1 Plan for today's lecture

Over the last few lectures, we have been looking at encryption schemes that are secure against 'read-only' adversries. We will start today's lecture with a quick summary of semantically secure encryption schemes. After the summary, we will discuss 'message integrity' - a useful stepping stone towards achieving encryption schemes secure against tampering attacks.

2 A summary of semantic security

- **Perfect secrecy:** We started the course with the notion of 'perfect secrecy'. We can define perfect secrecy in terms of the No-Query-Semantic-Security game (the challenger picks a key k and a bit b, the adversary sends two messages m_0, m_1, receives an encryption of m_b using key k, and must guess bit b). We say that an encryption scheme is perfectly secure if **no** adversary can win this game with probability greater than 1/2.
  - We saw that Shannon's one-time pad satisfies this definition.
  - There are a couple of issues with this Shannon's OTP scheme:
    1. the key size must be as large as the message size.
    2. while the scheme satisfies No-Query-Semantic-Security, it is insecure if the adversary has access to two or more encryptions using the same key.
- **Computational security:** Let us focus on the first limitation. The key-size issue is not specific to Shannon's OTP, it applies to any encryption scheme that achieves perfect security. Therefore, we must relax the security definition.
  - A first natural relaxation: allow the adversary to win with probability slightly greater than 1/2. That is, suppose we are fine with encryption schemes where **no** adversary can win the No-Query-Semantic-Security security game with probability greater than $1/2 + \epsilon$ for some small quantity ϵ (say $\epsilon = 0.01$). Unfortunately, the first limitation still applies — if we want to support messages of length ℓ, then the key length must be at least $\ell \cdot$(some function of ϵ).
  - The next natural relaxation is to restrict our attention to **efficient** adversaries — algorithms that run in polynomial time. We started with security against efficient adversaries that see only one ciphertext.

    An encryption scheme satisfies No-Query-Semantic-Security if for any efficient adversary, the probability of winning the No-Query-Semantic-Security game is at most 1/2+negligible.
- **Encryption schemes with short keys that achieve No-Query-Semantic-Security:** As a first step, we wanted to build an encryption scheme that has short keys (of length n), handles long messages (of length $\ell > n$) and offers security against adversaries that see only one ciphertext. Unlike Shannon's OTP where we could give an unconditional proof of security, here we will need to make some assumptions (this is necessary, otherwise we run into the P vs NP question). We will assume the existence of a few basic building blocks, and build all cryptography assuming these building blocks.
- **A fundamental building block: pseudorandom functions/permutations:** One of the fundamental building blocks in cryptography is the primitive called pseudorandom function/permutation (PRF/PRP). A PRF is a deterministic keyed function $F : \mathcal{X} \times \mathcal{K} \to \mathcal{Y}$ such that, for a random key k, the function $F(\cdot, k)$ is indistinguishable from a uniformly random function $f : \mathcal{X} \to \mathcal{Y}$. Note that since F is deterministic, the evaluation on the same input will produce the same output. For this summary, we will fix $\mathcal{X} = \mathcal{Y} = \mathcal{K} = \{0,1\}^n$, although this need not be the case always.

  Formally, the function F is a secure PRF if no adversary can distinguish between the following two scenarios: in the first world, the challenger chooses a random PRF key k, and for each query x, it sends $F(x, k)$. In the second world, the challenger chooses a uniformly random function $f(\cdot)$, and for each query x, it send $f(x)$. The adversary can make **polynomially many, adaptive** queries, and must finally guess whether it is interacting in the first world or the second world.

Similarly, a PRP is a keyed function $F : \mathcal{X} \times \mathcal{K} \to \mathcal{X}$, together with an inverse function $F^{-1} : \mathcal{X} \times \mathcal{K} \to \mathcal{X}$ such that for each key k and input x, $F^{-1}(F(x, k), k) = x$, and for a uniformly random key k, the permutation $F(\cdot, k)$ looks like a uniformly random permutation $f : \mathcal{X} \to \mathcal{X}$.

Why are PRFs/PRPs (as defined above) a fundamental building block?

There are a few natural ways to strengthen/weaken the above definition. Why is this one the 'sweet spot'? From a theoretical point of view, this definition is strong enough for a lot of applications, and yet can be achieved from one-wy functions (OWFs). As a result, if any sort of cryptography exists, then OWFs exist, and therefore PRFs exist.

- **Encryption scheme with long message, satisfying** No-Query-Semantic-Security**, from PRFs:** In class, we saw how to build such encryption schemes from PRGs. Since PRGs can be built from PRFs, we can build such encryption schemes from PRFs too. The construction is pretty simple: the encryption key is an n-bit PRF key k. To encrypt a message m of length $\ell \cdot n$, decompose m into ℓ chunks $m_1, m_2, \ldots, m_\ell$ and output $(m_1 \oplus F(1, k), m_2 \oplus F(2, k), \ldots, m_\ell \oplus F(\ell, k))$. Security stems from the observation that $(F(1, k), \ldots, F(\ell, k))$ look like ℓ uniformly random strings $(r_1, \ldots, r_\ell)$.

- **Encryption scheme with unbounded length message, satisfying** No-Query-Semantic-Security**, from PRFs:** The same construction described above can be extended to handle unbounded length messages too! This is the deterministic counter mode of encryption. There are also other modes of encryption (as discussed in Lecture 11).

  However, note that we don't have a direct transformation from encryption schemes with bounded length messages to encryption schemes with unbounded length messages. In particular, suppose $\mathcal{E}_1 = (\mathsf{KeyGen}_1, \mathsf{Enc}_1, \mathsf{Dec}_1)$ supports messages of length n. A natural idea for encrypting unbounded length messages is to break the message into chunks of size n, then encrypt each one separately using $\mathcal{E}_1$.

  This approach results in a broken encryption scheme!

- Semantic Security **(security against all efficient 'read-only' adversaries):** Finally, we defined security against ALL read-only adversaries. In this security game, the challenger picks a uniformly random key k and a bit b. The adversary gets to make polynomially many, adaptive queries. Each query consists of a pair of messages $(m_{i,0}, m_{i,1})$, and the challenger sends $\mathsf{Enc}(m_{i,b}, k)$. Finally, after all the queries, the adversary sends its guess b'.

  Again, you should ask yourself: if adaptivity was removed, do we get the same definition, or something strictly weaker? Similarly, if the number of queries was restricted, do we get the same definition, or something weaker? (This is not part of the syllabus, but if you have any thoughts/queries related to this question, I am happy to discuss after class/during office hours.)

- **Encryption scheme satisfying** Semantic Security**, from PRFs:** First, we observed that encryption schemes satisfying semantic security **must be randomized**. Next, we saw a construction achieving this notion, based on PRFs. To encrypt a message m, one chooses a random x and outputs $(x, m \oplus F(x, k))$ as the ciphertext. Security relies on two things here. First, it naturally depends on the PRF being secure. Additionally, we also use the fact that the domain of the PRF is much larger than the number of queries made by the adversary.

- **Encryption schemes with unbounded length messages, satisfying** Semantic Security**, from PRFs:** Finally, we would like our semantically secure encryption scheme to handle unbounded length messages. This also can be constructed from PRFs. Here are a couple of ways of doing it.

  – Let $\mathcal{E}_1$ be a semantically secure encryption scheme that handles messages of length n. To encrypt a message of length $n \cdot t$, break the message into t chunks of size n each, and encrypt each chunk separately using $\mathcal{E}_1$. If the message length is not a multiple of n, pad it with some special symbol so that the resulting message has length $n \cdot t$.

Prove that this construction is semantically secure.

This construction requires a fresh random string for each chunk. There are direct constructions that are more 'randomness-efficient'.

– We know that there exist PRF-based encryption schemes that handle unbounded length messages and achieve No-Query-Semantic-Security. We can use such an encryption scheme $\mathcal{E}_1 = (\mathsf{Enc}_1, \mathsf{Dec}_1)$, together with a PRF F as follows: to encrypt a message m, pick a random x, compute $k' = F(x, k)$ and output $(x, \mathsf{Enc}_1(m, k'))$.

3 Message Integrity

We will now move on to encryption schemes secure against stronger adversaries that can potentially tamper with the ciphertext. Suppose Alice sends a ciphertext to Bob. Intuitively, Bob expects the following:

- (Confidentiality) First, no adversary should learn the contents underlying the ciphertext. This is ensured via semantic security.
- (Integrity) Second, Bob should be convinced that it was indeed Alice who sent the ciphertext.

Integrity is a fundamental security property can be discussed in other scenarios as well. Consider the following: I would like to announce that the next quiz (Quiz 3) is on Tuesday, 20th September. There is no secrecy in the content of the announcement. However, I want to ensure that everyone gets the 'correct' announcement.

For this, I will send a separate 'message' to each student, containing the date of the quiz, together with some 'extra data'. The purpose of the 'extra data' is to ensure that no adversary can tamper with the announcement.

With the above motivation, we define syntax for a new cryptographic primitive — **message authentication codes** (or, as I prefer to call it, symmetric key signature schemes). In this primitive, there are two keyed functions: Sign and Verify. The signing algorithm takes as input a message m, a secret key k and outputs a 'signature' σ. The verification algorithm takes as input a message, a signature and a key, and outputs either 1 (indicating that σ is a valid signature) or 0 (indicating that σ is not a valid signature). Intuitively, no adversary should be able to produce a new message together with a valid signature, even after seeing many different (message, signature) pairs.

Sign should be keyed function: The first thing to note is that the signing algorithm must be a keyed algorithm. If Sign is a public algorithm that has no key, then the adversary can take any new announcement, and compute the signature. The verification algorithm will also need something that depends on the signing key. For now, we will insist on the verification key being same as signing key. In the second half of this course, we will make the verification key *public*.

We will define the syntax and security definitions shortly. First, let us check if the above suffices for the 'quiz announcement' application. I will share a separate key k_i with each student. Whenenver I need to put out an announcement, I will sign the announcement with key k_i, and send the announcement together with the signature to the i^{th} student. The student can verify the signature using k_i. Note that this is highly inefficient: I will need to produce a different signature for each student, even though the announcement is same for everyone. In the second half of the semester, we will see how to improve this (by introducing a 'public key' version of this). However, the symmetric key version will suffice for many other applications (including encryption schemes secure against active adversaries).

3.1 Security game(s)

As with any other cryptographic primitive, there can be multiple ways of defining security here.

Intuition for the definition(s): Let us go back to the toy scenario of quiz announcement. The adversary gets to see a few such announcements. The most basic requirement is that the adversary, even after seeing a few announcements, should not be able to come up with a **new** announcement. However, here are a few other ways to potentially give the adversary more power:

- the adversary can try to generate some (announcement, signature), send it to the student, and see his/her reaction.
- the adversary can try to generate a new signature on an old announcement. This may not be an attack in our application, but there can be other scenarios where generating a new signature on an old announcement is an attack to take care of.

MAC-Game-1

1. **Setup:** Challenger chooses a signing key k.
2. **Signature queries:** Adversary sends polynomially many signing queries (adaptively). The i^{th} query is a message m_i. The challenger sends $\sigma_i = \mathsf{Sign}(m_i, k)$ to the adversary.

   Note that the key k was chosen during setup, and the same key is used for all queries.
3. **Forgery:** The adversary sends a message m^* together with a signature σ^*, and wins if m^* is not one of the queried messages, and $\mathsf{Verify}(m^*, \sigma^*, k) = 1$.

Figure 1: Security Game for MACs: Game 1

Note that the "m^* is not one of the queried messages" is to prevent trivial adversaries that query for a message, receive the corresponding signature, and finally sends the same (message, signature) pair.

One of the questions asked in class: should the adversary be allowed verification queries as well? Consider the 'quiz announcement' application. Maybe the adversary sends fake announcements with fake signatures and observes the student's behaviour. This motivates the following security game (described in Figure 2).

MAC-Game-2

1. **Setup:** Challenger chooses a signing key k.
2. **Queries:** Adversary can send either signing or verification queries adaptively.
   - **Signing queries:** The adversary sends message m_i and receives $\sigma_i = \mathsf{Sign}(m_i, k)$ from the challenger.
   - **Verification queries:** The adversary sends a message m'_i, a signature σ'_i, and receives $\mathsf{Verify}(m'_i, \sigma'_i, k)$ from the challenger.

   Note that the key k was chosen during setup, and the same key is used for all queries. Also, the signing and verification queries can be interleaved.
3. **Forgery:** The adversary sends a message m^* together with a signature σ^*, and wins if m^* is not one of the queried messages, and $\mathsf{Verify}(m^*, \sigma^*, k) = 1$.

Figure 2: Security Game for MACs: Game 2

Finally, we consider scenarios where even submitting a **new** signature on an **old** message counts as a forgery. This is captured in the following security game (defined in Figure 3.

These are three security games. Next class, we will complete the security definition(s), and study the relation between these definitions.

MAC-Game-3

1. **Setup:** Challenger chooses a signing key k.
2. **Queries:** Adversary can send either signing or verification queries adaptively.
   - **Signing queries:** The adversary sends message m_i and receives $\sigma_i = \mathsf{Sign}(m_i, k)$ from the challenger.
   - **Verification queries:** The adversary sends a message m'_i, a signature σ'_i, and receives $\mathsf{Verify}(m'_i, \sigma'_i, k)$ from the challenger.

   Note that the key k was chosen during setup, and the same key is used for all queries. Also, the signing and verification queries can be interleaved.
3. **Forgery:** The adversary sends a message m^* together with a signature σ^*, and wins if $(m^*, \sigma^*) \neq (m_i, \sigma_i)$ for all i, and $\mathsf{Verify}(m^*, \sigma^*, k) = 1$.

Figure 3: Security Game for MACs: Game 3

4 MAC: Construction attempt

We ended the class with a candidate MAC construction using a semantically secure encryption scheme.

- The signing/verification key is the key for an encryption scheme.
- $\mathsf{Sign}(m, k)$: To sign a message m, compute $\mathsf{Enc}(m, k)$.
- $\mathsf{Verify}(m, \sigma, k)$: The verification algorithm computes $y = \mathsf{Dec}(\sigma, k)$, and outputs 1 iff y is equal to m.

This construction satisfies the correctness requirement, since a signature for message m will successfully pass the verification. But does it satisfy security?

Suppose we consider the PRF based construction of semantically secure encryption, and use it in the above construction. The signature on a message m is $\sigma = (x, F(x, k) \oplus m)$. Given a signature $\sigma = (\sigma_1, \sigma_2)$ on message m, an adversary can construct a new signature on a new message. Let r be any non-zero string. Note that $\sigma' = (\sigma_1, \sigma_2 \oplus r)$ is a valid signature on $m \oplus r$.

5 Lecture summary, plan for next lecture, additional resources

Summary: In this lecture, we started with a summary of semantic security for encryption schemes, and then started discussing message integrity, and a cryptographic primitive (message authentication codes) used to achieve message integrity.

Next lecture: We will look at MACs in more detail. First, we will see that verification queries don't give the adversary any additional power. Next, we will see a simple construction, based on a building block that we've already seen in this course.

Relevant sections from textbook [Boneh-Shoup]: The textbook contains a nice introduction to MACs (see first two pages of Section 6).