The Real/Ideal Paradigm: Lecture 2

Symmetric Encryption Scheme with Random and Pseudorandom Functions

Last time...

- Type key, text, and cypher (scheme specific)
- Scheme is something which implements the 3 functions `key_gen`, `enc`, `dec`, and is stateless
- IND-CPA security is the Encryption Oracle game from last time.
  - Take an encryption scheme, plug it into the game, and it will resolve the game where an Adversary needs to be able to use the encryption oracle a limited number of times in order to distinguish between two cyphertexts with greater than 50% probability when given the plaintext of one of them.
  - Adversary has access to the encryption oracle, it chooses a pair of plaintexts and then guesses which of the cyphertexts was encrypted (can call `encpost` at this time)
  - Parameterized by scheme and Adversary

To prove IND-CPA Security...

- Adversary only wins the Encryption Oracle game with a bounded probability (often 50% plus a negligible amount)
  - Can also be a function of the number of bits in the plaintext and the limit on number of calls to the oracle.

Pseudorandom Functions

- Takes in a key, then determines a function from text to text.

$$F : \text{key} \to \text{text} \to \text{text}$$

- We determine whether a PRF is good by comparing it to a Random Function Module (the "ideal" version of randomness).

- If PRF ("real") and RF("ideal") are indistinguishable to an Adversary, then you can treat the PRF as though it is an RF.
- The Random Function Adversary can access our candidate random function and determine whether it is interacting with the PRF (return False) or the ideal RF (return True).
- Our PRF is "good" if the Random Function Adversary is nearly equally likely to claim that it is interacting with the RF or the PRF.

Our Symmetric Encryption Scheme

- Construct `Enc` out of F
- Type `text` $\rightarrow$ `text` $\rightarrow$ `text`
  - Takes random text value and returns the pair of u and the text made from XORing it with a function on our key

Our Adversarial Attack Strategy on our New Scheme

- When asked to pick a pair of texts calling `encpre`, you can call it repeatedly on the all-0 bit string.
  - If we do this, we can get a copy of the key.
  - Since we can only call a number of times up to the limit, we are somewhat limited. But that's still a lot of information!
- When the scheme does the encryption, we might see the string that came up in our experimentation prior to that phase of the game. Therefore, this significantly increases the odds that the Adversary can win the game.
  - That's bad! Unless you're the Adversary. :)
- The Adversary could also experiment with the F function to just directly try to determine I/O behavior. Observing patterns in the output string can reveal the key, as again, the XOR will let some data fall through every time.
  - It's not quite the same as the first strategy, but it's similar!

IND-CPA Security for our Scheme

- Upper bound will be a function of the ability of the Adversary to tell the PRF and RF apart, plus the limit on the number of calls
- Why don't we involve key length?
  - The Adversary's ability to determine the difference between PRF and RF is the only part that matters. This is encompassed in that.

Sequence of Games Approach

- Each game is parameterized by the adversary, and does a procedure producing a boolean.
  - Again, we want the upper bound on whether the game outputs True to be as close to 50% as possible (or for the difference in probabilities between the True and False result to be as small as possible).
- Basically, this lets you sum the bounds from Real to Ideal games.
  - Each step has a penalty and we sum them all.
- Why does this work?
  - The triangular inequality (check the slides :D)
- What is our Real Game vs. Ideal Game?
  - The adversary has to win exactly half the time in the Ideal Game. If the Adversary does not do much better in the Real game, then we can treat our Real scheme as equivalent, in some sense to the Ideal scheme.

But what *is* an Ideal Game?

- Start with the real game and make some simplifications, stepping more in the ideal direction.
  - Sometimes involves code rewriting. No behavior changes, so no cost for doing so.
  - Some use cryptographic reductions (e.g., to the security of PRFs).
  - "Up to bad reasoning" can be used. The transition from one bit of code to the other is secure, unless something Bad happens. For example, we bound the probability of the Bad event happening, and as long as we know it's limited, we can just assume the Bad Thing doesn't happen.
    - Imagine, like in the XOR example, that we get a random key of all 0s. That would be very bad, because the ciphertext is just the plaintext! But it's very unlikely, so we assume it's not a real possibility.

Starting the Proof in a Section

- Adversary's Choose and Guess procedure do have to terminate with probability 1. So do the `encpre` and `encpost`.

Step 1: Replace PRF with TRF (true random function)

- This lets us get a baseline for the rest of the proof.
- We have an exact model of how the TRF works.
- We inline the scheme into a new oracle parameterized by a random function.
  - The oracle detects two kinds of "clashes" (check the slides for more info about this)
  - This oracle isn't parameterized by an encryption scheme like before, but instead by a random function.
  - The game basically has to be a total coinflip, unless there is a clash with the u's output in the pre-game.
    - e.g., if the Adversary just so happens to have checked, by pure coincidence, a random key u which the game is now going to use, it will know how to decrypt the outputs. That is the "Bad Thing" which we can choose to accept a small probability of occurring.
- This is easy to prove no penalty for, since it's truly random. The Adversary can't get any advantage from playing this game.
  - I did not understand the probabilities part after this :(
  - Tl;dr the game is about the Adversary not being able to tell the PRF and the TRF apart. We replaced the PRF with the TRF, and want the Adversary to be unable to determine which it is looking at (with more than coinflip probability).
- Big picture, we are comparing the results of different games parameterized by an Adversary.

Step 2: Oblivious Update in `genc`

- We use "up to bad reasoning".
- We throw away `inps_pre` and just use the domain of the TRF.
  - No more need for random functions as a parameter. We are now just using TRF.
- Key step is that when we do the game's encryption, generate a random v, send u to v, and ignore the fact that u may already be in the map's domain.
  - If the u has come up before, that's the "Bad Thing" that "up to bad reasoning" allows us to discard the possibility of.

  - This means that our game is only valid if the clash doesn't happen...

- What does this get us?
  - When we run the old game and the new one, either a clash has happened in both games, or in neither game. If the clash didn't happen in either, then the results end up being the same (probabilistically speaking)
  - If a clash did happen, then we don't know anything at all :(.
  - We can bound the chance of a clash by $2^{\text{text\_len}}$ (the odds of accidentally generating the same u twice).

Step 3: Independent Choice in `genc`

- We don't need `clash_pre` anymore!
- We remember u, generate a random v, XOR with x, but we don't update the map at all.
  - Problem is that in `enc_post`, if the u picked is the same used and saved before, we have a divergence again. Moreover, since we aren't updating the map, the map on `genc_imp` between the two games will always be different. Unless the clash happens, though, we will never notice.
  - Basically, we pick our random v, but only use it once! This is important for the next step.
- This is another "up to bad" step. We can prove whether the clash happens 2x, 1x, or 0x. If a clash doesn't happen, then we can use the optimization without issue.
- Subtly, to do this, this relies on the fact that the probability of causing a clash is $1/2^{\text{text\_len}}$
  - Use the FEL to bound the probability of this clash happening.
- We have our final bound, but we aren't finished yet.

Part 4: The One-time Pad Argument

- Here, we don't need instrumentation for detecting clashes.
- What happened to `genc`?
  - Change a step. Now we just put v in the return value, not even referring to the x we are encrypting. So the right side of the cyphertext is totally random.
- There's no cost to this! Cool!

  - This is our Ideal Game. If we had a perfect scheme, this would be the probability of success.
  - Basically, you can prove that this is equivalent to a One-time Pad, which is always secure for, well, one use (ie it's a one-time use scheme).

- `rnd` tactic can solve this for us. Cool!

Step 5: Proving the probability of G4

- You can view this proof via the slides! It's mostly code.

The IND-CPA Security Result

- How small is our upper bound?
  - By making some assumptions about the PRF and the capabilities of the Adversary, we can settle on whether we get a bound we are happy with or not.
- We need to restrict the Adversary somewhat, or else it could mess with the global state, which would make it hard to make the game work.
  - e.g. the Adversary could cheat and just never return.

Why did we do the games in this order?

- We switched to true randomness at the end to make the One-Time Pad Argument. The steps before that were needed to switch from the PRF to the truly random function needed to construct our Pad.

Example 2: Private Count Retrieval

- Three-party protocol called Private Count Retrieval
  - "Honest but curious" security against the three parties.
  - Can't change how the protocol works, but Adversary wants to learn more than it should. We will prove that it can't.
  - We don't trust any of the 3 parties.
  - We have a client, server, DB, and the DB can be queried.
  - See the slides for an example!
- Informally...
  - The client only learns the count for its queries
  - Server learns nothing about the queries made by the client.
  - The third party learns nothing about DB and queries but certain patterns

Hashing!

- Protocol uses it.
- We want the probability of a clash here to be negligible.
  - We can use SHA-3 for this, but in this pure world, we can instead use a random oracle. The Adversary also has access to this oracle, unlike in the symmetric encryption world.
- Diagram and walkthrough of the PCR Protocol are on the slides. That's helpful for understanding!

Next time...

- Review
- See EasyCrypt formalization
- We'll talk about leakage between the ideal game and the simulator and figure out how much the Adversary could possibly learn.