CSCI 5440: Cryptography
The Chinese University of Hong Kong, Fall 2020

Lecture 8
27 October and 2 November 2020

In Lecture 7 we described a protocol for two-party computation that is secure against honest-but-curious parties: No information is leaked beyond what is intended assuming both Alice and Bob follow the rules of the protocol. Information leakage can happen if, say, Bob deviates from his instructions. For example, in the AND protocol if Bob's input is zero he is required to generate a public key for encryption without knowing the corresponding secret key so that he cannot decrypt Alice's input. What is to prevent a cheating Bob from doing this?

It appears that we have to go back to the drawing board and redo the protocols so they are secure against such attacks. Fortunately we are doing theoretical cryptography and there is another way. We will describe a general-purpose "compiler" that turns any two-party computation that is secure against honest-but-curious parties into one that is secure against malicious parties that might not follow the protocol instructions.

The idea is deceptively naive: In addition to running the protocol, Alice and Bob prove to each other that they followed the instructions. For example, in the AND protocol Alice is supposed to send Bob the encryption of her input x under Bob's public key. Alice will accompany this message with a proof that the message $C = Enc(PK, x)$ is indeed an encryption of x under PK. She can prove this by revealing x and the randomness R she used and then Bob can check that C is indeed an encryption of x with randomness R. But revealing x (and R) breaks the simulatability of the protocol: This is precisely the information that Bob should not find out.

There is another type of proof, called a *zero-knowledge proof*, by which Alice can convince Bob that statements like "C is an encryption of my input under PK" are true without revealing any additional information to Bob. Let's start with a non-cryptographic example.

1 Graph isomorphism

An isomorphism from a graph G to a graph H (both on n vertices) is a bijection between their vertices that "preserves" the edges. More precisely, π is an isomorphism from G to H if for all pairs of vertices, (u, v) is an edge in G when and only when $(\pi(u), \pi(v))$ is an edge in H. If this is the case we write $\pi(G) = H$. For example the two graphs in Figure 1 are isomorphic.

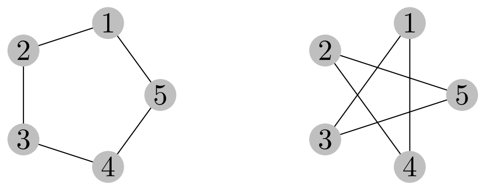

Figure 1: Two isomorphic graphs. One isomorphism is $\pi(1) = 1, \pi(2) = 3, \pi(3) = 5, \pi(4) = 2, \pi(5) = 4$.

In this example it was easy to see that the two graphs are isomorphic, but it is less clear what happens when the graphs are large (see Figure 2).

Suppose Alice knows an isomorphism π from G_0 to G_1 and wants to convince Bob that the two are isomorphic. One thing she can certainly do is send Bob π (namely, the list of values $\pi(1), \ldots, \pi(n)$ where n is the number of vertices). Bob can then verify that $\pi(G_1) = G_0$: He goes over all pairs of vertices (u, v) and checks that $(\pi(u), \pi(v))$ is an edge in H precisely in those cases when (u, v) is an edge in G_0. If G_0 and G_1 happened not to be isomorphic, no "proof" of Bob would pass.

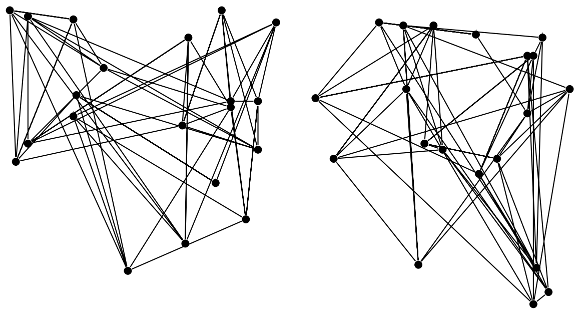

Figure 2: Two graphs that may or may not be isomorphic.

Now consider an "identification scheme" in which π is Alice's secret key and (G_0, G_1) is her public key. The challenge is for her to prove that G and H are isomorphic. Alice can certainly do this by furnishing π to Bob, but then Bob can impersonate her at will. Can she convince Bob that G and H are isomorphic without leaking any information about π?

Graph Isomorphism Protocol: Prover's input is G_0, G_1 and π. Verifier's input is G_0, G_1.

1. Prover chooses a random permutation ρ of $\{1, \dots, n\}$ and sends the permuted graph $\rho(G_0)$.
2. Upon receiving the graph G, Verifier sends a random bit b.
3. Prover sends ρ if $b = 0$ and $\rho \circ \pi$ (the permutation $(\rho \circ \pi)(u) = \rho(\pi(u))$ if $b = 1$.
4. Upon receiving ϕ, Verifier accepts if $\phi(G_b) = G$.

If G_0 and G_1 are isomorphic and both parties follow the protocol then Verifier will accept because G_0 and G_1 are both isomorphic to G and Prover supplies the correct isomorphism under both challenges $b = 0$ and $b = 1$.

If, on the other hand, G_0 and G_1 are not isomorphic then no matter which graph G Prover supplies it cannot be isomorphic to both G_0 and G_1. With probability half, Verifier's challenge b will identify the graph G_b that is not isomorphic to G, in which case no matter which "isomorphism" π Prover submits she won't pass validation.

This is an example of an *interactive proof*: After the interaction Verifier is convinced with some degree of certainty that G and H are isomorphic. If he wants more certainty he can ask Prover to run the protocol multiple times.

After running the protocol Verifier finds out that his inputs G_0, G_1 are isomorphic. What else does he learn? His view consists of a randomly permuted copy G of G_0 or G_1, the bit b, and an isomorphism ϕ from G_b to G. He can simulate his view by first picking b and ϕ then setting G to $\phi^{-1}(G_b)$. So Prover has learned nothing beyond the fact that G_0 and G_1 are isomorphic.

By this point you must have realized that the protocol we described is quite similar to Schnorr's identification scheme. Both are examples of zero-knowledge proofs. There is one subtle difference regarding the objectives. Graph isomorphism is a *proof of fact*: Alice convinces Bob that G_0 and G_1 *are* isomorphic, namely that there exists an isomorphism from G_0 to G_1. In contrast, security of Schnorr's identification required a *proof of knowledge*: Alice had to convince Bob that she knows SK such that $g^{SK} = PK$, not just that such an SK exists. While we will eventually want to talk about proofs of knowledge, proofs of fact are simpler to define. Let's do that first.

2 Zero-knowledge proofs

A *proof relation* R is a relation over pairs (x, π) called *the statement* and *the proof.* The statement "(x, π) satisfies R" means that π is a valid proof of x. A statement x is a *fact* if there exists a proof π such that (x, π) satisfies R. In the graph isomorphism example, x is the pair of graphs (G_0, G_1), π is the isomorphism, and $((G_0, G_1), \pi)$ satisfies R if and only if $\pi(G_0) = G_1$.

Definition 1. A *proof system* for R is a protocol between Prover P and Verifier V in which Verifier gets input x and Prover gets inputs x and π with the following two properties:

1. **Completeness:** If $(x, \pi) \in R$ then upon interacting with $P(x, \pi)$, $V(x)$ always accepts.
2. **Soundness:** If $(x, \pi) \notin R$ for every π then upon interacting with any P^* (that does not necessarily follow the protocol), $V(x)$ accepts with probability at most $1/2$.

The protocol for Graph Isomorphism has both properties: The honest prover passes verification, but no cheating prover can do so.

The security requirement is that Verifier should not learn anything except that x is a fact. Since the honest prover is only defined for inputs that satisfy the proof relation, security only needs to hold for such inputs.

Definition 2. The proof system (P, V) is (s, ε)-*honest-verifier zero-knowledge* if there is a simulator *Sim* such that for all $(x, \pi) \in R$, the verifier's view on input x is (s, ε)-indistinguishable from the random variable $Sim(x)$.

The graph isomorphism proof system is $(\infty, 0)$-honest-verifier zero-knowledge because the output of the simulator is identically distributed to the view of the verifier.

What about cheating verifiers that might not follow instructions? In the graph isomorphism protocol, Verifier has one opportunity to cheat by not choosing his challenge b at random. He can even make it depend on Prover's message G.

This is much like the analysis of Schnorr's protocol against impersonators. The same type of simulator works here. The simulator begins by making a random guess b for the cheating verifier's challenge b^*. It then samples the view expected by the verifier assuming the challenge is b, namely the pair $(\rho(G_b), \rho)$ for a random ρ. It then runs the cheating verifier on first message $\rho(G_b)$. Since $\rho(G_0)$ and $\rho(G_1)$ are identically distributed, the verifier's response b^* is independent of b, so the two are equal with probability half. Conditioned on this happening, $(\rho(G_b), \rho)$ together with the verifier's randomness is identical to the verifier's view, so the simulator can output this view. If not, the simulator tries again.

In this instructive example the simulator runs the cheating verifier, so its size will depend on the size of the cheating verifier. It is sensible to specify the simulator size as a function of the cheating verifier size. We call this function the simulation overhead.

Definition 3. The proof system (P, V) is (s, ε)-*zero-knowledge* with *simulation overhead oh* if for every verifier V^* of size t and every $(x, \pi) \in R$ there exists a simulator *Sim* of size $oh(t)$ for which the view of $V^*(x)$ when interacting with $P(x, \pi)$ is (s, ε)-indistinguishable from $Sim(x)$.

The graph isomorphism simulator has overhead $oh(t) = t + O(n \log n)$ for simulation error $\varepsilon = 1/2$. The $O(n \log n)$ term accounts for the complexity of sampling ρ and the verification. For lower error the simulator can be run multiple times, so for any k the protocol is $(\infty, 2^{-k})$-zero-knowledge with overhead $k(t + O(n \log n))$.

The intended use of zero-knowledge in the context of two-party computation is for the parties to prove to one another that they are following the rules of the protocol. These are statements like $h = g^{SK}$ that have nothing to do with graph isomorphisms. To handle them we will make use of a more general type of zero-knowledge protocol that can be used to prove *any* fact. For that we need one more ingredient.

3 Commitments

Let's go back to the oblivious transfer protocol from last lecture for a moment. To securely compute $OT(x_0x_1, b)$, Alice and Bob run AND protocols on inputs $(x_0, \overline{b})$ and (x_1, b). If Bob wants to convince Alice that he is playing by the rules, he has to prove to her that the inputs he uses in both runs refer to the same b (without revealing it). A cryptographic protocol for this is called a commitment scheme.

A commitment scheme consists of two phases. In the commitment phase, Sender sends a commmitment of her message M to Receiver. The commitment should bind Sender to M, but hide M from Receiver. In the disclosure phase, Sender sends M together with a certificate to Receiver who is convinced that M is the value that Sender committed to. Let's describe a protocol first and give the definitions later.

DDH commitments:

Commitment: Sender sends $(h, h', h'') = (g^X, g^Y, g^{XY} \cdot M)$ for random $X, Y \sim \mathbb{Z}_q$.

Disclosure: Sender reveals M and X, Y. Receiver verifies $h = g^X$, $h' = g^Y$, and $h'' = g^{XY} \cdot M$.

By the DDH assumption commitments to any two messages look alike to Receiver, so the commitment hides Sender's message. On the other hand, Sender cannot decommit to any message other than M because h and h' uniquely determine X and Y, and h'', X, Y uniquely determine M.

A *commitment scheme* is a two-phase protocol between Sender and Receiver so that (when both are honest) Receiver's output equals Sender's input with probability one. There are two security requirements:

- **Binding:** The probability that a (malicious) sender S^* of size at most s can make Receiver validate two different messages on the same commitment is at most ε.
- **Hiding:** The view of any (malicious) R^* in the commitment phase is (s, ε)-simulatable.

The above reasoning gives the following security guarantees. The proof of hiding is the same as the security analysis of El Gamal encryption.

Theorem 4. *Under the (s, ε)-DDH assumption, DDH commitments are $(\infty, 0)$-binding and $(s - t_\times, \varepsilon)$-hiding, where $t_\times$ is the size of a multiplication modulo the safe prime $2q + 1$.*

It is in fact possible to construct a commitment scheme from any pseudorandom generator.

4 Zero-knowledge proofs for all facts

A 3-coloring of a graph G is an assignment of one of three colors (red, green, or blue) to its vertices so that there are no conflicting edges. A conflicting edge is one whose endpoints have the same color. Figure 3 shows examples of valid and invalid 3-colorings. A coloring can be represented as a string in $\{\mathtt{R}, \mathtt{G}, \mathtt{B}\}^n$ with each symbol representing the color of the corresponding vertex.

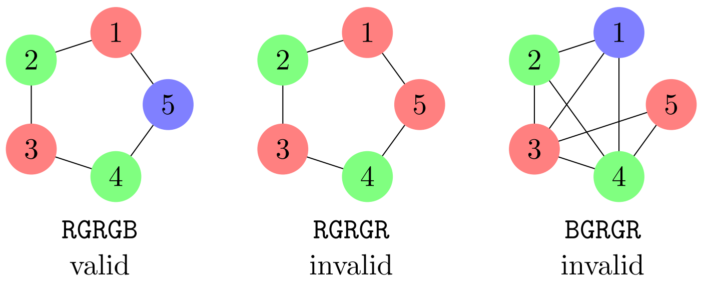

Figure 3: Valid and invalid 3-colorings. The second graph has a valid 3-coloring but not the third one.

The proof relation $3COL$ consists of graphs and their valid 3-colorings:

$$(G, \pi) \in 3COL \text{ iff } \pi \text{ is a valid 3-coloring of } G.$$

What makes $3COL$ special in contrast to graph isomorphism is that $3COL$ is a *complete* proof relation: Any fact can be converted into a graph so that (valid) proofs map to (valid) colorings. For example, if Alice wants to convince Bob that there exists (X, Y) such that his input (h, h', h'') is of the form (g^X, g^Y, g^{XY}), they can canonically convert (h, h', h'') into a graph G and (X, Y) into a coloring π so that

- If $h = g^X$ and $h' = g^Y$ and $h'' = g^{XY}$ then π is a valid 3-coloring of G, and
- If (h, h', h'') is not a DDH triple then G is not 3-colorable.

All that remains to do is to describe a zero-knowledge proof system for $3COL$.

Goldreich-Micali-Wigderson (GMW) proof system: Prover's input is a graph G with n vertices and m edges and a valid 3-coloring π. Verifier's input is G.

1. Prover randomly permutes the three colors in π (e.g., he replaces $\mathtt{R}$ with $\mathtt{G}$, $\mathtt{G}$ with $\mathtt{R}$ and leaves $\mathtt{B}$ intact) and sends commitments $(Com(\pi_1), \dots, Com(\pi_n))$ for the color of every vertex.
2. Upon receiving the commitments Verifier chooses a random edge (v, w) and sends it to Prover.
3. Prover discloses the colors π_v and π_w and the corresponding certificates to Verifier.
4. Verifier accepts if the certificates are correct and if $\pi_v \neq \pi_w$.

If π is a valid 3-coloring then $\pi_v \neq \pi_w$ are different for every edge (v, w), so the proof system is complete.

Claim 5. *If the commitments are perfectly binding*[1] *then for every G that is not 3-colorable and for every P^*, the probability that Verifier accepts G upon interacting with P^* is at most $1 - 1/m$.*

Proof. If G is not 3-colorable then for every first message $(C_1^*, \dots, C_n^*)$ there must exist an edge (v, w) for which Prover cannot decommit C_v^* and C_w^* to different colors. Otherwise, the decommitments would constitute a valid 3-coloring of G which does not exist. With probability at least $1/m$, Verifier challenges Prover on this exact edge, Prover is unable to provide certificates and verification fails. □

[1]It is enough to assume that they are (s, ε)-binding, but we have perfect binding so we might as well use it.

When the GMW proof system is applied, the number m of edges grows with the size of the fact being proved, so a cheating prover appears to have a solid chance of making Verifier accept a false proof. There are more robust ways to convert proofs of facts into 3-colorings of graphs so that if the fact is false, any 3-coloring of the graph violates some constant *fraction* of the edges like 10%. A cheating prover still has 90% chance of passing verification.

This *soundness error* can be lowered by repeating the protocol several times. After r independent repetitions the soundness error drops exponentially in r, so even if the original error is as large as $1 - 1/m$ after mk repetititons it becomes as small as $(1 - 1/m)^{mk} \leq e^{-k}$.

It remains to prove that the protocol is zero-knowledge. We consider cheating verifiers right away.

Theorem 6. *If the commitments are (s, ε)-hiding with simulation overhead $oh(t)$ and computable in size t_{Com} then the GMW proof system is $(s - nt_{Com}, (1 - 1/m + 2\varepsilon)^r + n\varepsilon)$-zero-knowledge with simulation overhead $r(n \cdot oh(t) + t + 2t_{Com} + O(\log n))$ for every r.*

Proof Sketch. The simulator for a cheating verifier V^* guesses the edge (v, w) at random. It generates commitments to two distinct random colors, e.g. $Com(\mathtt{R})$ and $Com(\mathtt{B})$ for vertices v and w and runs the commitment simulator to get "commitments" for all the other vertices. It sends these as a first message to V^* and receives a response (v^*, w^*). If $v = v^*$ and $w = w^*$ it reveals the commitments and certificates. Otherwise it tries again for up to r times.

By the hiding property, the first message generated by the simulator is $(s, 2\varepsilon)$-indistinguishable from n simulated commitments. On the other hand, n simulated commitments are independent of the choice of v and w so the probability that (v, w) equals V^*'s response (v^*, w^*) is $1/m$. The probability that this event will fail to happen in all r repetitions is therefore at most $(1-1/m+2\varepsilon)^r$.

Conditioned on $(v, w) = (v^*, w^*)$, the simulator's output is $(s-(n-2)t_{Com}, (n-2)\varepsilon)$-indistinguishable from V^*'s actual view. In both views, the commitments that are revealed to V^* are two independent random colors and all the others are independent. □

It turns out that the manner in which the proof system repetition is performed (in order to reduce the soundness error) affects zero-knowledge. The GMW proof system remains zero-knowledge if it is repeated sequentially, but it is not known if zero-knowledge is preserved if the repetition is performed in parallel. One drawback of sequential repetition is that it increases the round complexity of the protocol.[2]

[2] There is a different type of "repetition" that preserves zero-knowledge and adds only three messages regardless of the desired soundness error, assuming the cheating prover is computationally bounded.