COS 533: Advanced Cryptography — Princeton University
Lecture 9 (March 4, 2021)
Lecturer: Mark Zhandry — Scribe: Sam Breckenridge

Notes for Lecture 9

1 Introduction

In this lecture we finish our discussion of algebraic tools for Cryptography with a discussion of Integer Factorization as a hard problem, before introducing the idea of zero-knowledge proofs. We discuss what is meant by zero-knowledge and providee a zero-knowledge proof for 3-coloring.

1.1 Questions

Q: In the Goldreich-Levin theorem we have a binary alphabet and take the inner-product mod 2, does the Goldreich-Levin theorem also hold if we have a q-nary alphabet where q is prime and take the inner product mod q?

A: Sort of. If the value of q is polynomial, then it is known to generalize, however if the value of q is super-polynomial it turns out that it will not generalize. What will happen is in your expression for your final advantage in inverting the OWF you will see a loss that depends on q - when you move to q being super-polynomial the theorem still works but the loss will be super-polynomial which won't contradict the security of the OWF.

2 Integer Factorization

Let p, q be primes, we can compute $N = pq$ in time poly$(log(p), log(q))$ (polynomial in the bit-lengths of the inputs). On the other hand, we believe that computing p, q from N takes super-polynomial time. A simple "grade-school" algorithm for factoring is to try all factors up to $\sqrt{N}$, this will take time $\sqrt{N} = 2^{log(N)/2}$ which is better than $2^{log(N)}$ but not by much. We have sophisticated algorithms for factoring that run in time $2^{\sim log^{1/3}(N)}$, this is much better than trying all primes but is still super-polynomial. It is widely believed that factoring is in fact hard. Factoring being hard immediately gives us for example a OWF ($F(p, q) = pq$). However we can do more than this with integer factorization.

2.1 RSA

We can think of RSA as a public-key encryption scheme, even though it will not satisfy CPA-security on its own. We have:

$$\begin{aligned}\mathsf{Gen}() : \quad & p, q \leftarrow \text{Primes}[1] \\ & \text{Choose}[2]\ e \text{ such that } GCD(e, (p-1)(q-1)) = 1 \\ & \text{Compute}[3]\ d = e^{-1} \mod (p-1)(q-1)\end{aligned}$$

We then have a public key $\mathsf{pk} = (N, e)$ and a secret key $\mathsf{sk} = (N, d)$ where $N = pq$. The reason that we think this scheme might be secure is that the only known way to compute d from (N, e) is to factor N. If we know the factors of N we can compute $(p-1)(q-1)$ which allows us to invert e mod $(p-1)(q-1)$, if we don't know the factors of N we don't know how to compute this inverse, in particular we don't know how to compute $(p-1)(q-1)$.

Then to encrypt[4]:

$$\mathsf{Enc}(\mathsf{pk}, m \in \mathbb{Z}_N^*) = m^e \mod N$$

Technically we do not know $\mathbb{Z}_N^*$ as to describe it you need to exclude p, q and all their multiples, and as we do not know these we cannot exclude them. However p, q are big and their multiples are a sparse subset of the integers from 1 to N, this means that any m you choose will likely be in $\mathbb{Z}_N^*$. If a chosen m did happen to be a multiple, you could do a GCD computation to factor N, so the hardness of factoring implies that no efficient process of generating m will generate $m \notin \mathbb{Z}_N^*$.

To decrypt:

$$\mathsf{Dec}(\mathsf{sk}, c) = c^d \mod N$$

Correctness:

$$\begin{aligned}\mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, m)) &= \mathsf{Dec}(\mathsf{sk}, m^e \mod N) \\ &= m^{de} \mod N \\ &= m^{1+k(p-1)(q-1)} \mod N \quad (\text{Using } d = e^{-1} \mod (p-1)(q-1)) \\ &= m \cdot (m^{(p-1)(q-1)})^k \mod N \\ &= m \mod N\end{aligned}$$

[1] Here we assume we have the ability to sample large primes. There are some slight non-trivialities to this, but basically to sample a large prime you sample a large number and then do a primality test, resampling if the test fails.

[2] Typically $e = 3$ although it can be anything as long as the GCD requirement is satisfied

[3] The inverse exists and can be computed iff $GCD(e, (p-1)(q-1)) = 1$

[4] $\mathbb{Z}_N^* = \{\text{Integers } x \text{ from 1 to N such that } GCD(x, N) = 1\ \}$

The last step above uses $m^{(p-1)(q-1)} = 1 \mod N$. To show this we consider that $|\mathbb{Z}_N^*| = (p-1)(q-1)$. There are N elements in $\mathbb{Z}_N$, we subtract all the multiples of p (of which there are q), we subtract all the multiples of q (of which there are p), we have overcounted because $N = pq$ so we add 1 back, and factoring this we get $(p-1)(q-1)$. $\mathbb{Z}_N^*$ is a multiplicative group, so we know that $\forall m \in \mathbb{Z}_N^*, m^{|\mathbb{Z}_N^*|} = m^{(p-1)(q-1)} = 1 \mod N$. [5]

Caveat: RSA relies on factoring being hard, but we cannot base RSA on factoring being hard, so it is a stronger assumption than factoring integers.

2.2 Security of RSA

We note that RSA is not CPA-secure. We won't prove this, but one thing to notice is that RSA is not randomized, and we have seen that any CPA-secure encyrption scheme must be randomized. What we can conjecture about RSA is that it is one-way:

Conjecture 1 *(RSA assumption) Given* $(N, e, m^e \mod N), (N, e) \leftarrow \mathsf{Gen}(), m \leftarrow \mathbb{Z}_N^*$ *it is infeasible to compute* m

This conjecture does not say anything about CPA-security because it only considers the case where m is random. In the CPA-security setting m does not need to be random and could be chosen by the adversary from a small set, or the adversary could have some side information about the message. Therefore this is quite a weak notion of security for encryption schemes.

Note: We can upgrade to the indistinguishability required by CPA-security using Goldreich-Levin. Rather than directly encrypting the message, you encrypt a random string and extract a hardcore bit from this string, using this bit to mask the message.

RSA is often called a **Trapdoor Permutation**. The RSA assumption means that encryption with RSA is a one way permutation. It is a permutation because the fact that decryption is possible means that encryption is injective, and the domain and range are the same. It is one-way because it is easy to compute using modular arithmetic, yet is hard to invert under the RSA assumption. Yet we note that this permuation has a trapdoor that allows its one-wayness to be broken (in particular the value d).

2.3 Questions

Q: In RSA, doesn't knowing e give us some information about p, q?

[5] Using Euler's Theorem

A: e does in principle reveal some information about p, q but not very much. Let's say e is 3, requiring that p, q are not divisible by 3 is not revealing too much information. In addition, the only way we know how to factor N is not helped much by knowing whether p, q are divisible by 3.

3 Zero-knowledge Proofs

Intuition: A zero-knowledge proof is an interactive protocol between prover P and verifier V that is:

- **Sound:** V is convinced only if the statement is true
- **Zero-knowledge:** V learns nothing but the fact that the statement is true

This seems to be difficult to accomplish. Imagine trying to prove some NP statement, for instance P wants to prove to V that some SAT formula has a satisfying assignment. One way P can do this is by giving V the satisfying assignment (this would be sound, but would not satisfy zero-knowledge as V learns an assignment). With zero-knowledge, V must learn nothing beyond the fact that the statement has a verifying assignment.

The main conceptual challenge in defining a zero-knowledge proof is how to characterize what it means for V to learn nothing.

3.1 Soundness

Intuitively we define soundness as the property that given a statement x, if x is false, no cheating P can convince V that x is true. We consider a few different levels of soundness:

Definition 1 *(Perfect Soundness) For any computationally unbounded cheating prover P^*, the result of an interaction $P^* \rightleftarrows V$ is that V will reject with $Pr = 1$*

The canonical example of a perfectly sound proof system is using NP witnesses. If the statement x is that some SAT formula has a satisfying assignment but the formula does not in fact have a satisfying assignment, there is no assignment that can be proven using the protocol. Therefore a perfectly sound protocol is for P to supply a satisfying assignment. If the formula is unsatisfiable, there is no way for P to do so. Generally perfect soundness is too strong when combined with things like zero-knowledge, so with interactive protocols we generally relax to at least statistical soundess.

Definition 2 *(Statistical Soundness) For any computationally unbounded cheating prover P^*, the result of an interaction $P^* \overleftrightarrows V$ is that V will reject with $Pr = 1 - negligible$ (Pr is over random coins in protocol, not over x)*

Definition 3 *(Computational Soundness) For any PPT cheating prover P^*, the result of an interaction $P^* \overleftrightarrows V$ is that V will reject with $Pr = 1 - negligible$ (Pr is over random coins in protocol, not over x)*

3.2 Zero-knowledge

To define zero-knowledge we use a simulation based definition. Here View$(P(x) \overleftrightarrows V^*)$ is the transcript of the interaction:

Definition 4 *(Zero-knowledge) $\forall$ PPT cheating verifiers $V^*, \exists$ a PPT simulator S^* such that $\forall$ true x and corresponding interactions $P(x) \overleftrightarrows V^*$, View$(P(x) \overleftrightarrows V^*) \approx_c S^*(x)$*

We note that here we allow computationally unbounded provers, which is okay because soundess works against these. We typically allow this so that the prover can brute-force search for a solution if necessary. Sometimes the prover is given a witness, in which case we write $P(x, \pi)$ and the prover can be made efficient.

We interpret this definition as follows: The verifier can interact with the unbounded prover and even deviate from the protocol in order to learn more than simply the truthfulness of x. However no matter what they do, we can efficiently simulate the interaction between the prover and the verifier. The intution here is that V^* could have just run $S^*(x)$ (which outputs what looks like a transcript of the interaction) instead of interacting with P. Therefore anything V^* could have learned from interacting with P it could also have learned from looking at $S^*(x)$, meaning nothing is gained from interacting with P. The one thing they do learn is that the statement x is true, because S^* is only guaranteed to work for true x.

Remark: What about the private state of V^*? The interaction isn't necessarily the entirety of V^*'s view. Maybe V^* through this interaction stored something that is not revealed from the transcript, and after the interaction can use this side information to learn more.

To address this, we assume without loss of generality that the last message of the protocol is from V^* to P. We note that this has no effect on soundness - for soundness the prover is trying to convince the verifier, so there is no point in having the verifier sending a message to the prover. We have V^* always send its entire state in the last message. Clearly it does not want to do this in an earlier message because it may need to keep its state secret in order to maintain soundness, however once the prover sends

their final message, the soundness experiment is done. This final message means that $S^*(x)$ must also simulate the entire final state of V^*.

3.3 Zero-knowldge Proof for 3-coloring

We note that 3-coloring is NP-complete and therefore this protocol implies a zero-knowledge protocol for all of NP.

Let $G = (V, E)$ be a graph, X_g is the statement that there exists a valid 3-coloring:

$$X_G = \text{“}\exists \pi : V \to \{1, 2, 3\} \text{ such that } \forall (u, v) \in E, \pi(u) \neq \pi(v)\text{”}$$

We assume the prover knows a valid 3-coloring of the graph, their aim is to convince the verifier that a 3-coloring exists without revealing any of this coloring.

Idealized Protocol:

P has a coloring π. P chooses a random permuation $\sigma \leftarrow$ permutations($\{1,2,3\}$). We note that $\pi' = \sigma \circ \pi$ is also a valid 3-coloring - permuting the colors doesn't change the fact that two colors are different and so will not invalidate the 3-coloring. Now P constructs “locked boxes” $C_1, ..., C_n$ containing $\pi'(1), ..., \pi'(n)$ and sends these locked boxes to V. V replies with a random edge $(u, v) \leftarrow E$. P sends the keys for C_u, C_v. V now opens C_u, C_v and checks $\pi'(u) \neq \pi'(v)$.

Soundness

Suppose G is not 3-colorable. This means no matter what the contents of $C_1, ..., C_n$ are $\exists (u, v) \in E$ such that $\pi'(u) \neq \pi'(v)$. V will guess (u, v) with probability $\frac{1}{|E|}$ and the prover will be caught. Therefore, no cheating prover can win with probability $> \frac{1}{|E|}$

Zero-knowledge

The verifier sees n locked boxes, then gets to open 2 of them. Because the coloring is valid and a permutation was applied to the coloring, the values in the 2 boxes will just be random distinct colors. We define the following simulator:

$S(x)$: Choose a random coloring $\pi^* : V \to \{1, 2, 3\}$
Construct $C_1, ..., C_n$ containing $\pi^*(1), ..., \pi^*(n)$
Run V^* on $C_1, ..., C_n$
V^* will produce edge (u, v)
if $\pi^*(u) \neq \pi^*(v)$, sends keys to V^*
otherwise: restart/rewind to the beginning and try again

Each of these trials run by the simulator is independent. Conditioned on a trial succeeding, we generate something that looks exactly like V^* expects. Because the

colors will be distinct with $P = 2/3$ the expected number of trials before succeeding is 3/2.

Improving Soundness

We still have the problem of the soundness probability to address - we only guaranteed very weak soundness in which the prover gets caught with $\frac{1}{|E|}$ probability, whereas we would like the prover to be caught almost certainly. The solution to this is to run the protocol $\lambda|E|$ times, over this many times the probability that a cheating prover suceeds will be $(1 - \frac{1}{|E|})^{|E|\lambda} \approx e^{-\lambda}$ which is negligible.

We note that for zero-knowledge these rounds must be sequential, due to the rewinding of the simulator. If all the rounds are done in parallel, the chance of having $\pi^*(u) \neq \pi^*(v)$ in all rounds is quite low, and so the simulator will fail with high probability. On the other hand, if the rounds are done sequentially, on a round that fails the simulator simply rewinds that particular round.

3.4 Next Time

Next time we will replace our "locked boxes" with commitments.

3.5 Questions

Q: When we say a "cheating verifier", is it the case that the verifier still trying to discern whether x is true but just does not have to follow the protocol?

A: We don't know. Maybe the verifier doesn't even care about learning the truthfulness of x. Maybe they are willing to forgo the ability to learn the truthfulness of x in order to learn something else about x. However anything that the verifier tries to learn, they could have learned by simulating the transcript.

Q: What exactly is $S^*(x)$ simulating?

A: $S^*(x)$ outputs a purported transcript of interaction (what it claims to be the messages sent back and forth between P and V) although it does not interact with P to do so.

Q: In the 3-coloring do we assume the prover sticks to 3 colors? Couldn't the prover put a different color in each box

A: They could but we can have the verifier check that $\pi'(u), \pi'(v) \in \{1, 2, 3\}$, then if the prover tries to cheat they will trivially fail. Alternatively we imagine that the locked boxes only allow certain colors.

Q: Why is our zero-knowledge proof for 3-coloring only computationally sound and not statistically sound?

A: Right now it is actually statistically sound. There is no computation when using our locked box abstraction. When we instantiate this with a commitment scheme it will depend on the commitment. We will use a statistically binding commitment which ends up preserving the statistical soundness of the protocol. Zero-knowledge will be computational due to the hiding property of the locked boxes which will be a computational notion.

Q: When re-running the simulator do we re-run the entire procedure?

A: In the one-run version of the protocol you just re-run from the beginning until the response is one where the colors in the locked boxes happen to be different colors. When we repeat the protocol many times sequentially to increase soundness, we still rewind, but not to the very beginning. We just rewind to the beginning of the particular round before giving the verifier the locked boxes, keeping everything from previous rounds the same. The transcript will only be the messages from the successful rounds.

Q: Are we not able to run rounds in parallel because it allows the verifier to learn something?

A: It is unknown whether this protocol remains zero-knowledge in the parallel repetition setting, in general we know that there are protocols where it does not, and the verifier can learn something in the parallel setting. However this is an open question for this particular protocol.