathcal{K}$ consisting of a public key/secret key pair.
- $\text{Enc}(\text{pk}, m; r) \rightarrow c$ is a deterministic polynomial time encryption algorithm that takes as input a public key pk , a message $m \in \mathcal{M}$ and random coins r sampled from Σ , and outputs a ciphertext $c \in \mathcal{C}$, and outputs a ciphertext $c \in \mathcal{C}$.
- $\text{Dec}(\text{sk}, m; r) \rightarrow m$ is a deterministic polynomial time decryption algorithm that takes as input a secret key sk , a ciphertext $c \in \mathcal{C}$ and random coins r sampled from Σ , and outputs either a message $m \in \mathcal{M}$ or a distinguished error symbol $\perp$.

A public key encryption scheme is additively homomorphic if Enc is a homomorphism that is additive on the message space. Formally:

Definition 18. Additively homomorphic encryption Let Π be a public key encryption scheme. We say that Π is additively homomorphic if there exists an algorithm $\text{Hom} : \mathcal{K}_{\text{pk}} \times \mathcal{C} \times \mathcal{C} \rightarrow \mathcal{C}$ such that for any $(\text{pk}, \text{sk}) \leftarrow \$ \text{KGen}(1^\lambda)$ and for all b_0, b_1, r_0, r_1 ,

$$(\text{pk}, \text{sk}, b_0, b_1, r_0, r_1, \text{Hom}(\text{pk}, c_0, c_1)) \cong^c (\text{pk}, \text{sk}, b_0, b_1, r_0, r_1, \text{Enc}(\text{pk}, b_0 \oplus b_1)),$$

where $c_0 = \text{Enc}(\text{pk}, b_0; r_0)$ and $c_1 = \text{Enc}(\text{pk}, b_1; r_1)$.

Put simply, the function Hom allows you to compute an encryption $\text{Enc}(\text{pk}, b_0 \oplus b_1)$ (under *some* randomness r), given pk , $\text{Enc}(\text{pk}, b_0)$ and $\text{Enc}(\text{pk}, b_1)$.

Example 1. Define $f_{m_0, m_1} : \{0, 1\} \rightarrow \{0, 1\}$ by $f_{m_0, m_1}(b) = m_0 b + m_1$. Given a public key pk , two bits m_0 and m_1 , and the encryption $\text{Enc}(\text{pk}, b)$ of a bit b (where Enc is additively homomorphic), we can compute $E(\text{pk}, f_{m_0, m_1}(b))$ as

$$\begin{cases} \text{Hom}(\text{pk}, \text{Enc}(\text{pk}, b), \text{Enc}(\text{pk}, m_1)), & \text{if } m_0 = 0 \\ \text{Hom}(\text{pk}, \text{Enc}(\text{pk}, 0), \text{Enc}(\text{pk}, m_1)), & \text{if } m_0 = 1 \end{cases}.$$

We have that

$$\text{Enc}(\text{pk}, f_{m_0, m_1}(b)) = \text{Enc}(\text{pk}, m_0 b + m_1) = \text{Hom}(\text{pk}, \text{Enc}(\text{pk}, m_0 b), \text{Enc}(\text{pk}, m_1)).$$

Since

$$m_0 = 0 \implies \text{Enc}(\text{pk}, m_0 b) = \text{Enc}(\text{pk}, 0)$$

and

$$m_0 = 1 \implies \text{Enc}(\text{pk}, m_0 b) = \text{Enc}(\text{pk}, b),$$

we can compute $\text{Enc}(\text{pk}, m_0 b)$ from the information given. We can also compute $\text{Enc}(\text{pk}, m_1)$ from the information given. So, we can compute $\text{Hom}(\text{pk}, \text{Enc}(\text{pk}, m_0 b), \text{Enc}(\text{pk}, m_1))$ from the information given.

12.1 Building OT from additively homomorphic encryption

Suppose the receiver has the single bit input b , the sender has the input (m_0, m_1) , where m_0 and m_1 are both bits, and the sender wants to obtain the output m_b . Also, suppose $(\text{KGen}, \text{Enc}, \text{Dec}, \text{Hom})$ is an IND-CPA secure additively homomorphic encryption scheme. Then we can create an OT protocol as follows:

1. Receiver computes $(\text{pk}, \text{sk}) \leftarrow \$ \text{KGen}(1^\lambda)$.
2. Receiver sends $(\text{pk}, \text{Enc}(\text{pk}, b))$ to the sender.
3. Using the technique illustrated in example 1, the sender computes $c = \text{Enc}(\text{pk}, (1 + b)m_0 + bm_1)$, and sends c to the receiver.
4. Receiver computes $m_b = \text{Dec}(\text{sk}, c)$.

First, note that this protocol does in fact return m_b , since

$$b = 0 \implies (1 + b)m_0 + b(m_1) = m_0$$

and

$$b = 1 \implies (1 + b)m_0 + m_1 = m_1.$$

Now, we must argue security. From the receiver's point of view, they only ever see m_b , as the only value they receive is independent of m_1 . From the sender's point of view, security follows from the IND-CPA security of the encryption scheme.

13 OT from the Decisional Diffie-Hellman assumption

So now that we've seen that we can generically use 1-bit additively homomorphic encryption (A-HE) to make a 1-bit OT. If we now want to make OT from DDH assumption, it is natural to see if we can make 1-bit A-HE from DDH first. Recall that the DDH problem is defined as:

Definition 19 (Decisional Diffie-Hellman problem). For a given cyclic group G of order q and with generator g , distinguish between the triples (g^a, g^b, g^{ab}) and (g^a, g^b, g^c) , where a, b, c are randomly and independently chosen from $\mathbb{Z}_q$

The DDH problem is not hard for all groups. The existence of symmetric pairings makes the DDH problem easy for certain groups. Also, being able to solve the discrete log problem solves DDH.

13.1 1-bit A-HE from DDH

Recall the ElGamal public key encryption (PKE) scheme:

Definition 20 (ElGamal PKE). For a cyclic group G of order q with generator g , the ElGamal cryptosystem consists of three algorithms $(G_{\mathcal{E}\ell}, E_{\mathcal{E}\ell}, D_{\mathcal{E}\ell})$

The algorithms are defined as:

Algorithm 1 Generating Algorithm $G(1^\lambda; r) \rightarrow (pk, sk)$

- 1: $x \xleftarrow{R} \mathbb{Z}_q$
- 2: $h \leftarrow g^x$
- 3: $(pk, sk) \leftarrow (h, x)$
- 4: Output (pk, sk)

Algorithm 2 Encryption Algorithm $E(pk, m; r) \rightarrow c$

Ensure: $m \in G$

- 1: $y \xleftarrow{R} \mathbb{Z}_q$
- 2: $s \leftarrow h^y$
- 3: $c_1 \leftarrow g^y$
- 4: $c_2 \leftarrow m \cdot s$
- 5: $c \leftarrow (c_1, c_2)$
- 6: Output c

Informally, we can see that the key setup is just an instance of the discrete log problem and the ciphertext is of the form $(g^r, g^{(r \cdot x)} \cdot m)$. From this view, it is easy to see the correctness of the decryption algorithm.

Algorithm 3 Decryption Algorithm $D(sk, c) \rightarrow m$

- 1: $c_1, c_2 \leftarrow c$
 - 2: $s \leftarrow c_1^x$ $\triangleright c_1^x = g^{xy} = h^y$
 - 3: $m \leftarrow c \cdot s^{-1}$
 - 4: Output m
-

Note that the ElGamal scheme encrypts group elements of G , not binary bits. So to get a 1-bit A-HE from ElGamal, we need to find a way to map bits to the group elements and give a function Hom to homomorphically add two ciphertexts.

The most natural mapping of a bit b to a group element of G is $b \rightarrow g^b$. Using this mapping we can define a new scheme (G', E', D') which wraps around $G_{\mathcal{E}\ell}, E_{\mathcal{E}\ell}, D_{\mathcal{E}\ell}$. We can set $G' := G_{\mathcal{E}\ell}$, $E'(g^x, b; r) := E_{\mathcal{E}\ell}(g^x, g^b; r)$ and $D'(c_1, c_2) := D_{\mathcal{E}\ell}(c_1, c_2) \stackrel{?}{=} g$

Now, what could be a possible Hom function? Again, the most natural way seems to be to use coordinate wise multiplication of the ciphertexts. For two ciphertexts $(g^{r_1}, g^{r_1x+b_1})$ and $(g^{r_2}, g^{r_2x+b_2})$, we can call the Hom function to get:

$$Hom(g^x, (g^{r_1}, g^{r_1x+b_1}), (g^{r_2}, g^{r_2x+b_2})) = (g^{r_1+r_2}, g^{(r_1+r_2)x+b_1+b_2})$$

Which is a valid ciphertext for $b_1 + b_2$, but it is not what we want. Here, the addition is done over integers instead of xor between bits. This leads to a difference when we have both $b_1 = 1$ and $b_2 = 1$, since we want an encryption of g^0 , but what we actually get is an encryption of g^2 . This breaks our security definition for 1-bit A-HE as an adversary can easily differentiate between a fresh encryption of 0 and the output of the Hom function with $b_1 = 1, b_2 = 1$. Additionally, we can no longer get an encryption of $1 - b$ from an encryption of b , so our protocol wouldn't be able to run anyway.

As it turns out, there is no known way to turn the ElGamal scheme into a 1-bit A-HE. It remains an open problem to show such a transformation. However, we can still use this pseudo 1-bit A-HE to construct OT, if we change our syntax a little.

Instead of only sending $E(b)$, if the receiver instead sends over both $E(b)$ and $E(1 - b)$, the sender can compute $E(bm_1 + (1 - b)m_0)$ and we can continue with our protocol. How do we ensure that the receiver learns nothing about m_{1-b} ? This is ensured by noticing that the equation $bm_1 + (1 - b)m_0$ is upper bounded by 1 over the integers. Hence, we never run into the previous weird situation of getting $E(1 + 1)$ as an output of the Hom function.

As we will see next, there is also a much simpler way to get OT from DDH.

13.2 Simpler way to make OT from DDH

Consider an additional algorithm $G_L(1^\lambda; r) \rightarrow pk$ that takes in the security parameter and outputs a random public key, indistinguishable from a real public key. It is important to realize that the associated private key of pk is unknown and infeasible to compute for most properly instantiated cryptosystems. For the ElGamal cryptosystem, this can be done by randomly picking any group element of G .

Using this extra algorithm G_L , we can successfully carry out the OT protocol as follows:

1. Receiver chooses a bit b and computes the following messages to send:

$$(pk_0, pk_1) = \begin{cases} (pk_0, sk) \leftarrow G(1^\lambda; r), pk_1 \leftarrow G_L(1^\lambda; r) & \text{for } b=0 \\ pk_0 \leftarrow G_L(1^\lambda; r), (pk_1, sk) \leftarrow G(1^\lambda; r) & \text{for } b=1 \end{cases}$$

and keeps sk .

2. The sender receives (pk_0, pk_1) computes

$$c_0 \leftarrow E(pk_0, m_0; r)$$

$$c_1 \leftarrow E(pk_1, m_1; r)$$

and sends (c_0, c_1) to the receiver.

3. The receiver gets (c_0, c_1) and then runs $D(sk, c_b)$ to get message m_b .

It is clear that the receiver can only receive m_b , since they don't have the corresponding sk for pk_{1-b} . The security for the receiver is a direct consequence of the output of G_L being indistinguishable from a real public key. The security for the sender can be argued using the security of the PKE scheme.

This is how we get a simple OT from the DDH assumption using the ElGamal scheme and G_L .

14 CPA secure PKE from 2 round OT

Claim: OT is not a weaker assumption than CPA secure PKE. In particular, a generic 2-round OT can be used to build an IND-CPA secure PKE scheme.

Intuitively, we need to define the public key, encryption algorithm and decryption algorithm using the steps in OT to build a PKE. Informally, we can have the following:

1. Public Key: The first message sent by the receiver and the secret key is b
2. Encryption: The procedure the sender uses after receiving the first message. We can set $m_0 = m_1 = m$.
3. Decryption: The receiver's final procedure to recover m_b .

To formalize this construction, we first need to define the syntax of a generic 2-round OT. We say that a generic 2-round OT consists of three algorithms (OT_1, OT_2, OT_3) where

1. $OT_1(1^\lambda, b; r) \rightarrow OT_r$ is run by the receiver on bit b and randomness r to get message OT_r which it sends to the sender.
2. $OT_2(OT_r, (m_0, m_1)) \rightarrow OT_s$ is run by the sender on OT_r and messages (m_0, m_1) to get message OT_s which it sends to the receiver
3. $OT_3((b, r), OT_s) \rightarrow m_b$ which the receiver finally uses to get m_b

For security definitions, we need two indistinguishability guarantees:

1. $OT_r \leftarrow OT_1(1^\lambda, b)$ should be indistinguishable from $OT'_r \leftarrow OT_1(1^\lambda, 1 - b)$. This ensures that the sender learns nothing about the receiver's bit b .
2. We need that $OT_s \leftarrow (OT_r, m)$ is indistinguishable from $OT'_s \leftarrow (OT_r, m')$ where $m = (m_0, m_1)$ and

$$m' = \begin{cases} (m_0, 1 - m_1) & \text{for } b = 0 \\ (1 - m_0, m_1) & \text{for } b = 1 \end{cases}$$

This ensures that the receiver learns nothing about the other message the sender chose.

Based on this syntax, we can now formally define our CPA secure PKE. We have three algorithms G, E, D defined as:

1. $G(1^\lambda) \rightarrow (pk, sk) := OT_1(1^\lambda, b; r), (b, r)$ is the generating algorithm where $OT_r \leftarrow OT_1(1^\lambda, b; r)$ is the public key and (b, r) is the private key.
2. $E(pk, m) \rightarrow c := OT_2(OT_r, (m, m)) \rightarrow OT_s$ is the encryption algorithm with OT_s being the ciphertext.
3. $D(sk, c) \rightarrow m := OT_3((b, r), OT_s) \rightarrow m$ is the decryption algorithm.

To argue CPA security, we need to show that $(pk, E(pk, 0)) \approx (pk, E(pk, 1))$. Going back to our OT syntax, we need to show $(OT_r, OT_2(OT_r, (0, 0))) \approx (OT_r, OT_2(OT_r, (1, 1)))$. We can use the OT's security properties to show this. Consider the case where we have $b = 0$:

$$\begin{aligned} (OT_r, OT_2(OT_r, (0, 0))) &\approx (OT_r, OT_2(OT_r, (0, 1))) && \text{(Security for Sender)} \\ &\approx (OT'_r, OT_2(OT'_r, (0, 1))) && \text{(Security for Receiver, } OT'_r \text{ has } b = 1) \\ &\approx (OT'_r, OT_2(OT'_r, (1, 1))) && \text{(Security for Sender)} \\ &\approx (OT_r, OT_2(OT_r, (1, 1))) && \text{(Security for Receiver)} \end{aligned}$$

The proof process in an analogous way for the case where $b = 1$. This completes the proof. $\square$

15 Security of OT against outsiders

So far, we've considered security for both the sender and the receiver against each other. It is also worth considering the security of the protocol against outsiders. To help formalize this notion, we first define the notion of a transcript.

Definition 21 (Transcript of an OT). On an input b, m_0, m_1 , the transcript of an OT protocol are the messages $Trans(b, m_0, m_1) \rightarrow OT_r, OT_s$, where $OT_r \leftarrow OT_1(b; r)$ and $OT_s \leftarrow OT_2(OT_r, (m_0, m_1))$.

We can now define security against outsiders in terms of distinguishing between two distributions: $(b, m_0, m_1, b', m'_0, m'_1, Trans(b, m_0, m_1))$ and $(b, m_0, m_1, b', m'_0, m'_1, Trans(b', m'_0, m'_1))$. We say that an OT is secure against outsiders if those two distributions are indistinguishable.

It is important to note that if an OT is secure against outsiders, then MPC protocols built from that OT will not require private channels.

16 Question for later

In the previous lectures, we saw how to build an OT protocol if both parties access an asymmetric AND protocol. A related question worth thinking about: is it possible to build an OT protocol if both parties have access to a symmetric AND protocol?

17 Public-Key Operations vs Symmetric-Key Operations

First, we note the difference in efficiency for public-key operations vs symmetric-key operations. Symmetric-key operations are used in real-world constructions of private-key primitives. Examples of real-world constructions using symmetric-key operations include block ciphers such as DES and AES and collision-resistant hash functions such as MD5 and SHA. Such operations are fast, lightweight, and often hardware accelerated.

In contrast, public-key operations (such as performing modular arithmetic over large numbers, elliptic curve calculations, diffie-hellman group operations, etc.) are generally inefficient. However, they are more powerful, allowing us to build instantiations of public-key primitives such as public-key encryption, key-exchange, OT, etc. As of yet, it is an open problem whether public-key encryption can be built solely from private-key encryption without other assumptions. Due to the inefficiency of public-key operations, it is often a goal during protocol design to minimize the number of such operations.

17.1 Motivating Example

Consider a scenario where Bob wishes to securely transmit some 100GB message m to Alice. Alice has a public-key pk and a secret key sk . One simple solution would be for Bob to send $c = \text{Enc}_{\text{pub}}(pk, m)$ such that Alice can decrypt $m = \text{Dec}_{\text{pub}}(sk, c)$. Due to the inefficient public-key operations on a large message, this may take a very long time (half an hour, estimates Prof. Hajiabadi).

Instead, use **Hybrid Encryption**. Bob samples some symmetric key k , encrypts $c_1 = \text{Enc}_{\text{pub}}(pk, k)$, $c_2 = \text{Enc}_{\text{priv}}(k, m)$, and sends both ciphertexts to Alice. Alice can then retrieve $k = \text{Dec}_{\text{pub}}(sk, c_1)$ and decrypt $m = \text{Dec}_{\text{priv}}(k, c_2)$. Since $|k| \ll |m|$, the slow public-key operations are performed on small data while the fast symmetric-key operations are performed on large data. This takes less than 7 seconds as estimated by Prof. Hajiabadi.

18 OT Extensions

Recall that OT is a strong enough primitive to build public key encryption with, as discussed in 14. Since PKE can be built from OT, and PKE requires slow public-key operations, it therefore follows that OT also requires slow public-key operations. Previously, in the discussion of the GMW protocol in 9.3, each evaluation of an AND gate in a circuit required Alice and Bob to perform an OT execution. Given this, we would like to minimize the number of OT executions when evaluating a protocol. While GMW is very sequential — each OT execution may depend on the evaluation of prior OT executions — we will initially focus on a simpler form of the problem: **Oblivious Transfer Extensions**. In OT extensions, we wish to evaluate many OT operations (say, 10^7) while using an OT oracle a minimal number of times (say, 1000).

Formally, assume Alice has choice bits $b_1, \dots, b_l$. Bob has message pairs $(m_0^{(1)}, m_1^{(1)}), (m_0^{(2)}, m_1^{(2)}), \dots, (m_0^{(l)}, m_1^{(l)})$. Alice should learn $m_{b_i}^{(i)}$ for $1 \leq i \leq l$. Alice should learn nothing about $m_{1-b_i}^{(i)}$, and Bob should learn nothing whatsoever about Alice's choice bits b_i . Both Alice and Bob have access to an OT oracle and a random oracle (RO).

Definition 22 (Random Oracle). A Random Oracle behaves as a random function. When called on a particular input, it produces a random value from its output domain. Outputs on the same input are always the same, outputs on different inputs are always independent.

This task can clearly be completed by simply calling the OT oracle l times. To make this interesting, we impose the restriction that the number of calls to the OT oracle is $\text{Poly}(\lambda)$, where λ is a security parameter chosen independent of l . The number of RO calls is unlimited.

18.1 A Simplified Warmup

Consider the case where all of Alice's choice bits are the same and Bob is aware of this fact. In addition, assume the OT oracle only accepts single-bit messages. We can solve the OT-extension problem via symmetric-key encryption. Let $M_0 = m_0^{(1)} \parallel m_0^{(2)} \parallel \dots \parallel m_0^{(l)}$, let $M_1 = m_1^{(1)} \parallel m_1^{(2)} \parallel \dots \parallel m_1^{(l)}$. Then, the following protocol solves this limited form of the OT-extension problem:

1. Bob generates two keys of length λ : $k_0, k_1 \leftarrow \text{KGen}_{\text{priv}}(1^\lambda)$
2. Bob sends $c_0 = \text{Enc}_{\text{priv}}(k_0, M_0)$, $c_1 = \text{Enc}_{\text{priv}}(k_1, M_1)$ to Alice over a public channel.
3. Alice (receiver) and Bob (sender) then use the OT oracle λ times. In the i -th OT oracle call, Bob supplies the i -th bit of k_0 and the i -th bit of k_1 , whereas Alice always supplies her singular choice bit b . After these λ calls, Alice has learned k_b
4. Alice uses the key k_b to decrypt $M_b = \text{Dec}_{\text{priv}}(k_b, c_b)$

Note that since the key is only used to encrypt once, we do not even need to use a randomized encryption scheme. A simple, deterministic construction which will satisfy our constraints is the Pseudo-OTP scheme.

Definition 23 (Pseudo-OTP). The Pseudo-OTP encryption scheme given a Pseudorandom Generator (PRG) $G : \{0, 1\}^\lambda \rightarrow \{0, 1\}^l$ defines the following algorithms.

1. $\text{KGen}(1^\lambda)$: Uniformly sample a bitstring from $\{0, 1\}^\lambda$ and output it.
2. $\text{Enc}(k, m \in \{0, 1\}^l) = G(k) \oplus m$
3. $\text{Dec}(k, c) = G(k) \oplus c$

G can be instantiated with constructions such as AES/DES, which makes for an incredibly fast symmetric key encryption scheme. If we relax the assumption that the OT oracle takes single-bit messages and instead assume the OT oracles takes λ -bit messages, then the previous protocol can be solved with just one OT oracle call. This is not an unreasonable relaxation: in 13.2, we compute ciphertexts via encrypting group elements - we can map bitstrings of length λ into group elements with a λ -bit representation.

18.2 History of OT Extensions

OT Extensions was introduced in Beaver [1996]. In this paper, Beaver uses garbled circuits and pseudorandom generators to solve the problem. But, this requires a non-black box construction of a PRG, which makes this more of a feasibility result than something practical. In Ishai et al. [2003], the authors designed an OT extension protocol via a black-box use of a RO, which requires two round of communication: this is the protocol we now study.

18.3 High-Level Overview

The notation OT_l^m denotes the task of performing m oblivious transfers of l -bit strings. Our goal is to solve this problem generically using k calls to the OT oracle, where k is the security parameter. Other than these k calls, we wish to only use cheap symmetric-key operations. The OT oracle will be assumed to be work over m -bit strings; although m does not depend on the security parameter, we can efficiently translate an OT using k -bit strings to an OT using m -bit strings via the PRG trick. Rather than Bob invoking the oracle with the pair of m -bit strings, two random k -bit strings s_0 and s_1 are generated, which are used in the OT. Bob sends $m_0 \oplus G(s_0)$ and $m_1 \oplus G(s_1)$ to Alice over a public channel, and Alice can recover m_b via $G(s_b)$. See [Ishai et al., 2003, Appendix B] for more details.

The OT extension protocol will proceed in two phases: first, we will perform “OT correlations” which will use all of our OT oracle queries. Second, we will perform a simple cheap masking phase to complete the protocol. Since m now denotes the *the number of oblivious transfers*, we now denote Bob’s *messages* with the symbol f . I.e, Bob has $(f_0^{(1)}, f_1^{(1)}), \dots, (f_0^{(m)}, f_1^{(m)})$, and Alice has $b_1, \dots, b_m$. Alice should learn $f_{b_i}^{(i)}$ for $1 \leq i \leq m$, while learning nothing about $f_{1-b_i}^{(i)}$. Bob, as usual, should learn nothing whatsoever.

18.4 Phase 1: OT Correlations

Establishing OT correlations requires Bob to learn $(r_0^{(1)}, r_1^{(1)}), \dots, (r_0^{(m)}, r_1^{(m)})$, where each $r_0^{(i)}$ and $r_1^{(i)}$ are l -bit strings. Alice uses the same choice bits $b_1, \dots, b_m$ from the larger OT extension protocol, and should learn $r_{b_i}^{(i)}$ for $1 \leq i \leq m$. In addition, the following security parameters should hold:

1. Bob should not learn anything about Alice’s choice bits.
2. For Alice, $r_{1-b_i}^{(i)}$ (the strings she *isn’t* learning) should be indistinguishable from random l -bit strings.
3. For an outsider, all strings $r_0^{(i)}$ and $r_1^{(i)}$ should be indistinguishable from random l -bit strings.

Protocol:

Setup: First, Alice generates a matrix T with m rows and k columns where each entry is a random bit. We denote the *rows*, which are k -bit vectors as $t_1, \dots, t_m$. We denote the *columns*, which are m -bit vectors $t^{(1)}, \dots, t^{(m)}$. At the same time, Bob generates a row vector s containing k random bits.

$$T = \begin{bmatrix} - & t_1 & - \\ - & t_2 & - \\ - & t_3 & - \\ & \vdots & \\ - & t_m & - \end{bmatrix} = \begin{bmatrix} | & | & | & | \\ t^{(1)} & t^{(2)} & \dots & t^{(k)} \\ | & | & | & | \\ \vdots & \vdots & \ddots & \vdots \\ | & | & | & | \\ t^{(1)} & t^{(2)} & \dots & t^{(k)} \\ | & | & | & | \end{bmatrix}$$

We then perform k calls to the OT oracle where Alice is the sender, and Bob is the receiver. In the j -th call, Bob sends the bit s_j from his vector s , and Alice sends the pair of messages $(t^{(j)}, t^{(j)} \oplus b)$ (which are m -bit vectors). Denote the vector Bob receives as $q^{(j)}$ for $1 \leq j \leq k$. With k vectors containing m bits, Bob can create a matrix Q with m rows and k columns:

$$Q = \begin{bmatrix} | & | & | & | \\ q^{(1)} & q^{(2)} & \dots & q^{(k)} \\ | & | & | & | \\ \vdots & \vdots & \ddots & \vdots \\ | & | & | & | \\ q^{(1)} & q^{(2)} & \dots & q^{(k)} \\ | & | & | & | \end{bmatrix} = \begin{bmatrix} - & q_1 & - \\ - & q_2 & - \\ - & q_3 & - \\ & \vdots & \\ - & q_m & - \end{bmatrix}$$

Claim 18.1. $q_i = \begin{cases} t_i & b_i = 0 \\ t_i \oplus s & b_i = 1 \end{cases}$

Proof. The proof is by cases. Let $q_i[j]$ denote the j -th element of q_i , $1 \leq i \leq m$, $1 \leq j \leq k$. Let $b_i = 0$. Then, by identity, $q_i[j] = q^{(j)}[i]$. Due to the OT, if $s_j = 0$, then $q^{(j)}[i] = t^{(j)}[i]$, otherwise, $q^{(j)}[i] = t^{(j)}[i] \oplus b_i$. But, since $b_i = 0$, in either case, $q^{(j)}[i] = t^{(j)}[i] = t_i[j]$. We have shown that $q_i[j] = t_i[j]$ and hence, when $b_i = 0$, $q_i = t_i$.

Now, consider the case where $b_i = 1$. As before, $q_i[j] = q^{(j)}[i]$ by identity. If $s_j = 0$, then the OT selects the first message: $q^{(j)}[i] = t^{(j)}[i] = t_i[j] = t_i[j] \oplus s_j$. If $s_j = 1$, then the OT selects the second message: $q^{(j)}[i] = t^{(j)}[i] \oplus b_i = t_i[j] \oplus b_i = t_i[j] \oplus s_j$. In either case, $q_i[j] = t_i[j] \oplus s_j$, and hence when $b_i = 1$, $q_i = t_i \oplus s$. $\square$

OT Correlations: Now, Alice has $(t_1, \dots, t_m)$ for $1 \leq i \leq m$ and choice bits $b_1, \dots, b_m$. Bob has $(q_1, \dots, q_m)$ and s such that $q_i = b_i(t_i \oplus s) + (1 - b_i)t_i$. To complete OT correlations, Bob uses the random oracle and outputs $(r_0^{(i)}, r_1^{(i)}) = (RO(q_i), RO(q_i \oplus s))$.

Correctness: If $b_i = 0$, then $q_i = t_i$, so Bob outputs $(r_0^{(i)}, r_1^{(i)}) = (RO(t_i), RO(t_i \oplus s))$. Alice can also compute $RO(t_i)$, and so she can learn $r_0^{(i)} = r_{b_i}^{(i)}$.

If $b_i = 1$, then $q_i = t_i \oplus s$, so Bob outputs $(r_0^{(i)}, r_1^{(i)}) = (RO(t_i \oplus s), RO(t_i \oplus s \oplus s)) = (RO(t_i \oplus s), RO(t_i))$. Once again, Alice can compute $RO(t_i)$ and so she can learn $r_1^{(i)} = r_{b_i}^{(i)}$.

18.5 Phase 2: OT Extensions from OT Correlations

We have just shown how to perform OT correlations. Once Alice and Bob have set up their OT correlations, they can easily perform an OT extension. Recall that Bob has $(f_0^{(1)}, f_1^{(1)}), \dots, (f_0^{(m)}, f_1^{(m)})$, and Alice has $b_1, \dots, b_m$. In addition, having completed OT correlations, Bob has $(r_0^{(1)}, r_1^{(1)}), \dots, (r_0^{(m)}, r_1^{(m)})$, and Alice has $r_{b_1}^{(1)}, \dots, r_{b_m}^{(m)}$.

In this cheap masking phase, Bob sends Alice $\{(f_0^{(i)} \oplus r_0^{(i)}, f_1^{(i)} \oplus r_1^{(i)}) : 1 \leq i \leq m\}$. Since Alice has knowledge of $r_{b_i}^{(i)}$, she can compute $f_{b_i}^{(i)} \oplus r_{b_i}^{(i)} \oplus r_{b_i}^{(i)}$ to recover $f_{b_i}^{(i)}$. And, because Alice has no knowledge of $r_{1-b_i}^{(i)}$, Bob's other string, $f_{1-b_i}^{(i)}$ is indistinguishable from random since it has been masked by this unknown string $r_{1-b_i}^{(i)}$. In total, Alice has learned $f_{b_i}^{(i)}$ for $1 \leq i \leq m$, Alice has learned nothing of $f_{1-b_i}^{(i)}$, and Bob has learned nothing whatsoever. Hence, the OT extension protocol has been completed, using just k OT oracle calls during the OT correlations phase (which recall, k is the security parameter, and is assumed to be much smaller than m).

18.6 Purpose of the Random Oracle in OT Correlations

To understand the role of the Random Oracle in OT correlations, we can imagine the behaviour if Bob simply outputted $(r_0^{(i)}, r_1^{(i)}) = (q_i, q_i \oplus s)$. This would satisfy the correctness property of OT correlations: the value of $r_{b_i}^{(i)}$ is always t_i , which Alice knows. However, this doesn't satisfy the security property. As an example, consider if all of Alice's choice bits are zero, $b_i = 0$ for all $1 \leq i \leq m$. Then, we note that for all i , $r_1^{(i)} = t_i \oplus s$. Alice can then calculate $r_1^{(i)} \oplus t_i = t_i \oplus t_i \oplus s = s$ for all i . To satisfy the security property, all of the $r_1^{(i)}$ strings should be indistinguishable from random. Alice can easily distinguish between random strings and $r_1^{(i)}$ by XOR-ing each $r_1^{(i)}$ with t_i - since this computation yields s for all i , these strings clearly aren't randomly distributed.

18.7 Adaptive OT Extensions

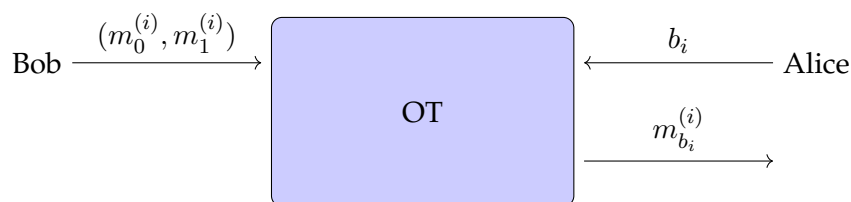
If Alice knows all ℓ choice bits ahead of time and Bob knows all ℓ pairs of messages ahead of time, they can use OT correlations to perform OT extensions. What if Alice wants to make adaptive choices based on the message pairs she receives in previous rounds?

Claim: We can perform sequential OT for very low additional cost.

For round $i \in \{1, \dots, \ell\}$,

1. Alice supplies bit b_i
2. Bob supplies $(m_0^{(i)}, m_1^{(i)})$
3. Alice should learn $m_{b_i}^{(i)}$

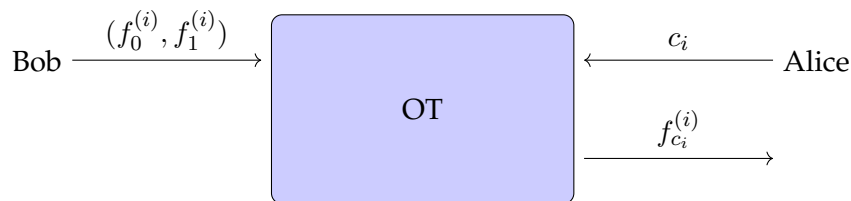
Visually:



We accomplish this in two phases: an offline phase and an online phase.

Offline Phase In the offline phase, Alice supplies ℓ random bits, and Bob supplies ℓ random pairs of messages. We use OT extension in the offline phase; Alice and Bob call OT k times where k is the security parameter.

For $i \in \{1, \dots, \ell\}$,



Here, all c_i , $f_0^{(i)}$, and $f_1^{(i)}$ are random. As these values are independent, these k OT calls can be made in parallel, and this offline phase requires only 2 rounds.

Online Phase In the offline phase, Alice made some arbitrary “guesses” about which value she wants in each round of the online phase. Now in the online phase, she can either stick with her original choice or FLIP her original choice bit to select the other message. Recall that $\forall i \in \{1, \dots, \ell\}$,

Bob knows $(f_0^{(i)}, f_1^{(i)})$ and $(m_0^{(i)}, m_1^{(i)})$, and

Alice knows $(c_i, f_{c_i}^{(i)})$.

For $i \in \{1, \dots, \ell\}$,

1. Alice sets $\text{FLIP}_i = b_i \oplus c_i$
2. Alice sends FLIP_i to Bob
3. Bob sends $(m_0^{(i)} \oplus f_0'^{(i)}, m_1^{(i)} \oplus f_1'^{(i)})$ to Alice, where

$$(f_0'^{(i)}, f_1'^{(i)}) = \begin{cases} (f_0^{(i)}, f_1^{(i)}) & \text{FLIP}_i = 0 \\ (f_1^{(i)}, f_0^{(i)}) & \text{FLIP}_i = 1 \end{cases}$$

Observe:

c_i	$f_{c_i}^{(i)}$	b_i	$\text{FLIP}_i = c_i \oplus b_i$	$f_0'^{(i)}$	$f_1'^{(i)}$	$(m_0^{(i)} \oplus f_0'^{(i)}, m_1^{(i)} \oplus f_1'^{(i)})$	Alice learns
0	$f_0^{(i)}$	0	0	$f_0^{(i)}$	$f_1^{(i)}$	$(m_0^{(i)} \oplus f_0^{(i)}, m_1^{(i)} \oplus f_1^{(i)})$	$m_0^{(i)}$
0	$f_0^{(i)}$	1	1	$f_1^{(i)}$	$f_0^{(i)}$	$(m_0^{(i)} \oplus f_1^{(i)}, m_1^{(i)} \oplus f_0^{(i)})$	$m_1^{(i)}$
1	$f_1^{(i)}$	0	1	$f_1^{(i)}$	$f_0^{(i)}$	$(m_0^{(i)} \oplus f_1^{(i)}, m_1^{(i)} \oplus f_0^{(i)})$	$m_0^{(i)}$
1	$f_1^{(i)}$	1	0	$f_0^{(i)}$	$f_1^{(i)}$	$(m_0^{(i)} \oplus f_0^{(i)}, m_1^{(i)} \oplus f_1^{(i)})$	$m_1^{(i)}$

Correctness: In all cases, Alice learns $m_{b_i}^{(i)}$.

Security for Alice: Bob does not learn c_i during the offline phase because of the security of OT. Bob does not learn b_i during the online phase, only $b_i \oplus c_i$, which, because he does not know c_i , leaks no information.

Security for Bob: Alice receives $m_{1-b_i}^{(i)} \oplus f_{1-c_i}^{(i)}$, but she never learns $f_{1-c_i}^{(i)}$, so she cannot determine $m_{1-b_i}^{(i)}$.

Observation: It is not safe to reuse correlations. Suppose Alice reused $c_i = c_j$ for some $i \neq j$. Then, Bob can determine whether $b_i = b_j$:

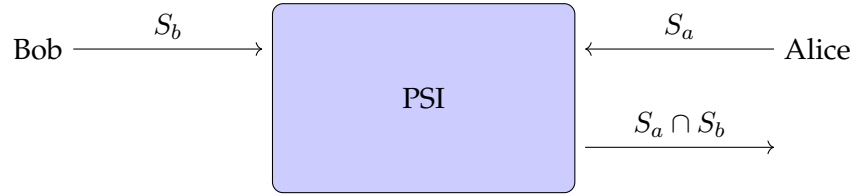
$$b_i = b_j \iff \text{FLIP}_i = \text{FLIP}_j.$$

19 Private Set Intersection (PSI)

Definition 24 (PSI). We define the protocol for **private set intersection (PSI)**:

1. Alice supplies $S_a \subset \{0, 1\}^k$
2. Bob supplies $S_b \subset \{0, 1\}^k$
3. Alice learns $S_a \cap S_b$
4. Bob learns nothing about S_a

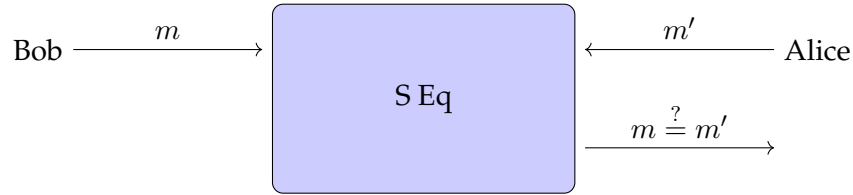
Visually:



Security for Alice: $\forall S_a, S'_a \subset \{0, 1\}^k$ s.t. $|S_a| = |S'_a|$, Bob cannot tell if Alice has S_a or S'_a .

Security for Bob: $\forall S_a, S_b, S'_b \subset \{0, 1\}^k$ s.t. $|S_b| = |S'_b| \wedge S_a \cap S_b = S_a \cap S'_b$, Alice cannot tell if Bob has S_b or S'_b .

This problem can be reduced to string equality. For $m, m' \in \{0, 1\}^k$,



A simple approach is to represent each element in S_a and S_b as a string, then compare all pairwise combinations of $m_a \in S_a$ and $m_b \in S_b$.

19.1 Building OT from PSI

If we can solve PSI, we can solve OT.

Suppose we have a PSI protocol for sets containing 2-bit strings: $S_a \subseteq \{0, 1\}^2, S_b \subseteq \{0, 1\}^2$.

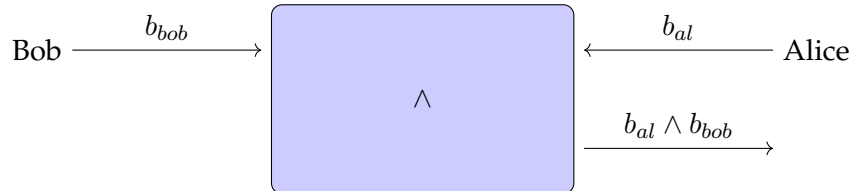


Given Bob has (m_0, m_1) and Alice has b , where $m_0, m_1, b \in \{0, 1\}$. Alice should learn m_b , and Bob should learn nothing. Set $S_b = \{0||m_0, 1||m_1\}$ and $S_a = \{b||0, b||1\}$. Then, $S_a \cap S_b = \{b||m_b\}$.

19.2 Designing PSI

19.2.1 Building PSI from Asymmetric AND

Suppose Alice and Bob have sets $S_a, S_b \subseteq \{1, \dots, n^c\}$ for some constant c . Alice should learn $S_a \cap S_b$ in 3 rounds. Alice and Bob make binary strings of length n^c such that each element is represented by a single bit (1 if the element is in the set, 0 if it is not). For each bit, they perform an asymmetric $\wedge$ such that Alice learns $b_{al} \wedge b_{bob}$.



Because this protocol is non-adaptive, all of these $\wedge$ operations can be performed in parallel.

Note: Alice can set her string to all 1 bits to learn Bob's entire set.

19.2.2 Designing PSI (First Attempt)

Suppose Alice has $S_a = \{x_1, \dots, x_m\}$ and Bob has $S_b = \{y_1, \dots, y_n\}$. (It is not necessary that Alice and Bob have the same number of elements.) Suppose that H is a hash function that outputs an element from the group G . Consider the following protocol for PSI:

1. Bob computes $H(y_1), \dots, H(y_n)$ and sends these values to Alice.
2. Alice computes $H(x_1), \dots, H(x_m)$ and compares these values with the ones sent by Bob.

Completeness: Alice learns $S_a \cap S_b$.

Security for Alice: Bob learns nothing, so $\forall S_a, S'_a \subset \{0, 1\}^k$ s.t. $|S_a| = |S'_a|$, Bob cannot tell if Alice has S_a or S'_a .

We do **not** have security for Bob. Because Bob sends Alice the hashes of his elements, Alice can compute $S_b \cap S'$ for any set S' of her choice. Recall that single-round protocols cannot be secure.

19.2.3 Designing PSI (Fixed)

1. Alice samples a at random, computes $H(x_1)^a, \dots, H(x_m)^a$, and sends all $H(x_i)^a$ to Bob.
2. Bob samples b at random, computes $H(y_1)^b, \dots, H(y_n)^b$ as well as $(H(x_1)^a)^b, \dots, (H(x_m)^a)^b$, and sends all $H(y_i)^b$ and $(H(x_j)^a)^b$ to Alice. Importantly, Bob must preserve the order of each $(H(x_j)^a)^b$.
3. Alice computes each $(H(y_i)^b)^a$ and compares them to the ordered $(H(x_j)^a)^b$ sent by Bob.

Completeness: Alice learns $S_a \cap S_b$.

Claim 19.1. Let g be a generator of G . Let $g_1, \dots, g_n \xleftarrow{\$} G$. (Note that because g is a generator, $\forall i, \exists a$ s.t. $g_i = g^a$.) Let r be selected randomly. Let $g'_1, \dots, g'_{n+1} \xleftarrow{\$} G$. Then,

$$\begin{bmatrix} (g, g_1, \dots, g_n) \\ (g^r, g_1^r, \dots, g_n^r) \end{bmatrix} \cong^C \begin{bmatrix} (g, g_1, \dots, g_n) \\ (g'_1, g'_2, \dots, g'_{n+1}) \end{bmatrix}$$

Proof. Let r' be the discrete logarithm of g'_1 s.t. $g^{r'} = g'_1$. For g_i , let r_{a_i} be the discrete logarithm of g_i s.t. $g^{r_{a_i}} = g_i$. Suppose $\exists$ a PPT adversary A s.t. A can distinguish between these two distributions. Then $\exists i$ s.t. A can distinguish between $(g, g^{r'}, g^{a_i}, g'_{i+1})$ and $(g, g^{r'}, g^{a_i}, g^{r' a_i})$. This violates the decisional Diffie-Hellman assumption. Thus, these distributions must be computationally indistinguishable. $\square$

I think this argument is valid since the elements are all chosen independently, but would someone please check my work?

Security for Alice: By Claim 19.1, Bob cannot distinguish between Alice's set of $\{H(x_i)^a\}$ and a set of random elements. Thus, for two sets S_a, S'_a s.t. $|S_a| = |S'_a|$, Bob cannot tell if Alice has S_a or S'_a .

Security for Bob: By Claim 19.1, for the elements Bob's set does not share with Alice's, Alice cannot distinguish between the hashes of these elements raised to the power of b : $\{H(y_i)^b\} - \{H(y_i)^b | y_i \in S_a\}$ and a set of random elements of the same size. Thus, $\forall S_a, S_b, S'_b \subset \{0, 1\}^k$ s.t. $|S_b| = |S'_b| \wedge S_a \cap S_b = S_a \cap S'_b$, Alice cannot tell if Bob has S_b or S'_b .

20 Oblivious Pseudorandom Functions (OPRFs)

Definition 25 (OPRF). Given a PRF: $\{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}^n$, which may be called PRF_k or F_k , we define the protocol for an **oblivious pseudorandom function (OPRF)**:

1. Alice supplies $x_1, \dots, x_n$
2. Bob supplies k
3. Alice learns $\text{PRF}_k(x_1), \dots, \text{PRF}_k(x_n)$ but nothing else about k
4. Bob learns nothing about $x_1, \dots, x_n$

Visually:



20.1 PSI from OPRFs

Given an OPRF, we can construct PSI:

1. Alice supplies the elements of her set S_a : $x_1, \dots, x_n$
2. Bob supplies k
3. Alice learns $F_k(x_1), \dots, F_k(x_n)$
4. Bob separately sends $F_k(y_1), \dots, F_k(y_m)$ where $S_b = \{y_1, \dots, y_m\}$
5. Alice computes the intersection

Correctness: As long as the probability of collisions in the output of the PRF is negligible, we have correctness.

Security: For $Y = \{y_1, \dots, y_n\}, Y' = \{y'_1, \dots, y'_n\}$ where $X \cap Y = X \cap Y', \forall i$ s.t. $y_i \notin X \cap Y, F_k(y_i)$ is indistinguishable from a truly random value.

20.2 Building OPRFs from DH

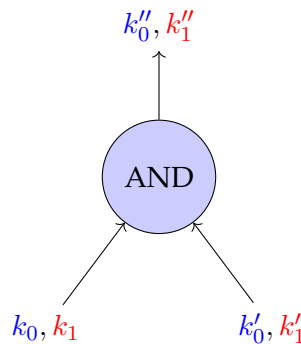
We can build an OPRF from Diffie-Hellman. Let $H : \{0, 1\}^* \rightarrow G$ be a “random oracle”. Define $F_k(x) = H(x)^k$. This is a PRF because of the DDH assumption.

21 2-round MPC using Garbling Circuit

The key technique in constructing a 2-round MPC is garbled circuits. First, we will go over the construction for a single gate in a circuit specifically focusing on AND and XOR gates. Once we establish how to garble a single gate, we can extend the approach to entire circuits.

21.1 Garbled Table for AND and XOR gate

Suppose Alice holds a bit b (left wire) and Bob holds a bit b' (right wire). The goal is for Bob to learn the product of the two bits, i.e., the output of an AND gate applied to b and b' . To achieve this securely, we will use oblivious transfer (OT) along with the concept of garbled circuits.



Alice can convey this information in a way that ensures Bob learns only one key (one of k''_0 or k''_1). She can achieve this using garbled table, which allow the keys to be connected in a manner consistent with the underlying circuit. During the protocol, Alice will:

1. Generate random keys for each wire.
2. Connect the keys together via a garbled table. The invariant is that if Bob has k_0 and k'_1 , he must be able to obtain k''_0 because $0 \wedge 1 = 0$. At this step, Bob doesn't have any key yet as this is only a sequence of encryptions.
3. Send the garbled table along with k_b and (k''_0, k''_1) to Bob.
4. Engage in a (1,2) OT, allowing Bob to learn $k'[b]$.

Garbled Table for AND gate

$E(k_0, E(k'_0, k''_0))$
$E(k_0, E(k'_1, k''_0))$
$E(k_1, E(k'_0, k''_0))$
$E(k_1, E(k'_1, k''_1))$

However, this approach is not secure. If Bob can successfully decrypt the first cell, he immediately learns that Alice's input was 0. To prevent this, we must shuffle the garbled table before sending it to Bob. Shuffling ensures that Bob cannot infer Alice's input simply by checking which row he can decrypt. The shuffled version will work if the symmetric key encryption scheme has the property that decrypting with an incorrect key results in an error symbol. However, designing such a private encryption scheme is not easy. While it is theoretically possible, it is practically inefficient.

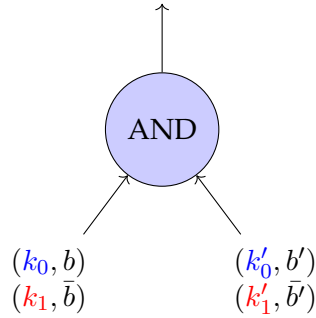
A simple way to construct a private encryption scheme is by using a pseudorandom function (PRF). Suppose we want to encrypt a message m under some randomness r with a key k . Let $F : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}^n$ be a pseudorandom function that maps n bits of a key, n bits of an input, and n bits of an output. We can define a PRF-based encryption scheme as follows:

$$E_k(m; r) = (r, F(k, r) \oplus m) = \{0, 1\}^n \times \{0, 1\}^n$$

The property of this private key encryption scheme is that if we decrypt any ciphertext with any key (even if the key was not used to generate the ciphertext), the decryption will still produce an output that make sense. This is because the decryption process always outputs an n -bit value and never returns an error. Therefore, designing an encryption scheme where decryption with an incorrect key results in an error symbol is not easy and is not trivial.

$$D_k(c_1, c_2) = c_2 \oplus F_k(c_1)$$

Now, we want to change the garbled table so that when Bob obtains two keys, he will immediately know which cell in the table to look at. The idea is that we have a key for each wire. We can add a color to each key and an additional signal bit. This idea is called permute and point. We shuffle the table in such a way that when the other party learns two keys, those keys will point to a single cell in the table. We append b and $\bar{b}$ for a random b to each $k0_i$ and $k1_i$.



The four ciphertexts are then arranged in the following order: $(0, 0)$, $(0, 1)$, $(1, 0)$, $(1, 1)$, essentially sorting them according to the last bit because this bit is chosen uniformly at random and it is generated independently of the underlying actual part. When Bob possesses the two keys, $(k[c], b)$ and $(k'[c'], b')$ where c represents Alice's input and c' represents Bob's original input, the values b and b' will index a specific cell in the table. Bob will then pick that cell and decrypt it. As a result, evaluating the garbled circuit becomes more efficient, requiring only two decryptions instead of eight.

For XOR gate, the idea will be similar. The unpermuted garble table version will be $(0, 0) \rightarrow 0$, $(0, 1) \rightarrow 1$, $(1, 0) \rightarrow 1$, $(1, 1) \rightarrow 0$. After we have the unpermuted version, we need to permute it and add the color bits to every key so we know which cell in the table to look at.

21.2 Garbled Table for Large Circuit

Suppose we have a large circuit C where $C : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}$. This circuit takes n bits from Alice and n bits from Bob and outputs a single bit. C consists of many AND and XOR gates. Suppose Alice has input x and Bob has input y , at the end, Bob will learn $C(x, y)$. Alice will generate a garbled circuit for the entire circuit by following these steps:

Alice is going to:

1. For each wire i in the circuit, Alice generates two keys for each wire: $k_0^{(i)}, k_1^{(i)}$.
2. For each gate G in the circuit, Alice generates a garbled table $GT(G) = \tilde{G}$.
3. The input wires are labeled as $1, \dots, 2n$ and the output wire is labeled as O .
4. Alice sends the sequence of all garbled tables $(\tilde{G}_1, \dots, \tilde{G}_m)$, where m is a number of gates in the circuit and $k_0^{(O)}, k_1^{(O)}$ to Bob.
5. Alice will also include $(k_{x_1}^{(1)}, \dots, k_{x_n}^{(n)})$. Now, Bob will learn the other part $(k_{y_1}^{(n+1)}, \dots, k_{y_n}^{(2n)})$ through OT.

At this point, Bob has all the input labels and the full sequence of garbled tables. So, if Bob sees the very first gate, Bob will decrypt the corresponding cell and Bob will learn one of these two keys. Then, one of these two keys will be mapped to the next garbled table for the two incoming wires. This protocol runs in two rounds. Bob and Alice will engage in n OTs, but these OTs execution can be done in parallel.

21.3 Garbled Circuit Definition

Definition 26 (Garbled Circuit). Suppose that we have a circuit $C : \{0, 1\}^m \rightarrow \{0, 1\}$. A **Garbled circuit** consists of two algorithms: $(\text{Garb}, \text{Eval})$ where;

1. $\text{Garb}(\lambda, C) \rightarrow (\tilde{C}, \{lb_{i,b}\}_{i \in [m], b \in \{0,1\}})$, where λ is a security parameter, l is a label, and $[m] = \{1, \dots, m\}$
2. $\text{Eval}(\tilde{C}, lb_1, x_1, \dots, lb_m, x_m) \rightarrow b' \in \{0, 1\}$

Correctness: $\forall x \in \{0, 1\}^m$; and for the variables as above:

$$\text{Eval}(\tilde{C}, lb_{1,x_1}, \dots, lb_{m,x_m}) = C(x)$$

Security: $\exists \text{Sim}_{PTT}$ such that for all x :

$$(x, \tilde{C}, \{lb_{i,x_i}\}_{i \in [n]}) \stackrel{C}{\cong} (x, \text{Sim}(C, C(x)))$$

The intuition of this security definition is that the garbled circuit and the sequence of garbled labels do not reveal any information beyond what can be simulated given only the circuit's output on the corresponding input.

(In-)Security example: Suppose we have a garbled circuit $\tilde{C} = C$ and define $lb_{i,b} = (i, b)$. The evaluation algorithm should be $\text{Eval}(\tilde{C}, (1, x_1), \dots, (m, x_m)) \rightarrow (C, (x_1, \dots, x_m))$. We have correctness but not security. This is because the garbled table will leak the input x entirely. We need

to simulate it in a way that $(x, C, (1, x), \dots, (m, x_m)) \stackrel{C}{\cong} (x, C, \text{Sim}(C, C(x)))$, where Sim will only have the output of the circuit. The only information the other party should have is C and $C(x)$, where $C(x)$ is a single bit. This shows that this garbled circuit is not secure.

Now, we argue that if we have a garbled circuit for a circuit C , we can combine it in a generic way with OT to build a secure two-party computation protocol for C . Let's say that the underlying circuit $C(x, y)$ maps $2n$ bits to 1 bit, where Alice has x and Bob has y . Now, Alice is going to garble $C : \text{Garb}(C) \rightarrow (\tilde{C}, \{lb_{i,b}\}_{i \in [2n], b \in \{0,1\}})$. Alice will send $(\tilde{C}, (lb_{1,x_1}, \dots, lb_{n,x_n}, \square, \dots, \square))$ to Bob. For the rest ($\square$), Bob will learn one of the two input labels by OT. Bob now has the garbled circuit and one input label for each input wire. According to the correctness definition, if Bob runs Eval , which is performed on Bob's side, he will learn the output and nothing more.

21.4 Building a Garbled Circuit without Cryptography

We have shown that using a symmetric key encryption scheme, we can build a garbled circuit for any circuit. However, it turns out that if the circuits are small, in other words, if they have log depth, we do not need to use cryptography to build a garbled circuit.

Suppose we have circuit $C = \{0, 1\}^n \rightarrow \{0, 1\}$, where $C(x) = i \oplus x_i$. We have a matrix that denotes the two input labels for each input wire, where $\tilde{C} = \emptyset$ and $lb_{i,b}$ will have 1 bit.

$$\begin{bmatrix} lb_{1,0} & \dots & lb_{n,0} \\ lb_{1,1} & \dots & lb_{n,1} \end{bmatrix} = \begin{bmatrix} 0 & 0 & \dots & 0 \\ 1 & 1 & \dots & 1 \end{bmatrix}$$

Correctness: $\text{Eval}(lb_{1,x_1}, \dots, lb_{n,x_n}) = x_1 \oplus \dots \oplus x_n$

We have correctness but not security. What we can do is adding $r_1, \dots, r_{n-1}, r_n$, where these values are generated uniformly at random and $r_n = r_1 \oplus \dots \oplus r_{n-1}$:

$$\begin{bmatrix} 0 + r_1 & 0 + r_2 & \dots & 0 + r_n \\ 1 + r_1 & 1 + r_2 & \dots & 1 + r_n \end{bmatrix}$$

Security proof: $\text{Sim}(b) = (r_1, \dots, r_{n-1}, b \oplus r_1 \oplus \dots \oplus r_{n-1})$, where $b = C(x)$

Suppose that we have a circuit $C : \{0, 1\}^m \rightarrow \{0, 1\}$ that outputs a single bit. We want to have a Garbling algorithm **Garb** and an Evaluation algorithm **Eval** where:

1. $\text{Garb}(k, C) \rightarrow (\tilde{C}, \{lb_{i,b}\}_{i \in [m], b \in \{0,1\}})$, where k is a security parameter, lb is a label, and $[n] = \{1, \dots, n\}$. The output is a Garbled circuit and n labels.
2. $\text{Eval}(\tilde{C}, lb_1, x_1, \dots, lb_n, x_n) \rightarrow b' \in \{0, 1\}$

Correctness: $\forall x \in \{0, 1\}^n$; and for the variables as above:

$$\text{Eval}(\tilde{C}, lb_1, x_1, \dots, lb_n, x_n) = C(x)$$

Informally, this states that output of the evaluation algorithm should be the same as the original circuit C on x .

Security: $\exists \text{Sim}_{PTT}$ such that $\forall x$:

$$(\tilde{C}, \{lb_{i,x_i}\}_{i \in [n]} \mid x) \stackrel{C}{\cong} (\text{Sim}(C, C(x)) \mid x)$$

Informally, the Garbled circuit should only be able to compute $C(x)$ and not learn extra information such as $C(x + 1)$. Even if a third party knows x , they cannot distinguish the two distributions of generated garbled labels. A consequence of this definition is that for two inputs x, x' if $C(x) = C(x')$, then the distribution that corresponds to x should be computationally indistinguishable from x' .

Question: If we have a garbled circuit for a complex circuit, does that imply any form of cryptography?

Answer: We don't know the answer to this question. We don't even know if they imply a one-way function. This serves as a motivation to build **Garbled Circuits** in an **Information-Theoretic** manner.

21.5 Information Theoretic Garbled Circuits

Definition 27 (Garbled Circuits for Degree-One Functions). Suppose that for input $x \in \{0, 1\}^n$ we have a function $C(x) = x_i$. We need a **Garb** and **Eval** algorithm such that **Eval** only reveals the i th bit of x and nothing else about x .

For all constructions in this lecture, the $\tilde{C}$ will always be empty. Thus, we only need to focus on the garbled labels generated denoted by a $2 \times n$ matrix. We start by generating $n - 1$ bits sampled uniformly at random where the n th bit is $r_n = r_1 \oplus r_2 \dots \oplus r_{n-1}$.

$$\begin{bmatrix} lb_{1,0} & \dots & lb_{n,0} \\ lb_{1,1} & \dots & lb_{n,1} \end{bmatrix} = \begin{bmatrix} r_1 & \dots & r_{i-1} & r_i & r_{i+1} & \dots & r_n \\ r_1 & \dots & r_{i-1} & \tilde{r}_i & r_{i+1} & \dots & r_n \end{bmatrix}$$

The Evaluation algorithm **Eval** simply receives n bits $(lb_1, \dots, lb_n)$ and outputs $lb_1 \oplus \dots \oplus lb_n$.

Correctness: Consider a label lb_{j,x_j} :

$$lb_{j,x_j} = \begin{cases} r_j & \text{if } j \neq i, \\ r_j & \text{if } j = i, x_i = 0, \\ \tilde{r}_j & \text{if } j = i, x_i = 1 \end{cases}$$

If $x_i = 1$, then $r_1 \oplus \dots \oplus \tilde{r}_i \oplus \dots \oplus r_n = 1$. If $x_i = 0$, then $r_1 \oplus \dots \oplus r_i \oplus \dots \oplus r_n = 0$ as r_n was generated as $r_n = r_1 \oplus r_2 \oplus \dots \oplus r_{n-1}$.

Security: We need to argue that these sequence of garbled labels will not reveal any information beyond $C(x)$ and that this sequence can be simulated.

Our simulation algorithm Sim_{PPT} (dc PTT or PPT) receives the output of the circuit on a given input. Sim_{PPT} should output a $\tilde{C} = \{\}$ and n labels. For output $b \in \{0, 1\}$, $\text{Sim}_{PPT}(b)$ generates $n - 1$ bits uniformly at random to create labels $lb_1 = r_1 \dots lb_{n-1} = r_{n-1}$, $lb_n = r_1 \oplus \dots \oplus r_{n-1} \oplus b$ (why b here). The distributions of labels generated by Sim_{PPT} and **Garb** are perfectly identical and thus statistically indistinguishable, since a viewer cannot tell whether the garbled labels are coming from the simulation algorithm or the garbling algorithm.

21.6 Garbling a Family of Functions

Consider the following family of functions $C(x) = \sum_{i=1}^n (x_i s_i) \pmod{2}$ for the inner product of a string $s \in \{0, 1\}^n$. This is now a family of functions. If you have a garbled circuit and a set of labels and input, you should only be able to compute the function on that input and learn the output, and nothing else. **Garb** then becomes:

$\text{Garb}(1^k, C, s) = (\tilde{C}, \{lb_{i,b}\})$. The security parameter k is not needed for information-theoretic garbled circuits and our **Eval** remains the same.

For **correctness**, we need $\forall s \in \{0, 1\}^n$ and any input x , $C_s(x) = \text{Eval}(\tilde{C}, \{lb\}_{i,x_i})$.

For **security**, our generated labels should only allow us to learn $C_s(x)$, and not learn s or x . Our simulation algorithm Sim_{PPT} will again only receive the final output. For $\forall x$ and s ,

$$(\tilde{C}, lb_{i,x_i} \mid x, s) \cong (\text{Sim}(C(x)) \mid x, s)$$

Thus, even if x and s are known, these two distributions are statistically indistinguishable.

Let us now try to build a **Garbled Circuit** for this family of functions. As a first attempt, let us try the following:

$$\begin{bmatrix} lb_{1,0} & \dots & lb_{n,0} \\ lb_{1,1} & \dots & lb_{n,1} \end{bmatrix} = \begin{bmatrix} 0 & 0 \dots & 0 \\ s_1 & s_2 \dots & s_n \end{bmatrix}$$

For **correctness**, $\text{Eval}(lb_1, \dots, lb_n) = \bigoplus_{i=1}^n lb_i$. This is because if $x_i = 1$, then $s_i = 1$ and because 0 is the identity element for $\bigoplus$ operation. However, for **security**, this construction is not secure because if $x \in \{1\}^n$, then the labels correspond to the values of s .

To fix our first attempt to obtain a security guarantee, we will add random noise to our labels to mask the value of s . We obtain $n-1$ bits generated uniformly at random and let $r_n = r_1 \oplus \dots \oplus r_{n-1}$. Our labels now become:

$$\begin{bmatrix} r_1 & \dots & r_n \\ r \oplus s_1 & \dots & r_n \oplus s_n \end{bmatrix}$$

Correctness is still maintained (exercise left for the reader). For **security**, the simulation algorithm is defined exactly as before since we generated our labels again by sampling $n-1$ bits uniformly at random. The resulting two distributions are now perfectly indistinguishable.

21.7 Garbled Circuits for Complicated Functions

We conjecture that we can build garbled circuits for even more complicated functions. Recursively, if we can build a garbled circuit for degree-1 functions, then we can inductively build garbled circuits for degree-2 functions and so on. For degree-1 functions if we define C as follows:

$$C = x_i s_i + f \text{ where } s \in \{0, 1\}^n \text{ and } f \in \{0, 1\}; f \text{ is a single bit function.}$$

The only change for our label matrix above is that $r_n = r_1 \oplus \dots \oplus r_{n-1} \oplus f$. Our correctness and security properties still hold.

Now for degree-2 functions, the functions will be of the following form:

$$C(x) = x_1 x_2 \text{ or in general, } C(x) = \sum_{i,j} x_i x_j \text{ for different pairs } (i, j).$$

Suppose, our circuit C is now $C : \{0, 1\}^2 \rightarrow \{0, 1\}$ and $C(x, y) = xy$. Our garbled circuit is still $\tilde{C} = \{\}$; however each label is now a pair of bits. Correspondingly the evaluation algorithm will be:

$$\text{Eval}((l'_1, b'_1), (l'_2, b'_2)) = l_1 l_2 \oplus b_1 \oplus b_2.$$

We apply a similar masking technique as before to our function by letting xy be,

$$xy = (x \oplus r_0)(y \oplus r_1) \oplus x r_1 \oplus (y r_0 \oplus r_0 r_1)$$

The last three terms are added to remove the noise once xy is successfully computed. Our labels now become:

$$\begin{bmatrix} (r_0, 0) & (r_1, r_0 r_1) \\ (\tilde{r}_0, r_1) & (\tilde{r}_1, r_0 \oplus r_0 r_1) \end{bmatrix}$$

Where $r_0, r_1 \leftarrow \{0, 1\}$ are generated uniformly at random. The evaluation algorithm is Eval shown above. This construction satisfies **correctness** but it is not **secure**. If $x = 1$, and the value of $r_1 = 1$ in the label pair $(\tilde{r}_0, r_1)$ then we can always determine that $x = 1$. This always allows them to learn the value of $x r_1$. Therefore, we mask it with third random value $r_2 \leftarrow \{0, 1\}$ sampled uniformly at random. Thus our equation now becomes:

$$xy = (x \oplus r_0)(y \oplus r_1) \oplus (x_1 r_1 \oplus r_2) \oplus (y r_0 \oplus r_0 r_1 \oplus r_2).$$

Our new set of labels becomes,

$$\begin{bmatrix} (r_0, r_2) & (r_1, r_0 r_1 \oplus r_2) \\ (\tilde{r}_0, r_1 \oplus r_2) & (\tilde{r}_1, r_0 \oplus r_0 r_1 \oplus r_2) \end{bmatrix}$$

This construction is secure. Our simulation algorithm **Sim** now takes in a single bit b and outputs two pairs of labels:

$$\text{Sim}(b) \rightarrow ((l'_1, b'_1), (l'_2, b'_2))$$

Sim samples l_1, l_2 and b uniformly at random. If we set $b_2 = b \oplus l_1 l_2 \oplus b_1$. Setting b_2 in such a manner ensures that our **Eval** algorithm holds.

The motivation behind this is that from degree-one functions we can go to degree-two. The function here was on two inputs but we can extend this construction to any function in this family.

Now let $C_{i,j}(x) = x_i * x_j \pmod{2}$ where $x \in \{0, 1\}^n$. Does our construction generalize? Yes, because $x_i x_j$ can be written as:

$$x_i x_j = (x \oplus r_i)(x_j \oplus r_j) \oplus (x_i r_j \oplus r'_i) \oplus (x_j r_i \oplus r'_j \oplus r_i r_j)$$

This equation contains inner products, which we know how to garble. As before, each label is a pair of bits but now we have n labels. Each function in this family is specified by a pair of bits, i and j . One insecure way to do this is to form labels for column i and column j and then tell the evaluator to only look at those labels. This obviously leaks information about i and j . Another idea would be to do $x_i x_j$ for all pairs. All the incorrect pairs will be 0 and then the $\oplus$ returns the correct $x_i x_j$.

Consider $f_{i,j} = x_i * x_j$. We add random noise by sampling $r, r' \leftarrow \{0, 1\}$ uniformly at random. Then,

$$\begin{aligned} f_{i,j} &= (x_i + r)(x_j + r') + (x_i r' + x_j r + r r') \\ f_1 &= (x_i + r) \\ f_2 &= (x_j + r') \\ f_3 &= (x_i r' + x_j r + r r') \end{aligned}$$

The garbling function and labels will be,

$$\begin{aligned} \text{Garb}(f_{i,j}) &= (\text{Garb}(f_1), \text{Garb}(f_2), \text{Garb}(f_3)) \\ lb'_{i,b} &= \{lb_{i,b}^{(1)}\} \quad lb'_{i,b} = \{lb_{i,b}^{(2)}\} \quad lb'_{i,b} = \{lb_{i,b}^{(3)}\} \\ lb'_{(i,b)} &= (lb_{(i,b)}^{(1)}, lb_{(i,b)}^{(2)}, lb_{(i,b)}^{(3)}) \end{aligned}$$

21.8 Randomized Encodings

For Eval, we evaluate per function based on its labels, $\alpha_1 = \text{Eval}_{f_1}(lb_1^{(1)}, \dots, lb_n^{(1)})$ and similarly for α_2 and α_3 . The output then becomes $f_{i,j} = \alpha_1 \alpha_2 + \alpha_3$. Notice that each garbled label consists of three labels and the number of labels grows with the number of gates. Therefore, we can only supports circuits of log security depth known as AC^1 circuits; the AC^i class of circuits with depth $O(\log^i n)$ and a polynomial number of unlimited fan-in AND and OR gates.

Let us now consider garbled circuits for *degree d* functions. Consider the function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ where the depth of the circuit C_f that encodes f is d (f is a *degree-d* function). We can probabilistically encode f as $g = (g_1, \dots, g_n)$ such that for $t \in [2^d]$ where each $g_i : \{0, 1\}^n \rightarrow \{0, 1\}^t$.

We claim that if we have a randomized encoding of a function, then we can immediately create a garbled circuit. We require 3 properties:

1. **Correctness:** We have a PPT algorithm Const such that for all x , $\text{Const}(g_1(x), \dots, g_n(x)) = f(x)$.
2. **Structural Property:** Each function g_i is a *degree-one* function that depends only on x_i (the i -th bit in x), $g_i(x) = (a_1 x_i + b_1, \dots, a_t x_i + b_t)$; $a_1, b_1, \dots, a_t, b_t \in [0, 1]$
3. **Security:** $g(x)$ reveals no information about x other than the value of $f(x)$. For all x and y such that $f(x) = f(y)$ we have $(x, y, g(x)) \cong (x, y, g(y))$. These two distributions should be identical since we require information theoretic garbled circuit.

If any of these conditions are removed then the resulting construction will be trivial. For example, if $f(x) = x_1 x_2 x_3$ and the encoding is $(g_1(x) = x_1, g_2(x) = x_2, g_3(x) = x_3, \dots)$. The structural

and correctness properties holds as $\text{Const}(b_1, b_2, b_3, \dots) = b_1 b_2 b_3$. However, clearly the security property doesn't hold. Another trivial example is if we don't care about the correctness property, $\text{Const}(\dots) = \perp$.

If we have such a randomized encoding for a function f , we can immediately convert it to a garbled circuit for f . As usual, $\tilde{C} = \{\}$. The garbling algorithm will be $\text{Garb} = (\tilde{f}, lb_{i,b})$ where $lb_{i,b} = g_i(b) \in \{0, 1\}^t$. The corresponding evaluation algorithm will be $\text{Eval}(lb_1, \dots, lb_n) = \text{Const}(lb_1, \dots, lb_n)$. The correctness property comes from the structural property as $lb_{i,b}$ only depends on the i th bit of the labels.

Let us now see how to build randomized encodings for *degree- d* functions. For *degree-one* functions of the form $f(x) = \langle a, x \rangle + b$ (inner product of n -bit string a with x added to b), define $g = (g_1, \dots, g_n)$ as $g_1(x) = a_1 x_1 + b$ where $b \in \{0, 1\}$ and $g_i(x) = a_i x_i$ for $i \geq 2$. This randomized encoding is insecure because the person knows f , a and b . To fix this, we sample $w_1, \dots, w_n \in \{0, 1\}$ uniformly at random such that $\sum w_i = 0$. Then each randomized encoding function becomes $g_i^*(x) = g_i(x) + w_i$ and $\sum g_i^*(x) = \sum g_i(x)$. Note that each g_i outputs a single bit. Therefore, to go from degree v to $v + 1$ the value of $t \in [2^d]$ will double, which is why we can only support AC^1 circuits.

Claim 21.1. If we have a randomized encoding for a degree- v function, we can construct a randomized encoding for a degree- $v + 1$ function.

Proof. Let $g(x) = Q(x)Q'(x)$ such that degree of Q, Q' is at most v . We can always write $g(x)$ as either $Q(x)Q'(x)$ or $Q(x) + Q'(x)$ by considering a circuit for $g(x)$ where the top-level gate is either an AND or an OR gate. Now we can use REs recursively. Let us now define a general function $f(x) = (Q(x) + r, Q'(x) + r', rQ'(x) + r'Q(x) + rr')$. The randomized encoding for $f(x)$ will be

$$\begin{aligned} Q(x) + r &= (g_1(x), \dots, g_n(x)) \\ Q'(x) + r' &= (g'_1(x), \dots, g'_n(x)) \\ rQ'(x) + r'Q(x) + rr' &= (h_1(x), \dots, h_n(x)) \end{aligned}$$

Lemma: Consider the function $f(x) = Q(x) + Q'(x)$ where both $Q(x)$ and $Q'(x)$ have an RE, and $g(x) = [Q(x) + r, Q'(x) + r', r + r']$. The randomized encoding for each gate should be computed bottom up in the circuit. Our set of labels similarly will be

$$((g_1(x_1), g'_1(x_1), h_1(x_1)) \dots ((g_n(x_n), g'_n(x_n), h_n(x_n))) \quad (21.1)$$

where $\text{Const}(g_1(x_1), \dots, g_n(x_n)) = Q(x) + r = \alpha_1$. This is similarly done for $Q'(x) + r' = \alpha_2$ and $r + r' = \alpha_3$. □

Claim 21.2. We can garble a function of any degree using randomized encoding (RE).

Proof. Without loss of generality, let us assume that our function $F(x)$ can be represented as a circuit $C : \{0, 1\}^{kn} \rightarrow \{0, 1\}$ where we have k input wires of size n and where each gate in the circuit is either a multiplication (AND) or an addition gate (OR). Furthermore, the C has a fan-in of 2 and

a fan-out of 1.

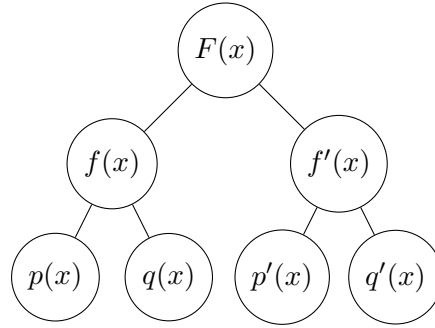
The garbling of C and the corresponding proof follows a bottom up approach where we begin at *level 0* (the input literals) and work our way up.

Base Case: Depth One: Gates with Input Wires

This is equivalent to a *degree-1* function of the form $f(x) = \langle a, x \rangle + b$ which can be garbled by using a randomized encoding as shown in Eq. [insert num].

Inductive Hypothesis: Depth $d-1$ $f(x) = p(x)q(x)$ or $f'(x) = p'(x) + q'(x)$

Consider the sub-tree induced under a gate at *depth* $d - 1$ with a left child and a right child. Let $p(x)$ and $q(x)$ again be of the form $f(x) = \langle a, x \rangle + b$ where $x, c \in \{0, 1\}^n$ and $b \in \{0, 1\}$. Assume that both $p(x)$ and $q(x)$ can be garbled via REs.



-

Inductive Step: Randomizing $f(x)$ or $f'(x)$ We begin with $f(x) = p(x)q(x)$,

$$f(x) = p(x)q(x)$$

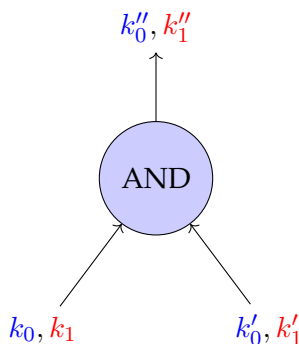
$$f(x) = (p(x) + r)(q(x) + r') + r'p(x) + rq(x) + rr'$$

From our inductive hypothesis, we can create REs for both $p(x)$ and $q(x)$. From (21.1), we showed that a function of such form can have an RE created for it. Thus, we can create an RE for $f(x)$ and in turn garble the gate that it represents. The same argument follows for $f'(x) = p(x) + q(x)$ which is transformed into $f(x) = (p(x) + r) + (q(x) + r')$. We can then apply our lemma from (21.1) to construct an RE for $f'(x)$.

Thus, by induction we are able to construct a randomized encoding for each gate in the circuit using a bottom-up approach, allowing us to garble the top level gate and as a result the entire circuit. $\square$

21.9 Secure MPC with Constant Round Complexity

Let us go back to Yao's construction for secure two party computation with Alice and Bob. For each wire in the circuit, we have two keys. And for any gate in the circuit, we generate a table as follows:



In the circuit above, Alice would like Bob to learn the product of the two private input bits. Alice can convey this information in a way that ensures Bob learns only one output key (one of k''_0 or k''_1). She can achieve this using garbled table, which allow the keys to be connected in a manner consistent with the underlying circuit. During the protocol, Alice will:

1. Generate random keys for each wire.
2. Connect the keys together via a garbled table. The invariant is that if Bob has k_0 and k'_1 , he must be able to obtain k''_0 because $0 \wedge 1 = 0$. At this step, Bob doesn't have any key yet as this is only a sequence of encryptions.
3. Send the garbled table along with k_b and (k''_0, k''_1) to Bob.
4. Engage in a (1,2) OT, allowing Bob to learn $k'[b]$.

Garbled Table for AND gate

$E(k_0, E(k'_0, k''_0))$
$E(k_0, E(k'_1, k''_0))$
$E(k_1, E(k'_0, k''_0))$
$E(k_1, E(k'_1, k''_1))$

Note that the two keys generated for each wire and the garbled table for each gate can be generated in parallel. So in the two-party setting where we used garbled circuit with Oblivious Transfer (OT), one party generated the garbled circuit with the corresponding labels and the other party evaluated the circuit using the labels. *How can we extend this construction to more than two parties?*

Let's say we have n parties. Let party one generate the entire garbled circuit. This party shouldn't evaluate the garbled circuit so we let the last party evaluate the garbled circuit. This could potentially allow us to evaluate in a constant number of rounds. The last party needs to get all sets of labels which can be done via OT. But as we will see this construction is not secure.

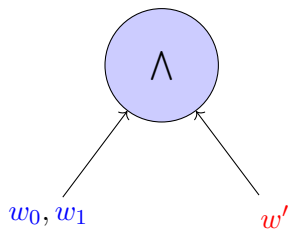
Imagine we have 3 parties and the corresponding circuit H might have a large depth. We would like to design a protocol that does not grow with the depth of the circuit. To begin, the first party generates the entire garbled circuit $\tilde{H}$. Now, each party via OT learns their corresponding labels and then we designate one of the parties evaluate on all the labels.

1. Let P_0 generates circuit $H = (\tilde{H}, \{Z_{b,i} | b \in \{0, 1\}, i \in [3n]\})$
2. P_1 and P_2 learn their corresponding garbled labels from P_0 via OT.
3. P_1 and P_2 send their labels to P_0 which evaluates the function.

Due to step three, this protocol is not secure because P_0 generated the garbled circuit. However, how many rounds does this protocol have? The OT in step two can be done in 2 rounds. For step three, it will take 1 round to send labels to the designated evaluator and if all parties are supposed to learn the output, the total number of rounds is 4. This is a constant number of rounds despite the original circuit depth.

However, as stated earlier this protocol is not secure because the party who generates the garbled circuit also evaluates the output. To fix this, we can have another party become the evaluator. P_0 and P_2 send their labels to P_1 which does the evaluation. In this setting, P_0 will also send the garbled circuit to P_1 and then P_1 can perform the evaluation computation. This protocol is secure as long as there is an honest majority and is not secure against collusion of any pair of parties. Even if the number of parties increases, the collusion is still possible with just two pairs of parties colluding (the garbled circuit creator and the evaluator).

However, if we consider the case where P_1 and P_2 collude, we can prove that at most they can only learn the output of the function (*Exercise left for the reader*). To provide intuition for this, consider the following gate:



If the evaluator knows both labels for a wire, then they might be able to learn further information about other party's labels. If a party knows w_0, w_1 , and w' is not known, then For an *AND* gate, you can determine if w' is 0 or 1 by changing the value of w_0, w_1 and determining if the output changes. Thus, you will learn the value of the wire in the entire circuit. Therefore, you need to make sure that for each wire, each party only knows one label for each wire. Thus, garbled

circuits are only one-time secure.

So now, for the multiparty case we will not have a single party generate the garbled circuit. Step one is going to be performed in a distributed manner. What if f can be determined by the four input labels (w_0, w_1, w'_0, w'_1) , two output labels (w''_0, w''_1) and some randomness R . For every gate in the circuit, each party will have their share of the of the input label of the incoming wire $w_0^i, w_1^i, w_0'^{(i)}$ and $w_1'^{(i)}$ for n parties such that:

$$w_0 = \sum w_0^{(i)}, w_1 = \sum w_1^{(i)}, R = \sum R_i$$

We then define a function F where the input for i -th party will be:

$$F(w_0^{(i)}, w_1^{(i)}, w_0'^{(i)}, w_1'^{(i)}, w_0''^{(i)}, w_1''^{(i)}, R^{(i)}) = (w_0, w_1, w_0', w_1', w_0'', w_1'', R)$$

such that the output of this function is precisely the inputs for f :

$$f(w_0, w_1, w_0', w_1', w_0'', w_1'', R).$$

F can be evaluated via any MPC protocol we have seen so far with the number of rounds being equal to the depth of the circuit F . This independent of the circuit H as $\text{depth}(F) \ll \text{depth}(H)$. Thus you can evaluate this transformation any way you want including via the GMW protocol. Since we want to generate a garbled table for each gate, we can evaluate F for each gate in parallel.

To see why, let us say that the original function $H(x_1 \dots x_n)$ where each $|x_i| = n$. The original circuit has n^2 input wires but h can have a very big depth. Suppose h has m gates, each label has n bits including R . Each party has n bits for each $F_1 \dots F_m$. The parties will perform m secure computation in parallel for F_i each of which will generate the garbled table for each gate. Now the only thing missing is the corresponding sequence of garbled inputs which one of the parties should learn as the evaluator. This can be done using another secure computation protocol to ensure for example that only one party learns the output of H .

The number of rounds depends on the depth of each row of the garbled table. The depth of each F_i is independent of the original function H but is still not constant. The depth can be made constant to 4-rounds as shown in “*The Round Complexity of Secure Protocols.*” [Beaver et al. \[1990\]](#).

21.10 Secure Computation Conclusion

If I have n parties and $t < n/2$ can collude (honest majority), then we can have do secure computation information-theoretically. But we saw how to build MPC protocols that are computationally secure even if $n - 1$ parties collude. Thus, can we have a protocol that provides information-theoretic security in the dishonest majority case? For example, for the function $f(x_1, x_2, x_3)$, can we have information-theoretic security even if two parties (such as x_1 and x_2) collude and share information and not learn the input of the last party, even if the colluding parties are computationally unbounded? This can be done via XOR secret sharing but this property is unique to XOR gates and not an AND or OR gate.

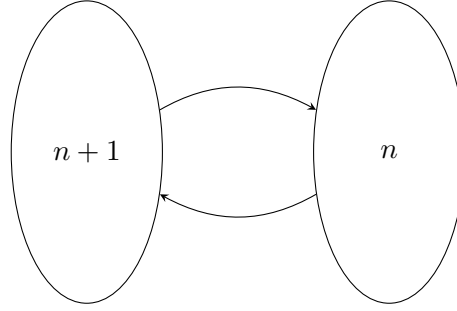
Claim 21.3. If we have a multi-party protocol that can provide information-theoretic security in the dishonest majority case, then we can also build Oblivious Transfer in an information-theoretic way.

Remark: Oblivious Transfer was a protocol that was between two parties and we cannot build OT in an information-theoretic way since we cannot have an informationally-theoretically secure public key encryption scheme.

Proof. Suppose we have an information-theoretic protocol for $f(x_1, x_2, x_3)$. The underlying function f can be represented as:

$$f(b, x_0, x_1) = x_0b + \overline{x_1}b$$

. Let us generalize this to any number of parties. Let $f(x_1 \dots x_{2n+1})$ be a multi-party function for $m = 2n + 1$ parties that has information-theoretic security in the dishonest majority case ($n + 1$). We can still reduce f to provide us with an information-theoretic Oblivious Transfer as follows.



In each set above, we have one of the two parties engaging in OT and $(2n + 1) - 2$ dummy parties divided between the two sets. Even though in MPC messages are sent from one party to every other party, for the diagram above we only consider the messages sent between the two sets. For the set on the left, the inputs to f are $((m_0, m_1), \perp_1 \dots \perp_n)$ and for the set on the right, the inputs are $(b, \perp_1 \dots \perp_{n-1})$. Since f is information-theoretic secure for the dishonest majority case, even if all $n + 1$ parties on the left collude after OT is performed they will not be able to learn the value of bit b . The same is true for the right, with all parties being unable to determine both m_0 and m_1 even with collusion.

Final Remark: In 2018, a sequence of papers came out that showed that MPC can be done in 2-rounds. 1-round MPC is not possible as proved in **Claim 7.1**. $\square$

22 Zero Knowledge Proofs

Let us consider ZKP for **NP** languages.

Definition 28 (Nondeterministic Polynomial Time (**NP**) Language L). For an **NP** language L , for any $x \in L$ and for some witness w of length $\text{poly}(|x|)$, we have a relation R such that $R(x, w) = 1$. Furthermore, for any $x \notin L$ there does not exist a witness w .

For example, consider the problem of graph 3-colouring..

Definition 29 (Graph 3-colouring). Given a graph $G(V, E)$, the colouring problem asks for an assignment of at most 3 colours to the vertices $c : V \rightarrow \{1, 2, 3\}$. We say that a colouring is proper if adjacent vertices receives different colours: $\forall (u, v) \in E : c(u) \neq c(v)$.

For every 3-colourable graph, the witness w is simply the colours assigned to all the vertices in V . For ZKP, we want to do the following: Suppose we have the prover with the statement x with w and want to convince the verifier that x is in the language. They want to do so via a protocol such that by the end the verifier is convinced or not. These are referred to as **proof systems**.

This problem becomes non-trivial if we want the property of zero-knowledge. That is, at the end of the protocol we don't want the verifier to learn any information beyond the fact that x is in the language or the witness w itself.

Definition 30 (Proof System). A *proof system* is a protocol that allows a prover with x and a witness w , $P(1^k, x, w)$, to interact with a PPT Verifier $V(1^k, x)$ where k is the security parameter. Let $\text{out}(P, V)$ represent the output of the verifier at the end of the protocol and let $\text{view}(P, V)$ represent the view of the verifier at the end of the protocol. The view consists of the transcript Π of the protocol and any randomness used by the verifier.

In our instance, P is an honest PPT prover. We require the following three properties to hold.

1. *Completeness*: $\Pr[\text{out}(P, V) = 1] = 1$.

If $x \in L$ and w is a witness for L , then by the end of the interaction V should be convinced that $x \in L$.

2. *Statistical Soundness*: $\Pr[\text{out}(P^*, V) = 1] = \text{negl}(k)$.

If $x \notin L$, and for any malicious unbounded prover P^* that could run in **EXP**.

Note: If P^* is computationally bounded (such as PPT), then our proof system becomes an *argument system*.

3. *Computational Zero Knowledge*: $S(x) \cong^c \text{view}[P(1^k, x, w), V^*(1^k, x)]$

Where for any malicious PPT verifier V^* , $\exists$ a PPT simulator $S : x \rightarrow \text{view}(P, V)$. Importantly, V^* can deviate from the protocol to try and learn more information about the witness w .

This definition can be made weaker to *Computational Honest Verifier Zero Knowledge* where we don't enumerate over all possible PPT verifiers V^* . We instead restrict V^* to only be an honest verifier that doesn't deviate from the protocol.

The motivation behind *proof systems* in general without the *Zero Knowledge* property is the setting where a computationally unbounded prover P wants to convince a computationally bounded verifier V that for a particular language L , $x \in L$. For example, it could be the case that $L \in \text{EXP}$ and P would like to convince the inclusion of a particular x to be in L for a PPT V .

The first example we begin with is Graph Non-Isomorphism (GNI).

Definition 31 (Graph Isomorphism). Let $G_0 = (V_0, E_0)$ and $G_1 = (V_1, E_1)$ be undirected graphs. An *isomorphism* from G_0 to G_1 is a bijection $\varphi : V_0 \rightarrow V_1$ such that for every pair of vertices $u, v \in V_0$,

$$\{u, v\} \in E_0 \iff \{\varphi(u), \varphi(v)\} \in E_1.$$

If such a map exists, we write $G_0 \cong G_1$ and say that G_0 and G_1 are *isomorphic*.

Note. G_0 and G_1 are *homomorphic* if φ preserves edges and not necessarily non-edges.

Definition 32 (Graph Isomorphism Language). Let $G_I = (G_a, G_b)$ be the language containing the pair of graphs that are isomorphic.

$$\text{GRAPH ISOMORPHISM} \in \text{NP}$$

Note. It is widely believed by complexity theorists that $\text{GRAPH ISOMORPHISM} \in \text{P}$.

Definition 33 (Graph Non-Isomorphism (GNI)). Let G_1 and G_2 be finite graphs. The *Graph Non-Isomorphism* language is

$$\text{GNI} = \{ \langle G_1, G_2 \rangle \mid G_1 \not\cong G_2 \},$$

i.e. the set of all encoded pairs of graphs that are *not* isomorphic.

For our proof system, the prover runs in **EXP** time while the verifier runs in linear time.

Algorithm 4 Interactive Proof System for GNI

- 1: **Verifier:** pick random bit b and permutation π ; send $H \leftarrow \pi(G_b)$.
- 2: **Prover:** send guess b' .
- 3: **Verifier:** accept iff $b' = b$.

1. *Completeness*: $\Pr[\text{out}(P, V) = 1] = 1$.

If $x \in L$ and both the prover and verifier act honestly, then the verifier will be able to determine the value correct value for b' .

2. *Statistical Soundness*: $\Pr[\text{out}(P^*, V) = 1] \leq \frac{1}{2}$.

When $G_0 \cong G_1$, the value of b is perfectly hidden by the permutation π , regardless of whether P^* is in **EXP**.

As an aside, this illustrates an active research area for cryptography known as *Delegation of Computation* where a bounded participant offloads computation to a powerful user and wants to ensure both soundness and correctness.

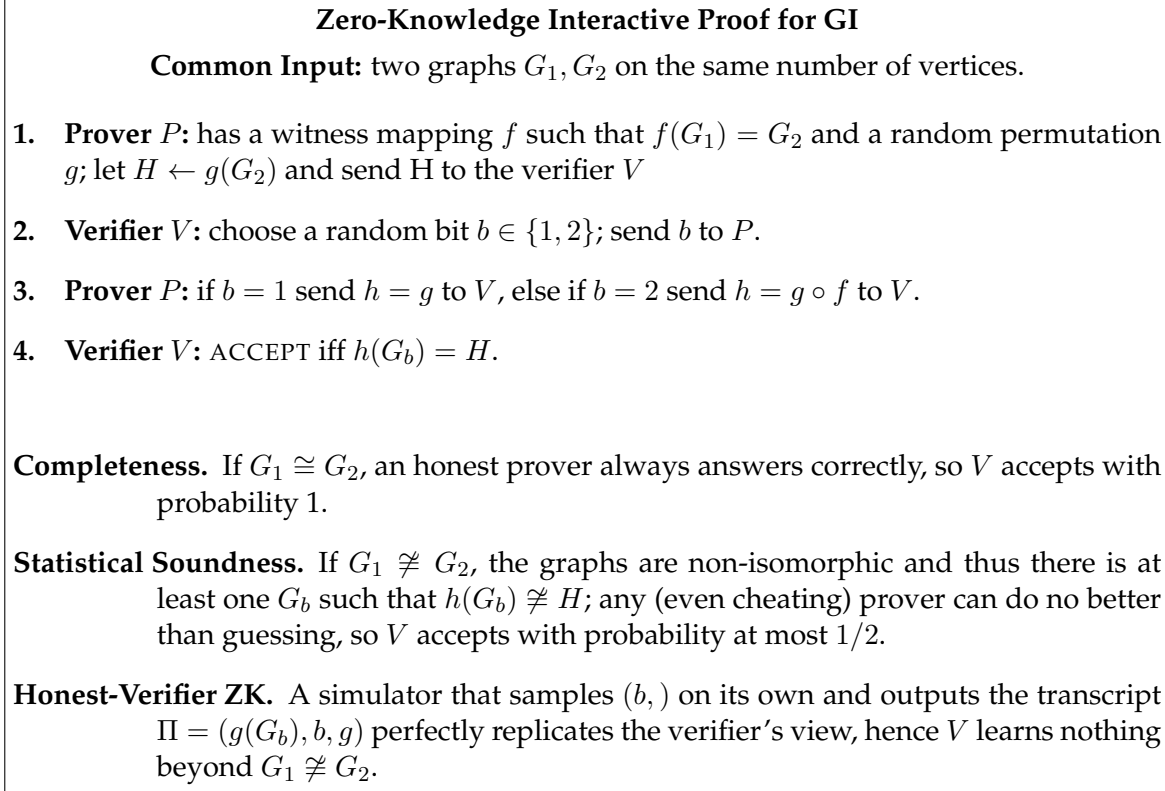


Figure 1: Zero-knowledge protocol for GI.

To reduce the soundness error, we can amplify our success by running the protocol multiple times. However, now it matters for ZK if the protocol is run in sequential or in parallel. For the honest verifier setting, the view of the verifier can be simulated by a PPT simulator S where the bit b is picked first uniformly at random and then used to pick a random permutation of G_b for the first first message $(GI(G_b), b, GI)$ where GI is the set of all isomorphic graphs.

For an unbounded and malicious verifier V^* , we need to show that there exists a PPT simulator S that generates the view for V^* . The simulator interacts with V^* in a black-box manner:

1. S randomly guesses a bit $b \leftarrow \{0, 1\}$.
2. S sends a random isomorphism $H \leftarrow g(G_b)$ to V^* and gets back b' .
3. If $b = b'$, S is done and generates the last message h and sends it to V^* .
4. If $b \neq b'$, S repeats this game.

Since H is perfectly hiding the value of b , the $\Pr[b = b'] \leq \frac{1}{2}$. To decrease this error to $\text{negl}(k)$, we need to execute the protocol sequentially. If the protocol is executed in parallel, only *Honest-Verifier Zero Knowledge* can be guaranteed. Briefly, the intuition behind this is that in the parallel

execution, the simulator S has to guess potentially 2^k bits b at the same time, leading to exponential time complexity.

Furthermore for statistical Zero-Knowledge proof systems, you can convert Honest-Verifier Statistical Zero-Knowledge to Malicious-Verifier Statistical Zero-Knowledge in a round preserving manner. This is discussed further in "*An Efficiency-Preserving Transformation from Honest-Verifier Statistical Zero-Knowledge to Statistical Zero-Knowledge*". [Hubáček et al., 2018]

22.1 Zero-Knowledge for NP-Complete Problems

First, some important properties hold that motivate our discussion. Firstly, if one way functions exist then it can be shown that all of NP have *Zero-Knowledge* proofs. Secondly, for NP-complete problems we can only have computational *Zero-Knowledge* and statistical soundness.

Recall the definition for the *graph 3-colouring* problem:

Definition 34 (Graph 3-colouring). Given a graph $G(V, E)$, the colouring problem asks for an assignment of at most 3 colours to the vertices $c : V \rightarrow \{1, 2, 3\}$. We say that a colouring is proper if adjacent vertices receive different colours: $\forall (u, v) \in E : c(u) \neq c(v)$.

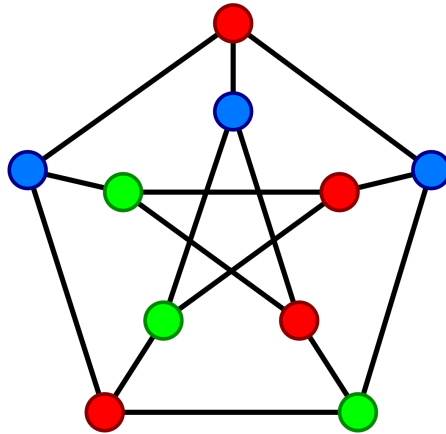


Figure 2: A 3-coloring of a graph.

Soundness If the graph is not 3-colourable, there must exist an edge in the graph $e_{i,j}$ such that $\phi(c_i) = \phi(c_j)$ (its two vertices must have the same colour). Since the public-key encryption scheme is publicly binding, the probability that the prover picks this edge is $\Pr[e_{i,j}] = \frac{1}{|E|}$.

Simulator $\mathcal{S}$ (Zero-Knowledge).

- S1. Pick a random edge $e = (u, v) \in E$ and a random permutation π of $\{1, 2, 3\}$.
- S2. Colour u with $\pi(1)$ and v with $\pi(2)$; colour *every other* vertex $w \in V \setminus \{u, v\}$ with the same colour $\pi(1)$.
- S3. For each vertex $w \in V$ choose fresh randomness r_w and output the ciphertext $c_w := \text{Enc}_{\text{pk}}(\text{col}(w); r_w)$ as the first message.
- S4. Let the verifier return a challenge edge $e^* = (u', v')$.
- S5. If $e^* = e$, open $c_{u'}, c_{v'}$ by revealing the plaintexts and randomness; output the resulting transcript. Otherwise REWIND the verifier and go back to S1.

Zero-Knowledge Proof for Graph 3-Colouring

Public input. Graph $G = (V, E)$ with n vertices.

Prover's secret. Proper 3-colouring $\phi : V \rightarrow \{1, 2, 3\}$.

Commitment Scheme. Public-key encryption scheme $\text{Enc}_{\text{pk}}(m; r)$ that is *semantically secure* and *perfectly correct*.

Setup. Verifier V generates (pk, sk) and sends pk to Prover P .

Interactive protocol (repeat Steps 1–4 for $k = O(\lambda)$ rounds):

Step 1: (Encrypt) P chooses a random permutation π of the colors. For every vertex $v \in V$ pick fresh randomness r_v and send the ciphertext

$$c_v := \text{Enc}_{\text{pk}}(\pi(\phi(v)); r_v) \quad \text{to } V.$$

Step 2: (Challenge) V selects a uniform random edge $e = (u, v) \in E$ and sends it to P .

Step 3: (Open) P reveals the plaintext–randomness pairs $(\pi(\phi(u)), r_u)$ and $(\pi(\phi(v)), r_v)$.

Step 4: (Verify) V accepts this round iff

- (a) $\text{Enc}_{\text{pk}}(\pi(\phi(u)); r_u) = c_u$ and $\text{Enc}_{\text{pk}}(\pi(\phi(v)); r_v) = c_v$, and
- (b) $\pi(\phi(u)) \neq \pi(\phi(v))$ (endpoints get different colors).

Decision rule. V outputs ACCEPT iff every round is accepted; otherwise REJECT. Soundness error after k rounds is at most 2^{-k} .

Figure 3: Step-wise zero-knowledge protocol for graph 3-colouring using public-key encryption.

Notice that $\Pr[e^* = e] = \frac{1}{|E|} - \text{negl}(k)$ and this probability depends on the security of our public-key encryption scheme. Contrast this with *Graph-Isomorphism* where the permutation perfectly hid the value of our randomly chosen bit b .

Amplifying Zero-Knowledge When this protocol is repeated sequentially to amplify soundness, the number of rounds clearly increases and the protocol is not constant round. Can we have constant round zero-knowledge with $\text{negl}(k)$ soundness error for *graph 3-colouring*? The answer is no; We **cannot** have constant round public coin with *black-box* zero-knowledge. Here, black-box refers to the simulator S and how it interacts with the verifier.

Definition 35 (Black-box Simulator). Let (P, V) be an interactive proof system for a language $L \subseteq \{0, 1\}^*$. A *black-box simulator* is a probabilistic polynomial-time algorithm S such that

- (i) S takes as input an instance $x \in L$ and has *oracle access* to every probabilistic polynomial-time verifier V^* (denoted $S^{V^*(x, \cdot)}(x)$); during its execution S may invoke the oracle, rewind it to any earlier state, and resume the interaction, but it never inspects the internal randomness or code of V^* .

- (ii) For every PPT verifier V^* and every $x \in L$, the random variable produced by the simulator is indistinguishable from the verifier's real view, i.e.

$$S^{V^*(x, \cdot)}(x) \cong \text{view}_{V^*}(\langle P(x), V^*(x) \rangle),$$

where the right-hand side is the joint distribution of all messages, random coins and internal state seen by V^* during an honest execution with P .

A protocol (P, V) is called *black-box zero-knowledge* if such an S exists.

Constant Round Zero-Knowledge Barak et. al showed in 2004 that there does not exist a two-round zero-knowledge **proof** system with perfect completeness for an NP-complete language among other important findings Barak et al. [2004]. Importantly, the constant-round protocol shown in Figure 3 is a private-coin model since Alice uses a public-key encryption scheme to announce her queries.

Constant-Round Zero-Knowledge Protocol for Graph 3-Coloring

Public input. Graph $G = (V, E)$. **Bob's secret.** Proper 3-coloring ϕ . **Primitive.** Public-key encryption Enc.

Round 1: Alice (Verifier) $\rightarrow$ Bob (Prover)

- (a) Generate (pk, sk) ; send pk .
- (b) Pick t random edges e_i ; encrypt to c_i ; send $\{c_i\}$.

Round 2: Bob $\rightarrow$ Alice

- (a) Choose random permutation π .
- (b) Encrypt permuted colors d_w ; send $\{d_w\}$.

Round 3: Open Commitments Verify

- (a) Alice reveals $\{e_i, r_i\}$; Bob checks $c_i = \text{Enc}(e_i; r_i)$.
- (b) Bob opens $\pi(\phi(u_i)), \pi(\phi(v_i))$ for every e_i .
- (c) Alice accepts iff each opening is correct *and* the two endpoints have different colors.

Accept iff every check passes; soundness error $\leq 2^{-t}$.

Figure 4: Constant-round (3-message) 3-color zero-knowledge protocol.

References

- Boaz Barak, Yehuda Lindell, and Salil Vadhan. Lower bounds for non-black-box zero knowledge. Cryptology ePrint Archive, Paper 2004/226, 2004. URL <https://eprint.iacr.org/2004/226>.
- D. Beaver, S. Micali, and P. Rogaway. The round complexity of secure protocols. In *Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing*, STOC '90, page 503–513, New York, NY, USA, 1990. Association for Computing Machinery. ISBN 0897913612. doi: 10.1145/100216.100287. URL <https://doi.org/10.1145/100216.100287>.
- Donald Beaver. Correlated pseudorandomness and the complexity of private computations. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 479–488, 1996.
- O. Goldreich, S. Micali, and A. 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. ISBN 0897912217. doi: 10.1145/28395.28420. URL <https://doi.org/10.1145/28395.28420>.
- Pavel Hubáček, Alon Rosen, and Margarita Vald. An efficiency-preserving transformation from honest-verifier statistical zero-knowledge to statistical zero-knowledge. Cryptology ePrint Archive, Paper 2018/164, 2018. URL <https://eprint.iacr.org/2018/164>.
- Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In *Annual International Cryptology Conference*, pages 145–161. Springer, 2003.