COS 533: Advanced Cryptography — Princeton University
Lecture 10 (March 9, 2021)
Lecturer: Mark Zhandry — Scribe: Zhijun Zhang

Notes for Lecture 10

1 Introduction

Last time, we introduced zero-knowledge proofs, in which a prover P interacts with a verifier V and tries to convince V that a statememt X is true, without revealing anything other than the truthfulness of X. We constructed an idealized protocol for 3-coloring involving lockboxes. The basic protocol uses three messages:

1. P first samples uniformly at random a permutation σ of the 3 colors, and re-colors the vertices to get another 3-coloring $\pi' = \sigma \circ \pi$, where π is the original 3-coloring that P holds. Then P constructs n lockboxes $C_1, \ldots, C_n$ containing colors $\pi'(1), \ldots, \pi'(n)$, respectively, and send all of them to V.
2. V samples a random edge $(u, v) \leftarrow E(G)$ and asks P to provide the keys to C_u, C_v.
3. P sends the requested keys so that V can open C_u, C_v and check if $\pi'(u) \neq \pi'(v)$.

This basic protocol is repeated $\lambda|E(G)|$ times to achieve statistical soundness. Today we are going to replace the abstract lockboxes with commitment schemes.

2 Commitment

2.1 Definition

A commitment scheme can be thought of as the digital analog of a lockbox. It consists of two phases. The prover first "commits" to a value X in the commit phase. Then in the reveal phase, the prover "reveals" the value X, and gives a proof π, which is analogous to the key to the lockbox, that X is actually what was committed to. There are two properties that we want a commitment scheme to have.

Hiding Ideally, the prover wants the verifier to learn nothing about X in the commit phase. We can formally define "learn nothing" much the same way as we defined security of encryption schemes. Based on different levels of the hiding property

that we want, there can be different definitions: perfect hiding, statistical hiding, and computational hiding[1]. Note that the hiding property is a property against the verifier that is required for zero-knowledge.

Binding Informally, the binding property requires that the prover cannot change the value X after he has commited. In other words, if the prover commits to X, he cannot later open to $X' \neq X$, by providing X' and π' that looks like a valid proof. We can also define perfect, statistical, and computational binding formally. Note that the binding property is a property against the prover.

Remark 1 *Both the hiding and binding properties cannot simultaneously be statistical/perfect. Put it another way, one of the two properties has to always be computational. Intuitively, if the hiding property is perfect, it means the distribution of commitments is independent of X and thus any transcript of the commit phase must be able to be opened to something else. A computationally unbounded attacker would be able to accomplish it so the hope is that it's computationally infeasible. For our purpose, we want statistical binding because we have already assumed the verifier is computationally bounded. Also, in the definition of zero-knowledge proofs, ideally we want that even computationally unbounded prover should be incapable of convincing the verifier of a false statement, i.e. statistical soundness.*

Remark 2 *Someone might suggest that in some settings we can expect perfect and statistical binding to be the same. The intuition is as follows. If the attacker can sometimes open to another value, then there must exist a valid opening that is different. So the attacker can always find it so long as he is inefficient. This intuition turns out to be true for* non-interactive commitments*, where the commitment is literally a single message from the prover to the verifier. Perfect and statistical binding collapses to the same thing based on this intuition.*

However, for an interactive protocol, what it could be is that the verifier sends a message to the prover, and the prover commits to something based on that message. It could be that there are certain bad choices of the verifier's message that would actually allow the prover to cheat. But for most choices of the verifier's message the prover cannot. Here the probability is taken over the random choices of the verifier during the commit phase. Perfect binding insists that the attacker succeeds with probability zero while statistical binding allows for negligible probability.

[1]Perfect hiding basically means the commitment is perfectly independent of the value X. Statistical hiding roughly says that the distribution of what the verifier learns is statistically independent. For computational hiding, it is defined in a similar way to the security of encryption schemes. That is, the distributions of commitments of any two X's are computationally indistinguishable.

2.2 Construction

Assume a PRG $G : \{0,1\}^\lambda \to \{0,1\}^{3\lambda}$. We are going to commit to a single bit and we can extend to many bits just by parallel repetition.

Commit The verifier[2] sends $r \leftarrow \{0,1\}^{3\lambda}$ to the prover. The prover samples $s \leftarrow \{0,1\}^\lambda$, and sends either $G(s)$ or $G(s) \oplus r$ depending on the commitment bit b. That is, the prover sends $G(s)$ if $b = 0$ and otherwise sends $G(s) \oplus r$.

Reveal The prover sends b, s. The verifier can check easily.

The (computational) hiding property follows immediately from PRG security. Indeed, from the verifier's perspective, at the end of the commit phase, $G(s)$ is pseudorandom and indistinguishable from uniformly random bits. Then what the verifier sees is always random regardless of b.

To show the (statistical) binding property, observe that if the prover commits to b but opens to $1 - b$, this means he is able to find c, s, s' such that the commitment is $c = G(s) = G(s') \oplus r$. Ignoring c, the prover actually finds s, s' such that $G(s) \oplus G(s') = r$. However, such s, s' exist only with probability $\leq 1/2^\lambda$ over randomness of r. The proof follows from a union bound. Note that $|\{G(s) \oplus G(s') \mid s, s' \in \{0,1\}^\lambda\}| \leq |\{(s, s') \mid s, s' \in \{0,1\}^\lambda\}| \leq 2^{2\lambda}$. On the other hand, there are $2^{3\lambda}$ different choices of r. Therefore, the probability of $r \in \{G(s) \oplus G(s') \mid s, s' \in \{0,1\}^\lambda\}$ is bounded by $2^{2\lambda}/2^{3\lambda} = 1/2^\lambda$ as claimed.

Remark 3 *The construction above requires transmitting a large number of bits. Actually, if the goal is to get a commitment scheme from one-way functions or PRGs, essentially nothing better is known. This construction of commitment and also the zero-knowledge proof construction are totally impractical for any practical purposes. One possible thing that could be done is reusing r. But that is not a huge deal as this protocol is extremely inefficient. In practice, there are more efficient protocols, using stronger tools (e.g. algebraic tools).*

2.3 Commitment from Collision Resistant Hashing

A much better (non-interactive) commitment scheme works as follows. To commit to a message m, the prover appends some private random bits r, and then hash it with a collision resistant hash function H. The prover sends the hash $H(m, r)$ to the verifier. In the reveal phase, the prover simply reveals r to the verifier. The binding

[2] In the context of commitments, the prover is often called the sender and the verifier called the receiver. We will keep with prover and verifier since those are the parties used in zero-knowledge proofs.

property follows from collision resistance of the hash function. However, it generally is going to be only computational since if the hash function is shrinking, there will be collisions and thus other openings. It turns out that we cannot base the hiding property on collision resistance of the hash function. But with some tweaks we can actually get statistical hiding.

The reason why we don't like this scheme is that its binding property is only computational, which means the proof system we get is only computationally sound so an inefficient attacker would actually be able to cheat and convince the verifier of a false statement. The other reason is that we don't know how to achieve collision resistance from one-way functions. From a theoretical perspective, we want minimal assumptions. But again, in practice, we don't really care about inefficient attackers so this scheme is fine.

2.4 How Many Rounds are Necessary for ZK

It turns out that at least 3 rounds is necessary for ZK. Here we only sketch a proof that 1 round is not enough. Suppose for the purpose of contradiction that there exists a 1-round protocol. Soundness means if X is false, there exists no message π such that $V(X, \pi)$ accepts. On the other hand, zero-knowledge means we can simulate the view of V without knowing a witness. Since the view of V is exactly π, that means there exists an S such that if X is true, $S(X)$ outputs a π such that $V(X, \pi)$ accepts. It immediately follows that to decide the truthfulness of X, we can just run $V(X, S(X))$. If X is true, it has to accept. If X is false, it has to reject. Therefore, this shows that X is actually in the complexity class BPP[3], for which ZK is trivial as the prover does not need to do anything and the verifier can decide X on his own. With some additional efforts, we can turn this into a proof for two messages.

3 Non-interactive Zero-knowledge Proof (NIZK)

3.1 CRS Model

In non-interactive zero-knowledge proofs, there is a common random string $crs \leftarrow \{0,1\}^{q(\lambda)}$ known to both the prover and the verifier[4]. Then the prover sends a message π to the verifier. The protocol still has to satisfy soundness and zero-knowledge. In fact, soundness is going to be statistical soundness where the probability is taken over randomness of crs. For zero-knowledge, the simulator S needs to simulate both π and

[3]BPP stands for bounded-error probabilistic polynomial time, which is essentially the class of decision problems solvable in PPT with error probability $\leq 1/3$.

[4]In real world, crs could be generated by a trusted third party. It turns out in many (application-specific) cases, crs could also be generated by one of the parties.

crs such that crs looks random and π looks like a valid proof. Note that if S needs to simulate π for given crs, it is meaningless as the simulator has no extra power[5] over the prover and we can again conclude that X is in BPP.

3.2 Application: CCA Security for Encryption

In CCA[6] experiments, the attacker is given the ability to make decryption queries on everything except the ciphertext being attacked, in addition to encryption queries. CCA-secure PKE is a known challenge. One way that we can do this is upgrade a CPA-secure PKE plus a NIZK to get a CCA-secure PKE. An inaccurate description is as follows. To encrypt a message m, what $\text{Enc}_{CCA}(pk, m)$ does is encrypt under CPA-security under two public keys, i.e. $\text{Enc}_{CPA}(pk_0, m), \text{Enc}_{CPA}(pk_1, m)$. Suppose the reduction tries to break pk_0 and it uses pk_1 to answer CCA queries. That is, the adversary has the ability to decrypt under pk_1 but don't have the ability to decrypt under pk_0.

Now the problem is that the adversary can submit malformed ciphertexts such as $\text{Enc}_{CPA}(pk_0, m_0), \text{Enc}_{CPA}(pk_1, m_1)$. When the reduction tries to decrypt, it cannot answer correctly. This will break the reduction. The solution is to provide a NIZK π, as part of the ciphertext, that the ciphertext is well-formed. By the soundness of NIZK, if the proof verifies that the ciphertext is well-formed, we can now decrypt the ciphertext just by decrypting either component and we will get the same answer.

As to the question of who generates crs, in this case it is just $\text{Gen}(\cdot)$, the setup algorithm for the public key encryption scheme, in real life. The reason is that $\text{Gen}(\cdot)$ is always assumed to be honest. In the reduction, crs is generated by the simulator $S(\cdot)$. We don't go into the details of the reduction.

3.3 Hidden Bits Model

We are going to first construct NIZK in a fake model called hidden bits model, and then show later how to instantiate the hidden bits model using crypto techniques. The hidden bits model is similar to the CRS model but is asymmetric. Here only the prover knows $crs \in \{0,1\}^{q(\lambda)}$. Based on crs, the prover construsts π as well as a subset $L \subseteq [q(\lambda)]$. The verifier would get π and crs_L, i.e. the bits of crs specified by L. Note that the prover cannot cheat in the sense that the bits always come straight from the original crs. The prover can only choose the subset of bits that the verifier gets to see but does not have the control of the value of the bits.

To go from the hidden bits model to the CRS model, assume a one-way permutation

[5] In (interactive) zero-knowledge proofs as we previously defined, the extra power of the simulator is basically the ability to rewind and re-sample.

[6] CCA stands for chosen ciphertext attack.

$p : \{0,1\}^\lambda \to \{0,1\}^\lambda$ and an associated hardcore predicate $h : \{0,1\}^\lambda \to \{0,1\}$. The actual $crs \leftarrow \{0,1\}^{\lambda q(\lambda)}$ has $\lambda q(\lambda)$ bits. So for every bit of the hidden bits crs, we have $q(\lambda)$ bits in the real crs. We interpret our crs as tuples of λ-bit strings so it has $q(\lambda)$ blocks of λ bits. The prover simulates the hidden bits protocol as follows. Let $\alpha_i = p^{-1}(crs_i)$, where crs_i is the ith block of crs. The ith bit of the hidden bits crs is $crs_i^{HB} = h(\alpha_i)$. The prover runs $(\pi^{HB}, L) \leftarrow P^{HB}(x, crs^{HB})$ (either with or without a witness), and sends $\pi = (\pi^{HB}, L, crs_L^{HB}, \alpha_L)$ to the verifier. Then the verifier checks $crs_i^{HB} = h(\alpha_i)$ and $p(\alpha_i) = crs_i$, and simulates V^{HB}. For those bits not revealed by the prover, α_i is hidden by the one-wayness of p and then by the hardcore property the corresponding crs_i^{HB} is also hidden.

Note that the computation of α_i is the only inefficient part. To get an efficient prover, we can replace p with a trapdoor permutation. One candidate for trapdoor permutations is the RSA scheme we introduced previously.

4 Next Time

Next time, we will construct NIZK in the hidden bits model. The above procedure can translate it into the CRS model.