Lecture 16 - CCA Security

Boaz Barak

March 31, 2010

Reading Boneh-Shoup 12.1,12.2,12.3,12.6

Review - hardcore bits

Key exchange Suppose we have following situation: Alice wants to buy something from the well known website Bob.com

Since they will exchange private information (Alice's credit card, address etc.) they want to use encryption. However, they do not share a key between them.

Using a key exchange protocol. It seems that we already learned a protocol to do that: Alice and Bob can run a *key exchange protocol*. One such protocol is the Diffie-Hellman protocol, but they can also run the following RSA-based protocol:

$A \leftarrow B$ Bob chooses a pair of RSA keys (e, d) and sends e to Alice.

$A \rightarrow B$ Alice chooses a key $k \leftarrow_R \{0,1\}^n$ and sends $\mathsf{E}_e(k)$ to Bob.

$A \leftrightarrows B$ Bob and Alice can now can now continue their interaction with the shared secret key k.

Insecurity of basic key exchange protocol: This protocol is secure for a *passive / eavesdropping* adversary, but it is not secure against an *active* adversary. Indeed, a man-in-the-middle Charlie can play Bob to Alice and Alice to Bob. That is, Charlie will receive (e, d) from Bob but will not pass this on to Alice. Rather he will choose his own RSA pair (e', d') and send e' to Alice. Alice will then send $\mathsf{E}_{e'}(k)$ to Charlie. Charlie can decrypt to find k and then send $\mathsf{E}_e(k)$ to Bob.[1] From now on Charlie will be able to listen in to all of Alice and Bob's communication.

Obvious fix. This attack is inherent since if Bob and Alice don't know anything about each other then of course Charlie can impersonate them to one another. However, we are in a setting where Bob is a well known web site, and hence we can assume that Alice already has Bob's public key. This prevents this attack but it is not clear that it is secure.

Example: SSL protocol. The SSL protocol is the most widely used protocol for such transactions (this is the protocol used to access encrypted web sites, and is the standard for all transactions involving credit card etc.). However, in V3.0, the heart of the protocol was the following interaction:

[1] He can also choose his own key k' and send $\mathsf{E}_e(k')$ to Bob.

- Client sends $\mathsf{E}_e(k)$ to the server where $\mathsf{E}_e(\cdot)$ is padded RSA according to standard PKCS #1 V1.5 (a scheme believed to be semantically secure).
- Server validates decryption is according to standard, otherwise sending `invalid decryption`, and if so, uses k as the key.

The padding scheme is the following: if $\{f_e\}$ is the RSA trapdoor permutation collection then to encrypt x choose r to be a random string (of length at least 8 bytes) conditioned on not having any zero byte, and let $x' = 0 \circ 2 \circ r \circ 0 \circ x$. Define the function $PKCS(x')$ to output 1 iff x' is of this form. For a random x', the probability that $PKCS(x') = 1$ is about 2^{-16}.

In a surprising paper, Bleichenbacher proved that the function $PKCS(\cdot)$ is some kind of a *hard core* of RSA.[2] That is, he showed that if you have an oracle that given y outputs 1 iff $PKCS(f_e^{-1}(y)) = 1$, then you can use it to invert the one-way permutation $f_e(\cdot)$ using not too many queries. It follows that SSL protocol is insecure, since an attacker can open as many sessions with the server as it likes, essentially using the server as this oracle. (Note that no matter what happens later in the protocol, once the attacker received this error message, she got the response she needed, even if the server will abort later.)

Reflection: In retrospect, it should have been clear that it is a bad idea to use a scheme that is only CPA secure and not a CCA secure scheme. In fact, if the designers of SSL had tried to *prove* security of their protocol, they would have seen that CCA (or a close variant) is an essential condition for such a proof to go through.

More details on the actual SSL protocol appear in the BS book. Some other attacks on SSL include Goldberg-Wagner attack on the pseudorandom generation, Version-rollback attack, Protocol changing attack, Variations/extensions of the Bleichenbacher attack.

Constructions of CCA secure public key encryption Constructing CCA secure public key encryption is more challenging than the private key case. In this lecture we'll do so only in the random oracle model. We start with a CPA secure scheme in the random oracle model (we've already seen such an encryption but this one has has some efficiency advantages over the Goldreich-Levin hardcore based one)

[2] Actually, there were several previous results about very related hard-core functions for RSA, but people always thought about these results as establishing theoretical security and not practical insecurity.

A CPA Secure Scheme:

- Let $G : \{0,1\}^n \to \{0,1\}^n$ be a random oracle and $\{(f, f^{-1})\}$ be collection of trapdoor permutations. The public key of the scheme will be $f(\cdot)$ while the private key be f^{-1}.
- To encrypt $x \in \{0,1\}^n$, choose $r \leftarrow_{\text{R}} \{0,1\}^n$ and compute $f(r), G(r) \oplus x$.
- To decrypt y, z compute $r = f^{-1}(y)$ and let $x = r \oplus z$.

Theorem 1. *The above scheme is CPA secure in the random oracle model.*

Proof. For public key encryption, the encryption oracle is redundant and so CPA security means that an adversary A that gets as input the encryption key ($f(\cdot)$ in our case) cannot tell apart $\mathsf{E}(x^1)$ and $\mathsf{E}(x^2)$ for every x^1, x^2.

However, in the random oracle model we need to give A also access to the random oracle $G(\cdot)$.

We denote the ciphertext A gets as challenge by y^*, z^* where $y^* = f(r^*)$ and $z^* = G(r^*) \oplus x^*$. We start by proving the following:

Claim 1.1. *The probability that A queries r^* of its oracle $G(\cdot)$ is negligible.*

Proof. Consider the following experiment: instead of giving $z^* = G(r^*) \oplus x^*$, we give A the string $z^* = u \oplus x^*$ where u is a uniform element. The only way A could tell apart the two cases is if he queries r^* to G and sees that the answer is different from u, but then we already "lost". Thus, the probability that A queries r^* in this experiment is the same as the probability that it queries r^* in the actual attack.

However, in this experiment the only information A gets about r^* is $f(r^*)$ - thus if it queries $G(\cdot)$ the value r^* then it inverted the trapdoor permutation! □

Now this means we can ignore the probability that A queried r^* and hence we can (like in the proof of the claim) assume that $z^* = u \oplus x^*$ where u is chosen independently at random. However, this means that

A gets *no information* about x^* and hence will not be able to guess if it's equal to x^1 or x^2 with probability greater than $1/2$.

□

The CCA secure encryption First note that if we have one random oracle we can have many independent oracles (just have $G_i(x) = G(i \circ x)$). We'll use two independent random oracles G, H in the next scheme.

- Let $G, H : \{0,1\}^n \to \{0,1\}^n$ be two independent random oracles and $\{(f, f^{-1})\}$ be collection of trapdoor permutations. The public key of the scheme will be $f(\cdot)$ while the private key be f^{-1}.
- To encrypt $x \in \{0,1\}^n$, choose $r \leftarrow_{\text{R}} \{0,1\}^n$ and compute $f(r), G(r) \oplus x, H(x,r)$.
- To decrypt y, z, w compute $r = f^{-1}(y)$ and let $x = r \oplus z$. Then, check that $w = H(x,r)$: if so then return x, otherwise return $\perp$.

Theorem 2. *The above scheme is CCA secure.*

Proof. Let A be an algorithm in a CCA attack against the scheme. Again, denote by y^*, z^*, w^* the challenge ciphertext A gets where $y^* = f(r^*)$, $z^* = G(r^*) \oplus x^*$ and $w^* = H(x^*, r^*)$.

Since H is a random oracle, we can assume that throughout the attack, no one (the sender, receiver or A) will ever find a two pairs x, r and x', r' such that $x \circ r \neq x' \circ r'$ but $H(x,r) = H(x',r')$.

Thus, at each step i of the attack and for every string $w \in \{0,1\}^n$ we can define $H_i^{-1}(w)$ in the following way: if the oracle H was queried before with some x, r and returned w then $H_i^{-1}(w) = (x,r)$. Otherwise, $H_i^{-1}(w) = \perp$.

We also observe that a pair x, r completely determines a ciphertext y, z, w that is a function of x and r and also that y, z completely determine x and r.

We consider the following experiment: at step i, we answer a decryption query y, z, w of A in the following way: if $H_i^{-1}(w)$ is equal to some x, r that determine y, z, w then return x. Otherwise, return $\perp$.

Note that the difference between this oracle and the real decryption oracle is that we may answer $\perp$ when the real decryption oracle would give an actual answer. However, we claim that A will not be able to tell apart with non-negligible probability the difference between this decryption oracle and the real one. Indeed, the only difference would be if A managed to ask the oracle a query: y, z, w satisfying the following:

- $w \neq w^*$ (since if $w = w^*$ then we have that $H_i^{-1}(w) = x^*, r^*$ and hence A either asked a query that both oracles answer with $\perp$ or it asked the disallowed query y^*, z^*, w^*).
- w was not returned as the answer of any previous query x, r to $H(\cdot)$ by A.
- If we let x, r be the values determined by y, z then $H(x, r) = w$. However, since (x, r) was not asked before, the probability that this happens is only 2^{-n}.

Thus, we see that we can simulate the decryption box of A without knowing f^{-1}, x^* and r^*. This means that A basically has no use for the decryption box and hence it would be sufficient to prove that the scheme is just CPA secure. This proof follows in a similar way to the previous scheme. $\square$

OAEP+ Have $f : \{0,1\}^{3n} \to \{0,1\}^{3n}$ a TDF, $H_1, H_2, H3$ random oracles, $Enc(m)$: choose $r \leftarrow_{\mathrm{R}} \{0,1\}^n$, $h = H_1(r) \oplus m$, $s = H_2(m, r)$, $t = H_3(h, s) \oplus r$, output $f(h, s, t)$