1 Last lecture, and plan for today's lecture

In the last few lectures, we studied public key encryption, and saw various constructions for the same. We discussed the 'gold-standard' security notion for public key encryption (CCA security) and even saw a construction that's secure in the random oracle model. However, this still does not solve one fundamental problem with public key encryption : suppose Bob wants to send an encrypted message to Alice, how does Bob get Alice's public key (assuming they haven't met before)? We can imagine there is a secure central database where Alice's public key $\mathsf{pk}_{\text{Alice}}$ is stored. When Bob requests the database for Alice's public key, and the database sends $\mathsf{pk}_{\text{Alice}}$, an adversary can intercept this communication, and replace $\mathsf{pk}_{\text{Alice}}$ with pk_{Adv}. This completely compromises security, as the adversary has the secret key corresponding to pk_{Adv}, and can therefore read Bob's message. To handle this, we require a different cryptographic primitive - digital signatures. Digital signatures have immense applications in ensuring authenticity of digital communications.

2 Digital Signatures: Definitions

A digital signature scheme consists of three algorithms: $\mathsf{KeyGen}, \mathsf{Sign}$ and Verify. These algorithms have the following syntax:

- $\mathsf{KeyGen}(1^n)$: The key generation algorithm takes as input the security parameter n. It outputs a signing key sk and a verification key vk. In practice, the signing key sk must be kept secret, while the verification key vk is public.
- $\mathsf{Sign}(m, \mathsf{sk})$: The signing algorithm takes as input the message m, the signing key sk and outputs a signature σ.
- $\mathsf{Verify}(m, \sigma, \mathsf{vk})$: The verification algorithm takes as input the message m, signature σ and verification key vk. It outputs either 1 (indicating that σ is a valid signature for m) or 0 (indicating that σ is not a valid signature for m).

For correctness, we require that if $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(1^n)$, $\sigma \leftarrow \mathsf{Sign}(m, \mathsf{sk})$, then $\mathsf{Verify}(m, \sigma, \mathsf{vk}) = 1$.

2.1 Security definitions for signature schemes

The security definition(s) are similar to the security definitions we had for MACs. An adversary can query for signatures on messages of its choice, and must finally output a new valid signature. If this new signature is on a new message, then we say that the adversary breaks weak unforgeability. If this new signature is on one of the queried messages, we say that the adversary breaks strong unforgeability. The formal security game is given below.

Unforgeability under Chosen Message Attack

- (Setup) Challenger chooses $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(1^n)$ and sends vk to the adversary.
- (Signature Queries) The adversary is allowed polynomially many signature queries. For every query m_i, the challenger sends $\sigma_i \leftarrow \mathsf{Sign}(m_i, \mathsf{sk})$.
- (Forgery) Finally, the adversary sends a message m^* and signature σ^*. The adversary breaks **weak unforgeability** if $m^* \neq m_i$ for all i, and $\mathsf{Verify}(m^*, \sigma^*, \mathsf{vk}) = 1$. The adversary breaks **strong unforgeability** if $(m^*, \sigma^*) \neq (m_i, \sigma_i)$ for all i, and $\mathsf{Verify}(m^*, \sigma^*, \mathsf{vk}) = 1$.

Figure 1: Unforgeability of Signature Schemes

Note: The adversary is allowed to repeat signing queries. Next, note that weak unforgeability guarantees strong unforgeability only if, for every message m, there is a unique signature that verifies. If the signature scheme is deterministic (but does not satisfy this uniqueness property), then we cannot conlude that weak unforgeability implies strong unforgeability.

Definition 35.01. A signature scheme $\mathcal{S} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ satisfies weak (resp. strong) unforgeability if for every p.p.t. adversary $\mathcal{A}$, there exists a negligible function $\mu(\cdot)$ such that for all n,

$$\Pr\left[\mathcal{A} \text{ wins the weak (resp. strong) unforgeability game against } \mathcal{S}\right] \leq \mu(n).$$

3 A few candidates discussed in class

In this section, we discuss a few candidate signature schemes discussed in class.

3.1 Textbook RSA signature scheme

- $\mathsf{KeyGen}(1^n)$: The key generation scheme chooses two n bit primes p and q. It sets $N = p \cdot q$. Next, it chooses an exponent $e \in \mathbb{Z}_{\phi(N)}$ such that $\gcd(e, \phi(N)) = 1$, together with $d \in \mathbb{Z}_{\phi(N)}$ such that $e \cdot d = 1 \bmod \phi(N)$. It sets the signing key $\mathsf{sk} = (N, d)$, and the verification key $\mathsf{vk} = (N, e)$.
- $\mathsf{Sign}(m \in \mathbb{Z}_N, \mathsf{sk} = (N, d))$: The signing algorithm outputs $\sigma = m^d \bmod N$.
- $\mathsf{Verify}(m, \sigma, \mathsf{vk} = (N, e))$: The verification algorithm outputs 1 iff $\sigma^e \bmod N = m$.

First, note that the scheme satisfies correctness. It also satisfies the unique signatures property. However, the scheme is not secure. Given a signature on message m_1 and m_2, one can compute a signature on $m_1 \cdot m_2$.

3.2 Textbook RSA, with a pseudorandom permutation

This candidate was proposed in Lecture 36. I'm including it here since it helps build familiarity with the unforgeability definition. Let $F : \mathbb{Z}_N \times \mathbb{Z}_{\phi(N)} \to \mathbb{Z}_N$ be a pseudorandom permutation.

- $\mathsf{KeyGen}(1^n)$: The key generation scheme chooses two n bit primes p and q. It sets $N = p \cdot q$. Next, it chooses an exponent $e \in \mathbb{Z}_{\phi(N)}$ such that $\gcd(e, \phi(N)) = 1$, together with $d \in \mathbb{Z}_{\phi(N)}$ such that $e \cdot d = 1 \bmod \phi(N)$. It sets the signing key $\mathsf{sk} = (N, d, e)$, and the verification key $\mathsf{vk} = (N, e)$.
- $\mathsf{Sign}(m \in \mathbb{Z}_N, \mathsf{sk} = (N, d, e))$: The signing algorithm first computes $y_1 = m^d \bmod N$. Next, it computes $y_2 = F(y_1, e)$. Finally it outputs $\sigma = y_2^d \bmod N$.
- $\mathsf{Verify}(m, \sigma, \mathsf{vk} = (N, e))$: The verification algorithm first computes $y_2 = \sigma^e \bmod N$. Next, it computes $y_1 = F^{-1}(y_1, e)$. Finally, it outputs 1 iff $m = y_1^e \bmod N$.

While this candidate looks complicated, it should be clear that we will not be able to prove security here, since PRP's security guarantee holds only if the key is hidden. In this case, the key is included in the verification key, and this should be a red flag. In fact, there is an attack that doesn't even require any queries! The adversary simply picks a random number σ in $\mathbb{Z}_N$. This will be the signature corresponding to some message, and the adversary finds the corresponding message as follows:

1. Compute $y_2 = \sigma^e$.
2. Compute $y_1 = F^{-1}(y_1, e)$.
3. Compute $m = y_1^e$.

Output (m, σ) as the forgery. Check that σ is indeed a valid signature on m.

We will see how to 'fix' the RSA based construction to make it secure in the random oracle model. (We will also see, in Lecture 37, how to use RSA to get a construction secure in the standard model).

4 Expanding the message space of a signature scheme

We would like to sign large documents. Suppose you are given a signature scheme that supports n-bit messages. To sign longer (unbounded length) messages, we will need to first compress the message, and then sign the digest. A natural candidate that can be used here (and also gives provable security guarantees) is a collision resistant hash function H. The new key generation algorithm first chooses $(\mathsf{sk}', \mathsf{vk}')$ for the signing scheme, then chooses a hash key k, and sets $\mathsf{sk} = (k, \mathsf{sk}')$, $\mathsf{vk} = (k, \mathsf{vk}')$.

To sign a message m, first compute $h = H_k(m)$, then output $\sigma = \mathsf{Sign}(h, \mathsf{sk})$. The verification algorithm computes $h = H_k(m)$, then checks if $\mathsf{Verify}(h, \sigma, \mathsf{vk}') = 1$. 'Hash-and-sign' was also used in the MAC setting, and we proved security assuming H is collision resistant and the base MAC scheme was secure. The same proof also works here. Note that in the MAC setting, we could have used a UHF instead of a CRHF. Here, it is necessary to use a CRHF. (Why?).

This is because you don't want the adversary to produce a collision and output a valid forgery.

A natural question is whether CRHFs are necessary here. In class, the following candidate was proposed. Let us assume $\mathcal{S} = (\mathsf{KeyGen}', \mathsf{Sign}', \mathsf{Verify}')$ is a secure signature scheme where the message space is $\{0,1\}^{2n}$, and the signature space is $\{0,1\}^n$ (that is, each message is a $2n$ bit string, and each signature is an n bit string). Consider the following signing scheme that handles $\mathcal{M} = (\{0,1\}^n)^*$.

- $\mathsf{KeyGen} = \mathsf{KeyGen}'$.
- $\mathsf{Sign}(m = (m_1, \ldots, m_\ell), \mathsf{sk})$: Compute $y_1 = \mathsf{Sign}'((m_1, m_2), \mathsf{sk})$. For all $i > 1$, compute $y_i = \mathsf{Sign}'((y_{i-1}, m_{i+1}), \mathsf{sk})$. Output all the strings $(y_1, \ldots, y_{\ell-1})$ as the signature.
- $\mathsf{Verify}(m = (m_1, \ldots, m_\ell), \sigma = (y_1, \ldots, y_{\ell-1}), \mathsf{vk})$: Output 1 iff all the checks pass:
  1. $\mathsf{Verify}'((m_1, m_2), y_1, \mathsf{vk}) = 1$.
  2. For all $i \in [2, \ell - 1]$, $\mathsf{Verify}'((y_{i-1}, m_{i+1}), y_i, \mathsf{vk}) = 1$.

Easy stuff

This candidate is similar to the CBC-MAC that we saw a few lectures back. And correctness holds here. However, the scheme is not secure. Consider the following adversary:

- Query for signature on (m_1, m_2, m_3). It receives (σ_1, σ_2).
- Next, query for signature on (σ_1, m_4). It receives σ_3.
- Send $m^* = (m_1, m_2, m_4)$ and $\sigma^* = (\sigma_1, \sigma_3)$.

Check that the adversary succeeds in breaking unforgeability. The above attack uses messages of different length. Question 35.01 asks for an attack that works with messages of same length (therefore, the vulnerability is not arising from the message lengths being different; this is different from the CBC-MAC situation). In Question 35.02, we provide a different modification, and you should either prove security, or provide a forgery attack.

5 A secure construction in the random oracle model

The textbook RSA scheme can be modified to get a candidate that is provably secure in the random oracle model. The construction uses a hash function $H : \{0,1\}^* \to \mathbb{Z}_N$ (this will be modeled as a random oracle in the security proof).

- KeyGen : The signing key $\mathsf{sk} = (N, d)$, and the verification key $\mathsf{vk} = (N, e)$ (as usual, $e \cdot d = 1 \bmod \phi(N)$).
- $\mathsf{Sign}(m \in \mathbb{Z}_N, \mathsf{sk} = (N, d))$: The signing algorithm outputs $\sigma = H(m)^d \bmod N$.

- $\mathsf{Verify}(m, \sigma, \mathsf{vk} = (N, e))$: The verification algorithm outputs 1 iff $\sigma^e \bmod N = H(m)$.

5.1 Security proof

We will prove security in the random oracle model. First, let us recall the unforgeability security game in the random oracle model. It is similar to the game described in Figure 1. However, the adversary is now allowed queries to the random oracle.

Unforgeability under Chosen Message Attack, in the random oracle model

- (Setup) Challenger chooses $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(1^n)$ and sends vk to the adversary. It also chooses a uniformly random function f that maps $\{0,1\}^*$ to $\mathbb{Z}_N$.
- (Queries) The adversary is allowed two kinds of queries: random oracle queries and signature queries.
  - (Signature Queries) The adversary is allowed polynomially many signature queries. For every query m_i, the challenger sends $\sigma_i \leftarrow \mathsf{Sign}(m_i, \mathsf{sk})$. Note that the signing algorithm can query the random oracle too.
  - (Random oracle queries) The adversary queries for the random oracle output on x_i, and gets $f(x_i)$.
- (Forgery) Finally, the adversary sends a message m^* and signature σ^*. The adversary breaks **weak unforgeability** if $m^* \neq m_i$ for all i, and $\mathsf{Verify}(m^*, \sigma^*, \mathsf{vk}) = 1$. Note that the verification algorithm (and hence the winning condition also) can depend on the random oracle. Also, while m^* should not be equal to any of the m_i strings, it is allowed to be equal to one of the random oracle queries.

Figure 2: Unforgeability of Signature Schemes in the random oracle model

We will prove security of our RSA based candidate via a sequence of games. The first game is identical to our security game in the ROM. The challenger, instead of choosing the random function, samples the function on-the-fly.

Sequence of hybrid games
Game 0: This is the original unforgeability game in the random oracle model.

- (Setup phase) The challenger chooses $\mathsf{vk} = (N, e)$, $\mathsf{sk} = (N, d)$. It chooses the random oracle 'on-the-fly'. Initially, it creates an empty table $\mathcal{T}$.
- (Queries)
  - (Random oracle queries) The adversary sends $x_i \in \{0,1\}^*$ as a random oracle query. If $(x_i, y_i) \in \mathcal{T}$, the challenger sends y_i in response. Else, it chooses a uniformly random number $y_i \leftarrow \mathbb{Z}_N$, adds (x_i, y_i) to the table, and sends y_i to the adversary.
  - (Signature queries) The adversary sends message m_i as a signing query. If there exists an entry $(m_i, y_i) \in \mathcal{T}$, the challenger sends $y_i^d \bmod N$ to the adversary. Else, it samples a fresh $y_i \leftarrow \mathbb{Z}_N$, adds (m_i, y_i) to $\mathcal{T}$, and sends $\sigma_i = y_i^d$ as the signature.
- (Forgery) The adversary sends m^*, σ^*. It wins if the following conditions are met:
  - $m^* \neq m_i$ for all i
  - If $(m^*, y^*) \notin \mathcal{T}$, the challenger samples a random string y^*, and adversary wins if $(\sigma^*)^e = y^*$. If $(m^*, y^*) \in \mathcal{T}$, the adversary wins if $(\sigma^*)^e = y^*$.

Game 1: This game is identical to the previous game, except that the winning condition is altered slightly. In order to win, the adversary must have queried the random oracle at the forgery message.

- (Setup phase) The challenger chooses $\mathsf{vk} = (N, e)$, $\mathsf{sk} = (N, d)$. It chooses the random oracle 'on-the-fly'. Initially, it creates an empty table $\mathcal{T}$.
- (Queries)
  - (Random oracle queries) The adversary sends $x_i \in \{0,1\}^*$ as a random oracle query. If $(x_i, y_i) \in \mathcal{T}$, the challenger sends y_i in response. Else, it chooses a uniformly random number $y_i \leftarrow \mathbb{Z}_N$, adds (x_i, y_i) to the table, and sends y_i to the adversary.
  - (Signature queries) The adversary sends message m_i as a signing query. If there exists an entry $(m_i, y_i) \in \mathcal{T}$, the challenger sends $y_i^d \bmod N$ to the adversary. Else, it samples a fresh $y_i \leftarrow \mathbb{Z}_N$, adds (m_i, y_i) to $\mathcal{T}$, and sends $\sigma_i = y_i^d$ as the signature.
- (Forgery) The adversary sends m^*, σ^*. It wins if the following conditions are met:
  - $m^* \neq m_i$ for all i
  - ~~If $(m^*, y^*) \notin \mathcal{T}$, the challenger samples a random string y^*, and adversary wins if $(\sigma^*)^e = y^*$.~~ $(m^*, y^*) \in \mathcal{T}$, and $(\sigma^*)^e = y^*$.

Game 2: In this game, the challenger guesses the random oracle query that corresponds to the forgery. Let Q denote the number of random oracle queries made by the adversary. The adversary wins if the guess is correct.

- (Setup phase) The challenger chooses $\mathsf{vk} = (N, e)$, $\mathsf{sk} = (N, d)$. It chooses the random oracle 'on-the-fly'. Initially, it creates an empty table $\mathcal{T}$.

  The challenger chooses an index $i^* \leftarrow [Q]$.
- (Queries)
  - (Random oracle queries) The adversary sends $x_i \in \{0,1\}^*$ as a random oracle query. If $(x_i, y_i) \in \mathcal{T}$, the challenger sends y_i in response. Else, it chooses a uniformly random number $y_i \leftarrow \mathbb{Z}_N$, adds (x_i, y_i) to the table, and sends y_i to the adversary.
  - (Signature queries) The adversary sends message m_i as a signing query. If there exists an entry $(m_i, y_i) \in \mathcal{T}$, the challenger sends $y_i^d \bmod N$ to the adversary. Else, it samples a fresh $y_i \leftarrow \mathbb{Z}_N$, adds (m_i, y_i) to $\mathcal{T}$, and sends $\sigma_i = y_i^d$ as the signature.
- (Forgery) The adversary sends m^*, σ^*. It wins if the following conditions are met:
  - $m^* \neq m_i$ for all i
  - m^* is the $(i^*)^{\text{th}}$ random oracle query.
  - $(m^*, y^*) \in \mathcal{T}$, and $(\sigma^*)^e = y^*$.

Game 3: In this game, the challenger does not use d. Instead, it samples the y_i strings differently (for responding to new random oracle/signature queries). For each fresh random oracle query x_i, it first samples $z_i \leftarrow \mathbb{Z}_N$, sets $y_i = x_i^e$ and adds (x_i, y_i, z_i) to the table. It performs a similar thing for signing queries. Note that $z_i = y_i^d$.

z_i^e hona chiahye

- (Setup phase) The challenger chooses $\mathsf{vk} = (N, e)$, $\mathsf{sk} = (N, d)$. It chooses the random oracle 'on-the-fly'. Initially, it creates an empty table $\mathcal{T}$.

  The challenger chooses an index $i^* \leftarrow [Q]$.
- (Queries)

- (Random oracle queries) The adversary sends $x_i \in \{0,1\}^*$ as a random oracle query. If $(x_i, y_i, z_i) \in \mathcal{T}$, the challenger sends y_i in response.

  Else, it chooses a uniformly random number $z_i \leftarrow \mathbb{Z}_N$, sets $y_i = z_i^e$, adds (x_i, y_i, z_i) to $\mathcal{T}$, and sends y_i to the adversary.
- (Signature queries) The adversary sends message m_i as a signing query. If there exists an entry $(m_i, y_i, z_i) \in \mathcal{T}$, the challenger sends z_i to the adversary.

  Else, it samples a fresh $z_i \leftarrow \mathbb{Z}_N$, sets $y_i = z_i^e$, adds (m_i, y_i, z_i) to $\mathcal{T}$, and sends $\sigma_i = z_i$ as the signature.

- (Forgery) The adversary sends m^*, σ^*. It wins if the following conditions are met:
  - $m^* \neq m_i$ for all i
  - m^* is the $(i^*)^{\text{th}}$ random oracle query.
  - $(m^*, y^*) \in \mathcal{T}$, and $(\sigma^*)^e = y^*$.

Analysis: First, observe that the only difference between Game 0 and Game 1 is that in Game 0, the adversary can win even if it has not queried the random oracle on m^*. However, in this case, y^* is chosen at random (after the adversary sends its forgery), and therefore, the adversary's winning probability in this case is at most $1/N$. Hence the winning probabilities in Game 0 and Game 1 differ by at most $1/N$, which is negligible.

Next, note that if an adversary wins with probability ϵ in Game 1, then it wins with probability ϵ/Q in Game 2. This is because the challenger's guess i^* is correct with probability $1/Q$. Conditioned on the guess being correct, the adversary's winning probability is still ϵ, and therefore the winning probability in Game 2 is ϵ/Q.

Third, note that Game 2 and Game 3 are identical from the adversary's perspective. This is because sampling $y_i \leftarrow \mathbb{Z}_N$ is identical to sampling $z_i \leftarrow \mathbb{Z}_N$ and setting $y_i = z_i^e$.

Finally, we need to show that any p.p.t. adversary has negligible advantage in Game 3. This follows from the RSA assumption. We include the complete reeduction below.

Claim 35.01. Suppose there exists a p.p.t. adversary $\mathcal{A}$ that makes Q random oracle queries, and wins in Game 3 with probability ϵ. Then there exists a reduction algorithm $\mathcal{B}$ that breaks the RSA assumption with probability ϵ.

Proof. The reduction algorithm receives (N, e, y^*) from the RSA challenger. It sets $\mathsf{vk} = (N, e)$ and sends it to $\mathcal{A}$. It also maintains an empty table $\mathcal{T}$ and picks $i^* \leftarrow [Q]$. The adversary makes polynomially many signing queries, and Q random oracle queries. For each of the queries, the reduction follows the Game 3 challenger, and does the following:

- (Signing query) The reduction, on receiving a query for m_i, checks if $(m_i, y_1, z_i) \in \mathcal{T}$. If so, it sends z_i. Else, it samples $z_i \leftarrow \mathbb{Z}_N$, sets $y_i = z_i^e$, adds (m_i, y_i, z_i) to $\mathcal{T}$ and sends z_i.

- (Random oracle query) The reduction, on receiving a random query for x_i, checks if $(x_i, y_i, z_i) \in \mathcal{T}$. If so, it sends y_i.

  Else, if it is the $(i^*)^{\text{th}}$ random oracle query, the challenger sends y^* (and adds nothing to the table $\mathcal{T}$).

  Else, it samples $z_i \leftarrow \mathbb{Z}_N$, sets $y_i = z_i^e$, adds (x_i, y_i, z_i) to $\mathcal{T}$ and sends y_i.

Finally, the adversary sends a forgery (m^*, σ^*). If m^* is not equal to the $(i^*)^{\text{th}}$ random oracle query, then the adversary loses Game 3. If it is equal to the $(i^*)^{\text{th}}$ random oracle query, and verification passes, then $(\sigma^*)^e = y^*$. Note, in order to win in Game 3, the adversary does not query for signature on m^* (and therefore the reduction does not need to add m^* to $\mathcal{T}$). □

6 Lecture summary, plan for next lecture, additional resources

Summary: We introduced the notion of digital signatures, saw a few insecure constructions. Next, we discussed how to handle large message spaces. Here, it was necessary to use a CRHF (the CBC-MAC style construction does not work). Finally, we saw a secure construction based on RSA. The security is proven in the random oracle model.

Next lecture: We will see a one-time secure candidate in the standard model, and briefly discuss how to bootstrap it to full security in the standard model. Finally, we will do a course review, and see how these primitives are put together for secure communication in real-world protocols.

Additional resources Chapter 13 of the course textbook.

7 Questions

Question 35.01. Recall the construction described in Section 4. Modify the attack so that it works even if the message space is $(\{0,1\}^n)^3$.

Question 35.02. Let $\mathcal{S} = (\mathsf{KeyGen}', \mathsf{Sign}', \mathsf{Verify}')$ be a strongly unforgeable signature scheme with message space $\{0,1\}^{2n}$ and signature space $\{0,1\}^n$. Consider the following signature scheme with message space $(\{0,1\}^n)^\ell$:

- KeyGen : Choose $\ell - 1$ signing/verification keys $(\mathsf{sk}'_i, \mathsf{vk}'_i) \leftarrow \mathsf{KeyGen}'$ for each $i \in [\ell - 1]$. The signing key is $\mathsf{sk} = (\mathsf{sk}'_1, \dots, \mathsf{sk}'_{\ell-1})$ and the verification key is $\mathsf{vk} = (\mathsf{vk}'_1, \dots, \mathsf{vk}'_{\ell-1})$.
- $\mathsf{Sign}((m_1, \dots, m_\ell), (\mathsf{sk}'_1, \dots, \mathsf{sk}'_{\ell-1}))$: Set $y_1 = \mathsf{Sign}'((m_1, m_2), \mathsf{sk}'_1)$. For all $i \in [2, \ell - 1]$, set $y_i = \mathsf{Sign}'((y_{i-1}, m_{i+1}), \mathsf{sk}'_i)$. Finally, output $y_{\ell-1}$ as the signature.

Verification can be defined as in Question 35.01. Is this signature scheme unforgeable?