This lecture will introduce two schemes that use random oracle (RO) model: chosen cyphertext secure encryption and public key signatures.

1 Chosen Cyphertext Secure Encryption

The following scheme uses a trapdoor one way permutation (OWP) f to build a chosen cyphertext text secure *encryption*:

$$E(m) = (f(r), H(0||r) \oplus m, H(1||r||m)).$$

Let us call: $f(r) \equiv y$, $H(0||r) \oplus m \equiv z$, $H(1||r||m) \equiv tag$. For instance OWP in this encryption scheme can be RSA. Two hash functions are used as random oracles (RO), and that will be some sort of "one time pad". Last time we saw a scheme that was built secure under CPA. Today we will prove that this construction is secure against CCA2. The tags: z and tag are independent random functions and are used here to preserve the integrity of the message. Note that the trick used here is a good way to construct two independent random functions with a single hash oracle H. Intuitively, the change of r or m in this construction will make parts of the above 3-tuple invalid and thus provide security against CCA2.

Decryption: $D(y, z, tag)$:
1. $r \leftarrow f^{-1}(y)$
2. $m \leftarrow H(0||r) \oplus z$
3. If $tag = H(1||r||m)$ return M else return $\perp$

Our aim is to show that the pair (E, D) as defined above is a scheme that is secure against chosen cyphertext attack (CCA2). Without loss of generality we can reformulate the security of the construction with the security of OWP in the following manner.

Theorem 1
If (f, f^{-1}) is a (t, ϵ)-TOWP, then (E, D) is a $(t - \mathcal{O}(1), 1, q_v, \epsilon + \frac{t}{2^{160}})$-RO-IND-CCA2 secure, where $q_v < t$, and $length(tag) = 160$.

Note that it is sufficient to prove the security against one query to the encryption oracle as similar result will also hold for any number of queries to E by hybrid argument. This result is very useful in the practice as it uses RO model without any demanding assumption, other than the existence of OWP.

We now prove this theorem by the following reduction.

Lemma 2
If A breaks (E, D) with $Adv\ A > \epsilon + \frac{t}{2^{160}}$ then on input y one can compute $f^{-1}(y)$ without the trapdoor with the probability greater then ϵ.

Let us first consider the following construction depicted in the figure 1, in which we first assume that A does not use decryption oracle D.

To prove the lemma, we show that A, as defined above, can be used to reverse the one way permutation for us. The reduction is pretty simple, but first notice that we can run A in virtual a world, in which we take over the oracles and provide answers to the queries A makes in unexpected manner, as long as we keep the same the distribution over all possible answers that A is expecting. Therefore, each time A makes a query a_i we assume that we return some uniform random value, $H(a_i)$, as a random function.

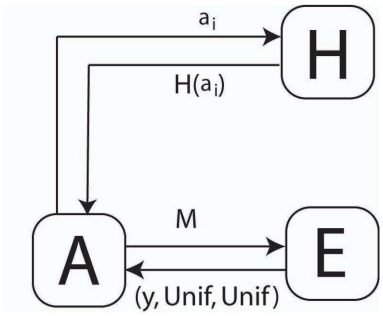

Figure 1.

But when A makes only one query to the encryption oracle with the message M, we will return value $y = f(r)$, that we would like A to invert for us, followed by two $Unif$ distribution tags: $(y, Unif, Unif)$. As the true value of r is not known to us, we cannot provide valid tags.

Just as the last time, it could have happened that A queried H with some value $a_i = 0||r_i$ or $a_i = 1||r_i||m$ such that $f(r_i) = y$. In that case our reduction falls apart, as A has basically managed to reverse f as: $r_i = f^{-1}(y)$ and to detect that we are cheating by patching two uniformly random strings to y.

This is the *bad* situation, as we can detect that A has broken the OWP but we cannot necessarily do something about it. On the other hand, if the bad situation does not happen, that means the hash function has never seen $a_i = 0||r_i$ or $a_i = 1||r_i||m$ such that $f(r_i) = y$, cheating is not detected, A's behavior is unchanged and it has zero advantage. [1] Moreover, if hash function has never seen $a_i = 0||r_i$ or $a_i = 1||r_i||m$, for some m, such that $f(r_i) = y$ the truth table has not been filled in yet, so we can fill two uniform values that we have used to patch y, as they have exactly the same distribution.

Now to prove the security against CCA2, we consider the case when A queries decryption oracle D as well, figure 2. Our aim now is to prove that security against CPA implies CCA2 security as the decryption oracle does not provide any new information to A.

How do we answer decryption queries?

We have to be ready for both valid and invalid cyphertext queried by A. Remember, A has the access to hash function so given the message m' and random r', A knows $H(0||r')\oplus m'$ and $H(1||r'||m')$ and can construct the valid cyphertext without even querying E. We should also see how to handle the situation when A is so lucky to come up with a valid cyphertext without even making a $1||r'||m'$ query to H oracle.

Here is one possibility how decryption oracle can handle A's queries:

[1]Note that when the bad event *does not* happen A's response is independent of M! For example in terms of right-left attacker, if A is querying its message M in the form (M_0, M_1) and the response is independent of M, A has no way to figure out weather E is right or left oracle, so it gets the zero advantage against CPA.

Decryption oracle $D(y', z', tag')$:

If $\exists i$ so that $a_i = 0||r'$, $1||r'||m'$ has been queried to H and $f(r\prime) = y\prime$;
  $m\prime = z\prime \oplus H(0||r\prime)$;
  if $(1||r\prime||m\prime)$ was queried on H and $H(1||r\prime||m\prime) = tag\prime$;
    return $m\prime$;
  else return $\perp$;
else return $\perp$.

However, this behavior of D oracle does not provide the valid answer in the case when A comes up with the *valid* $tag' = H(1||f^{-1}(y')||z' \oplus H(0||f^{-1}(y'))$ without previously querying H to construct $H(1||f^{-1}(y')||z' \oplus H(0||f^{-1}(y'))$. In this case our D oracle returns $\perp$ for the valid cyphertext. Therefore, with probability $\frac{t}{2^{160}}$ A has the chance to discover that it is being exploited. Note also that then A could have simulated D itself with same probability of winning. Thus the decryption oracle does not increase the probability of A to win the game and therefore security against CPA implies the secure against CCA2.

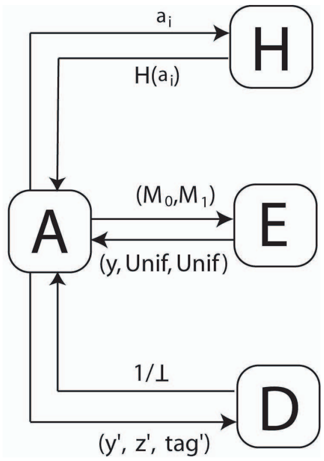

Figure 2.

2 Public Signatures

Public signature scheme has two functions: signing and verification algorithm. Communication parties, Alice and Bob, would like to share the message that adversary A can not tamper with. The set up is as follows.

Alice has some private key and Bob knows only Alice's public key. Alice would like to send a message to Bob such that integrity of her message is guarantied, so that message can not be signed by someone else. In order to do that, Alice sends a message with her signature. Bob will receive the message and use Alice's public key and verification algorithm to see that the message is valid.

Signing Algorithm: $S : TK \times \{0,1\}^* \longrightarrow Tags$;
Verification Algorithm: $V : PK \times \{0,1\}^* \times Tags \longrightarrow 0, 1$, so
$V_{pk}(m, S_{tk}(m)) = 1$ if the message m is authentic.

Of course, there is also key generation algorithm that Alice can use to generate public key pk and private key tk. This scheme is in contrast with the MAC as Bob is able to check the authenticity without being able to generate signature himself.

With the public signatures we would also like to ensure that adversary A, who sees a signature on many messages, can not forge a signature on the message that she has not previously seen. Formally this is defined as follows.

Security against the external forgeries:

$$\begin{aligned} AdvA &= Pr[A^{H,S^H_{tk}(\cdot)}(pk) \text{ forges }] \\ &= Pr[(m,\tau) \leftarrow A^{H,S^H_{tk}(\cdot)}(pk); m \text{ was not queried to } S^H_{tk}(\cdot) \text{ and } V^H_{pk}(m,\tau)=1] \ . \end{aligned}$$

It is important to notice that A can check herself whether the forging was successful, by running the verification algorithm. So when we talk about (t,q,ϵ) attack we think of A running in time t, making q queries, and having the probability of forging at most ϵ. Random oracle version of this, is when we provide a hash function H and allow that signature and verification algorithm depend on H, that we denote as $S^H_{tk}(\cdot)$ and $V^H_{pk}(\cdot,\cdot)$ respectively, in the above definition of security against the external forgeries.

Here are a few signature schemes that depend on RSA.

1) Raw RSA: Public key is $pk=(e,N)$, the private key is $tk=d$ and messages $m \in M_{sgs}=(\mathbb{Z}/n\mathbb{Z})^*$.
Signature algorithm: $S(m)=m^d \ mod \ N=\tau$
Verification algorithm: $V(m,\tau)=\begin{cases} 1 \text{ if } \tau^e=m \\ 0 \text{ otherwise.} \end{cases}$

1') In general if (f,f^-1)-TOWP, we can have a signature and verification as:

$$S(m)=f^{-1}(m)=\tau \ ,$$

$$V(m,\tau)=\begin{cases} 1 \text{ if } f(\tau)=m \\ 0 \text{ otherwise,} \end{cases}$$

respectively.

2) Signature using a hash function:

$$S(m)=Hash(m)^d \, mod \, N \ ,$$

$$V(m,\tau)=\begin{cases} 1 \text{ if } f(\tau)=Hash(m) \\ 0 \text{ otherwise.} \end{cases}$$

2') Hash with TOWP signature:

$$S(m)=f^{-1}(Hash(m)) \ ,$$

$$V(m,\tau)=\begin{cases} 1 \text{ if } f(\tau)=Hash(m) \\ 0 \text{ otherwise.} \end{cases}$$

All of the above schemes are fast but broken! Here are the attacks:

- For scheme 1) : Attack#1 if we use RSA:

$$S(m_1,m_2)=S(m_1)\cdot S(m_2) \ mod \ N$$

Attack#2: $S(1)=1$

Attack#3: We can trick the verificatation algorithm just by making $\tau^e = m$, so pick $\tau \xleftarrow{\$} (\mathbb{Z}/n\mathbb{Z})^*$ and let $m = \tau^e \, mod \, N$. Such m might be irrelevant but this is still a forgery.

- For the scheme 2) since we use hash functions, we would need to impose some assumptions as follows:

  - hash function must not be homomorphic: $Hash(m_1 \cdot m_2) = Hash(m_1)Hash(m_2)$;
  - hash function must be one way function;
  - hash function must be a collision resistant, i.e. collisions should be hard to find.
  - etc...

  These requirements are however unjustifiable and we would like to find the sufficient conditions. Unfortunately, such conditions are in general hard to find.

  For example, here is how even if all of the above itemized conditions are satisfied, scheme that uses hash function is still insecure. Let:

$$S(m) = (00...0||SHA1(m))^d \; mod \; N \; ,$$

  where $length(SHA1(m)) = 160$ and we fill the rest, say up to 1000 bits, with zeros. This scheme is broken because it is easy to take an inverse of RSA if a random element belongs to the small subset of $(\mathbb{Z}/N\mathbb{Z})^*$. Remember, for hardness of RSA we have assumed that it is hard to invert RSA on a random element over the whole group, not a random element over some exponentially small subset of the group. In this construction element $00...0||SHA1(m)$ belongs to such exponentially small subset of the group, therefore no RSA assumption holds and the signature might be easy to forge:

$$m \xrightarrow{Hash} SHA1(m) \xrightarrow{RSA^{-1}} \tau = (00...0||SHA1(m))^d \; mod \, N$$

Strangely, there is also so called PKCS#1 scheme, that has similar construction but instead of padding with zero string we pad with the different one:

$$S(m) = (01FF...FF||SHA1(m))^d \; mod \, N \; .$$

Security of this construction is unjustified as no security proof exists, however the scheme is widely used because currently there are no known attacks. As in the previous case any security proof for this construction by contradicting RSA assumptions cannot exist.