The final ciphertext is $(\mathsf{ct}_1, \mathsf{ct}_2)$.

While this construction takes care of the malleability issue, we cannot recover the message from the ciphertext, since H is not invertible. ◊

Lecture 24: November 3rd, 2023

We will now discuss how to fix Construction 6.27. Note that Construction 6.27 is using $H(r)$ as the key for one-time padding the message. Since the one-time pad is malleable, so is the resulting construction. To prevent this malleability attack, we should encrypt the message using an authenticated encryption scheme, with $H(r)$ as the key.

Construction 7.3 (RSA-based CCA secure PKE). *Let* $\mathcal{E}_{\mathsf{ske}} = (\mathsf{Enc}_{\mathsf{ske}}, \mathsf{Dec}_{\mathsf{ske}})$ *be a symmetric key authenticated encryption scheme with message space* $\{0,1\}^*$ *and key space* $\{0,1\}^\lambda$. *Let* $H : \mathbb{Z}_N \to \{0,1\}^\lambda$ *be a deterministic hash function (which will be modeled as a random oracle in the proof).*

- $\mathsf{Enc}(\mathsf{pk} = (N, e), m)$: *Choose* $r \leftarrow Z_N$, *compute* $\mathsf{ct}_1 = r^e \bmod N$, $k = H(r)$, $\mathsf{ct}_2 = \mathsf{Enc}_{\mathsf{ske}}(k, m)$.
- $\mathsf{Dec}(\mathsf{sk} = (N, d), (\mathsf{ct}_1, \mathsf{ct}_2))$: *Compute* $r = \mathsf{ct}_1^d \bmod N$, $k = H(r)$. *Output* $\mathsf{Dec}_{\mathsf{ske}}(k, \mathsf{ct}_2)$.

◊

Correctness is immediate, let us consider the CCA security proof.

Lemma 7.4. *Assuming $\mathcal{E}_{\mathsf{ske}}$ is a CCA secure encryption scheme, and assuming the RSA assumption holds, Construction 7.3 is a public key encryption scheme that is CCA secure in the random oracle model.*

Proof idea: The main challenge in this proof is responding to decryption queries. Recall, in the case of symmetric key encryption, handling decryption queries was easy: if the ciphertext was previously sent by the challenger in response to an encryption query, then we know the decryption of the message; else we send $\perp$. However, since this is a public key encryption scheme, the adversary can generate an encryption of any message of its choice, and send the encryption as a decryption query. In the CCA experiment, the challenger can compute the decryption because it has the RSA secret key d. However, looking ahead, the reduction breaking RSA security will not have the RSA secret key. How does it respond to decryption queries?

The key idea here is that the reduction responds to decryption queries issued by the adversary. As a result, if the adversary queries for a decryption of $\mathsf{ct} = (r^e, \mathsf{Enc}_{\mathsf{ske}}(H(r), m))$, then it should have queried the random oracle on the input r. The reduction maintains a table $\mathcal{T}$, where it has all random oracle queries, together with the corresponding responses. The reduction checks if there is an entry (r, z) in the table such that $r^e = \mathsf{ct}_1$. If so, it uses z to decrypt ct_2.

There is one minor subtlety: what if the adversary first queries for the decryption of ct, and then queries the random oracle on r? To handle this, the reduction should sample a random z, use z to decrypt ct_2, and store (ct_1, z) in a separate

we will see in the formal proof that the reduction needs to store only one table.

table $\mathcal{S}$. Later, when it receives a random oracle query for r, it checks if $r = \mathsf{ct}_1$ for some entry (ct_1, z) in $\mathcal{S}$. If so, it adds (r, z) to $\mathcal{T}$ and sends z as the random oracle response.

Proof. The proof proceeds via a sequence of hybrid games.

Game 0: This is the original semantic security game in the random oracle model.

- (Setup phase) The challenger chooses $\mathsf{pk} = (N, e)$, $\mathsf{sk} = (N, d)$. It chooses the random oracle 'on-the-fly'. Initially, it creates an empty table $\mathcal{T}$.
- (Pre-challenge queries)
  - (Pre-challenge Random oracle queries) The adversary sends $x_i \in \mathbb{Z}_N$ 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 n bit string y_i, adds (x_i, y_i) to the table, and sends y_i to the adversary.
  - (Pre-challenge Decryption queries) The adversary sends ciphertext $\mathsf{ct}_i = (\mathsf{ct}_{i,1}, \mathsf{ct}_{i,2})$ as a decryption query. The challenger checks if there exists an entry $(\mathsf{ct}_{i,1}^{1/e}, y) \in \mathcal{T}$. If so, it sends $\mathsf{Dec}_{\mathsf{ske}}(y, \mathsf{ct}_{i,2})$. Else it samples a fresh $y \leftarrow \{0,1\}^n$, adds $(\mathsf{ct}_{i,1}^{1/e}, y)$ to $\mathcal{T}$, and sends $\mathsf{Dec}_{\mathsf{ske}}(y, \mathsf{ct}_{i,2})$.
- (Challenge phase) The adversary sends m_0^*, m_1^*. Challenger chooses $x^* \leftarrow \mathbb{Z}_N$ and $b \leftarrow \{0,1\}$. Next, it checks if $(x^*, y) \in \mathcal{T}$. If so, it sets $\mathsf{ct}_1^* = (x^*)^e$, $\mathsf{ct}_2 = \mathsf{Enc}_{\mathsf{ske}}(y, m_b^*)$.

  If no $(x^*, y) \in \mathcal{T}$, it chooses $y \leftarrow \{0,1\}^n$, adds (x^*, y) to $\mathcal{T}$, sets $\mathsf{ct}_1^* = (x^*)^e$, $\mathsf{ct}_2^* = \mathsf{Enc}_{\mathsf{ske}}(y, m_b^*)$.

  It sends $(\mathsf{ct}_1^*, \mathsf{ct}_2^*)$ to $\mathcal{A}$.
- (Post-challenge queries)
  - (Post-challenge Random Oracle queries) Same as pre-challenge random oracle queries
  - (Post-challenge Decryption queries) Same as pre-challenge decryption queries, except that the challenger outputs $\perp$ if the decryption query is equal to the challenge ciphertext.

Game 1: This is similar to the previous game, except that the challenger stores (x^e, y) in $\mathcal{T}$ instead of (x, y). Note that since everything else is identical, the game is identical in the adversary's view. However, this change is important because instead of computing the e^{th} root (which requires the secret key component $d = 1/e$), the challenger only computes the e^{th} power (which is an easy operation).

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

  - (Pre-challenge Random oracle queries) The adversary sends $x_i \in \mathbb{Z}_N$ as a random oracle query. If $(x_i^e, y_i) \in \mathcal{T}$, the challenger sends y_i in response. Else, it chooses a uniformly random n bit string y_i, adds (x_i^e, y_i) to the table, and sends y_i to the adversary.
  - (Pre-challenge Decryption queries) The adversary sends ciphertext $\mathsf{ct}_i = (\mathsf{ct}_{i,1}, \mathsf{ct}_{i,2})$ as a decryption query. The challenger checks if there exists an entry $(\mathsf{ct}_{i,1}, y) \in \mathcal{T}$. If so, it sends $\mathsf{Dec}_{\mathsf{ske}}(y, \mathsf{ct}_{i,2})$. Else it samples a fresh $y \leftarrow \{0,1\}^n$, adds $(\mathsf{ct}_{i,1}, y)$ to $\mathcal{T}$, and sends $\mathsf{Dec}_{\mathsf{ske}}(y, \mathsf{ct}_{i,2})$.

- (Challenge phase) The adversary sends m_0^*, m_1^*. Challenger chooses $x^* \leftarrow \mathbb{Z}_N$ and $b \leftarrow \{0,1\}$. Next, it checks if $((x^*)^e, y) \in \mathcal{T}$. If so, it sets $\mathsf{ct}_1^* = (x^*)^e$, $\mathsf{ct}_2 = \mathsf{Enc}_{\mathsf{ske}}(y, m_b^*)$.

  If no $((x^*)^e, y) \in \mathcal{T}$, it chooses $y \leftarrow \{0,1\}^n$, adds $((x^*)^e, y)$ to $\mathcal{T}$, sets $\mathsf{ct}_1^* = (x^*)^e$, $\mathsf{ct}_2^* = \mathsf{Enc}_{\mathsf{ske}}(y, m_b^*)$.

  It sends $(\mathsf{ct}_1^*, \mathsf{ct}_2^*)$ to $\mathcal{A}$.

- (Post-challenge queries)

  - (Post-challenge Random Oracle queries) Same as pre-challenge random oracle queries
  - (Post-challenge Decryption queries) Same as pre-challenge decryption queries, except that the challenger outputs $\perp$ if the decryption query is equal to the challenge ciphertext.

Game 2: In this game, the challenger does not use the table for the challenge ciphertext. It just samples a random key k and uses it for the challenge ciphertext, but does not add $((x^*)^e, k)$ to the table $\mathcal{T}$.

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

- (Pre-challenge queries)

  - (Pre-challenge Random oracle queries) The adversary sends $x_i \in \mathbb{Z}_N$ as a random oracle query. If $(x_i^e, y_i) \in \mathcal{T}$, the challenger sends y_i in response. Else, it chooses a uniformly random n bit string y_i, adds (x_i^e, y_i) to the table, and sends y_i to the adversary.
  - (Pre-challenge Decryption queries) The adversary sends ciphertext $\mathsf{ct}_i = (\mathsf{ct}_{i,1}, \mathsf{ct}_{i,2})$ as a decryption query. The challenger checks if there exists an entry $(\mathsf{ct}_{i,1}, y) \in \mathcal{T}$. If so, it sends $\mathsf{Dec}_{\mathsf{ske}}(y, \mathsf{ct}_{i,2})$. Else it samples a fresh $y \leftarrow \{0,1\}^n$, adds $(\mathsf{ct}_{i,1}, y)$ to $\mathcal{T}$, and sends $\mathsf{Dec}_{\mathsf{ske}}(y, \mathsf{ct}_{i,2})$.

- (Challenge phase) The adversary sends m_0^*, m_1^*. Challenger chooses $x^* \leftarrow \mathbb{Z}_N$ and $b \leftarrow \{0,1\}$.

  ~~Next, it checks if $((x^*)^e, y) \in \mathcal{T}$. If so, it sets $\mathsf{ct}_1^* = (x^*)^e$, $\mathsf{ct}_2 = \mathsf{Enc}_{\mathsf{ske}}(y, m_b^*)$.~~

  It chooses $k \leftarrow \{0,1\}^n$, ~~adds $((x^*)^e, y)$ to $\mathcal{T}$,~~ sets $\mathsf{ct}_1^* = (x^*)^e$, $\mathsf{ct}_2^* = \mathsf{Enc}_{\mathsf{ske}}(k, m_b^*)$.

  It sends $(\mathsf{ct}_1^*, \mathsf{ct}_2^*)$ to $\mathcal{A}$.

- (Post-challenge queries)

- (Post-challenge Random Oracle queries) Same as pre-challenge random oracle queries. Note: if there is a random oracle query for x^*, then the adversary samples a fresh string y^*, adds (x^*, y^*) to $\mathcal{T}$ and sends y^*. This is because $((x^*)^e, k)$ was not added to $\mathcal{T}$ during the challenge phase.
- (Post-challenge Decryption queries) Let ct_i be the decryption query. If $\mathsf{ct}_i = \mathsf{ct}^*$, then output $\perp$.
  If $\mathsf{ct}_{i,1} = \mathsf{ct}_1^*$ but $\mathsf{ct}_{i,2} \neq \mathsf{ct}_2^*$, the challenger sends $\mathsf{Dec}_{\mathsf{ske}}(\mathsf{ct}_{i,2}, k)$.[3]
  If $\mathsf{ct}_{i,1} \neq \mathsf{ct}_1^*$, it proceeds similar to the pre-challenge decryption queries. It first checks if $(\mathsf{ct}_{i,1}, y)$ is present in $\mathcal{T}$. If so, it sends $\mathsf{Dec}_{\mathsf{ske}}(y, \mathsf{ct}_{i,2})$. Else it samples $y \leftarrow \{0,1\}^n$, adds $(\mathsf{ct}_{i,1}, y)$ to $\mathcal{T}$ and sends $\mathsf{Dec}_{\mathsf{ske}}(y, \mathsf{ct}_{i,2})$.

Analysis: First, observe that Game 0 and Game 1 are equivalent from the adversary's perspective. In one case, the challenger adds (x, y) to the table, in the other case it adds (x^e, y) to the table. Therefore, if an adversary wins with non-negligible advantage in Game 0, then it wins with non-negligible advantage in Game 1.

Next, let us consider the differences between Game 1 and Game 2.

- The first difference is in the challenge phase. In Game 1, the challenger first samples $x^* \leftarrow \mathbb{Z}_N$, then checks if $((x^*)^e, y) \in T$. In Game 2, the challenger simply picks a random k and uses it for computing ct_2^*.
- The second difference is in the post-challenge queries. Note, in Game 1, after the challenge phase, there exists a tuple $((x^*)^e, y) \in \mathcal{T}$. If there is a post-challenge random oracle query on x^*, the challenger sends y. If there is a post-challenge decryption query for $(\mathsf{ct}_1^*, \mathsf{ct}_{i,2})$, then the challenger uses y to decrypt $\mathsf{ct}_{i,2}$. In Game 2, the challenger sends $((x^*)^e, \mathsf{Enc}_{\mathsf{ske}}(k, m_b))$, 'stores' k separately, but does not add $((x^*)^e, k)$ to $\mathcal{T}$. As a result, if there is a post-challenge random oracle query for x^*, the challenger samples a uniformly random string y^* and sends this to the adversary. However, if there is a decryption query for $(\mathsf{ct}_1^*, \mathsf{ct}_{i,2})$, the challenger uses k to decrypt $\mathsf{ct}_{i,2}$. Therefore, the difference is in the post-challenge random oracle response to x^*.

Intuitively, the first event will happen with very low probability (since x^* is chosen at random during the challenge phase). The second one will also happen with low probability, and this argument relies on the hardness of RSA problem. If an adversary has significant difference in winning probabilities in the two games, then it must break the RSA assumption.

Formal proof: Let p_b denote the probability of adversary $\mathcal{A}$ winning in Game b.

Claim 7.5. *Suppose there exists a p.p.t. adversary $\mathcal{A}$ such that $p_1 - p_2 = \epsilon$. Then there exists a p.p.t. algorithm $\mathcal{B}$ that solves the RSA problem with probability ϵ.*

[3] This is needed because we have not added (ct_1^*, k) to $\mathcal{T}$.

Proof. The reduction algorithm receives (N, e, y) from the RSA challenger. It sends $\mathsf{pk} = (N, e)$ to the adversary. The reduction also maintains a table $\mathcal{T}$, which is initially empty.

Next, the adversary makes pre-challenge random oracle, and pre-challenge decryption queries. For each RO query x_i, the reduction algorithm does the following:

It should be (x_i^e,y_i)

- If there exists an entry (x_i, y_i) in $\mathcal{T}$, then the reduction algorithm sends y_i.
- If there exists no entry corresponding to x_i in $\mathcal{T}$, the reduction algorithm samples $y_i \leftarrow \{0,1\}^n$, adds (x_i, y_i) to $\mathcal{T}$ and sends y_i to $\mathcal{A}$. It also checks if $x_i^e = y$. If so, it sends x_i to the RSA challenger (and wins the RSA game).

For the challenge phase, the reduction algorithm receives challenge messages m_0^*, m_1^* from the adversary. It picks a uniformly random string $k \leftarrow \{0,1\}^n$, sets $\mathsf{ct}_1^* = y$. Next, it picks a random bit $b \leftarrow \{0,1\}$, computes $\mathsf{ct}_2^* = \mathsf{Enc}_{\mathsf{ske}}(k, m_b^*)$ and sends $(\mathsf{ct}_1^*, \mathsf{ct}_2^*)$ to $\mathcal{A}$.

Note: If the e^{th} root of y was queried in one of the random oracle queries, then the reduction has already won the RSA game. If it was not queried, then Game 0 and Game 1 are identical up to the challenge phase (and the reduction perfectly simulates the two games up to this point).

In this proof, Game 0 and Game 1 should be replaced by Game 1 and Game 2

The adversary then makes post-challenge random oracle queries. For each RO query x_i, the reduction algorithm checks if $x_i^e = y$. If so, it sends x_i to the RSA challenger. Else, it checks if there exists an entry $(x_i, y_i) \in \mathcal{T}$. If so, it sends y_i to $\mathcal{A}$. If there is no such entry, then it samples $y_i \leftarrow \{0,1\}^n$, sends y_i to $\mathcal{A}$ and adds (x_i, y_i) to $\mathcal{T}$.

Similar to what is mentioned above, if the adversary does not make a random oracle query on input $y^{(1/e)}$, then Game 0 and Game 1 are identical. Therefore, if the winning probabilities are different in the two games, then it must query the random oracle on input $y^{(1/e)}$, in which case the reduction wins the RSA game. □

Claim 7.6. *Suppose there exists a p.p.t. adversary $\mathcal{A}$ that wins in Game 1 with probability ϵ. Then there exists a reduction algorithm $\mathcal{B}$ that wins the CCA security game against $\mathcal{E}_{\mathsf{ske}}$ with probability ϵ.*

Should be Game 2

Proof. The proof follows from the simple observation that in Game 1, string k is only used in the challenge phase (for computing the challenge ciphertext) and in the post-challenge decryption query phase (for answering decryption queries where the first component is identical to ct_1^*).

The reduction algorithm chooses (N, e) and sends it to the adversary. The reduction maintains a table $\mathcal{T}$ which is initially empty. On receiving a random oracle query x_i, the reduction checks if there exists (x_i, y_i) in the table. If so, it sends y_i. Else, it chooses $y_i \leftarrow \{0,1\}^n$, adds (x_i, y_i) to $\mathcal{T}$ and sends y_i to $\mathcal{A}$.

When the adversary sends (m_0, m_1) as the challenge messages, the reduction sends (m_0, m_1) to the $\mathcal{E}_{\mathsf{ske}}$ challenger. It receives ct from the challenger. The reduction algorithm chooses $x \leftarrow \mathbb{Z}_N$, sets $\mathsf{ct}_1 = x^e \bmod N$, $\mathsf{ct}_2 = \mathsf{ct}$ and sends $(\mathsf{ct}_1, \mathsf{ct}_2)$ to $\mathcal{A}$.

this should be ct_1*

The post-challenge random-oracle queries are handled similar to the pre-challenge ones. For the post-challenge decryption query $(\mathsf{ct}_{i,1}, \mathsf{ct}_{i,2})$, if $\mathsf{ct}_{i,1} = \mathsf{ct}_1^*$,

the reduction sends $\mathsf{ct}_{i,2}$ to the $\mathcal{E}_{\mathsf{ske}}$ challenger, and forwards the decryption to the adversary.

Note: In Game 2's challenge phase, the challenger chooses $x \leftarrow \mathbb{Z}_N$, $k \leftarrow \{0,1\}^n$, but does not add (x,k) to $\mathcal{T}$. As a result, if the adversary queries the random oracle on this x, it receives a fresh random string y. This is important for our reduction here.

Finally, the adversary sends its guess b', which the reduction forwards to the challenger. The winning probability of $\mathcal{A}$ in Game 1 is equal to the winning probability of the reduction against $\mathcal{E}_{\mathsf{ske}}$ (why?). □

□

7.3 *PKE-CCA History and Notes*

- The definition of CCA security for public key encryption was given by Moni Naor and Moti Yung [NY90] and Charles Rackoff and Daniel Simon [RS91]. They gave a weaker security notion (called CCA-1, where only pre-challenge decryption queries are allowed), and also gave a construction *in the standard model*. The Naor-Yung construction is very elegant, using *noninteractive zero-knowledge proofs*. Although it is very inefficient practically, the proof technique has found several other applications. The full CCA definition (also referred to as CCA-2), as well as a construction in the standard model, was given by Danny Dolev, Cynthia Dwork and Moni Naor [DDN03]. Note that the first CCA definitions came much before Bleichenbacher's attack.

- Daniel Bleichenbacher demonstrated the practical CCA attack in 1998 [Ble98]. This attack has immense impact on cryptography and security. Researchers have shown Bleichenbacher-style attacks on several other systems. This attack is also the reason why CCA security is the 'gold standard' for encryption schemes. For instance, see the recent NIST call for post-quantum cryptosystems - they want CCA security in the random oracle model:

  > *NIST intends to standardize one or more schemes that enable "semantically secure" encryption or key encapsulation with respect to adaptive chosen ciphertext attack, for general use. This property is generally denoted IND-CCA2 security in academic literature.*

- The first practical CCA secure encryption scheme in the standard model was given by Ronald Cramer and Victor Shoup [CS98]. See also Victor's excellent writeup on why chosen ciphertext security matters.

- In practice, we use schemes that are proven CCA secure in the random oracle model. After Bleichenbacher's attack, the PKCS standards were updated. Today, a widely used candidate is *RSA with optimal asymmetric encryption padding* (RSA-OAEP). This was proposed by Mihir Bellare and Philip Rogaway [BR94]. Similar to the PKCS v1.5 scheme, here the ciphertext consists of only one component. However, it is provably secure in the random oracle model.

- A major open question in theoretical public key cryptography is whether CPA security implies CCA security in the standard model. Quite often, constructing a CPA secure scheme is much easier than constructing a CCA secure scheme. Therefore, it would be nice if we could take any CPA secure scheme, and convert that into a CCA secure scheme. However, currently we don't know how to do this, and it is also not clear if there are any barriers towards such a transformation.
- In the random oracle model, we have an efficient transformation from CPA security to CCA security. This was given by Eiichiro Fujisaki and Tatsuaki Okamoto [FO99]. The leading proposals for the post-quantum encryption schemes first build a post-quantum CPA secure encryption scheme, and then use the Fujisaki-Okamoto transformation to obtain a post-quantum CCA secure encryption scheme in the random oracle model.

Chapter Summary and References

- We saw that CPA security may not suffice for real-world applications. One of the cryptographic standards in the 90s, PKCS v1.5, was CPA secure, but there was an active attack against this encryption scheme.
- Next, we defined security that captures active attacks — security against chosen ciphertext attacks (CCA). The constructions discussed in the previous section were not CCA secure.
- Finally, we saw a heuristic construction that can be proven CCA secure in the random oracle model, assuming hardness of RSA.

Textbook References: Boneh-Shoup (v6, [BS23]): 12.1, 12.2, 12.3, 12.8.3.

Suggested textbook exercises: 12.2, 12.4, 12.5, 12.16.