CS 346 Class Notes

Mark Lindberg

Apr 27, 2016

Last Time:

El Gamal type techniques to get a KEM that is CCA-secure under the random oracle model. Therefore, we have a CCA-secure public key encryption scheme for arbitrary length messages, with the discrete log assumption.

This Time:

Similar results based on the RSA assumption instead of the discrete log assumption.

11.5 RSA Encryption:

11.5.1 Plain RSA.

Construction 11.26?

Previously, we had $\mathsf{GenModulus}(1^n) \to (N, p, q)$.

GenRSA:

1. Run $\mathsf{GenModulus}$ to get (N, p, q).
2. Compute $\phi(N) = (p-1)(q-1)$. This is the order of the group $\mathbb{Z}_N^*$.
3. Choose e such that $(e, \phi(N)) = 1$.
4. Set $d = e^{-1} \mod \phi(N)$. (Can be computed in polynomial time using the Extended Euclidean Algorithm.)

Consider x^e, where $x \in \mathbb{Z}_N^*$. This is a bijection over $\mathbb{Z}_N^*$. Also, $(x^e)^d = x$ when working mod N.

This is because $x^{\phi(N)} = 1 \ \forall x \in \mathbb{Z}_N^*$.

"Plain RSA":

- Gen: Run $\mathsf{GenRSA}(1^n)$. $pk = (N, e)$, $sk = (N, d)$.
- Enc: On input (N, e) and message $m \in \mathbb{Z}_N^*$, ciphertext $c \leftarrow m^e$.
- Dec: On input (N, d) and ciphertext c, we have that $m = c^d$.

Note: This is a DETERMINISTIC Enc, so it cannot be CPA-secure. Oops.

Recall the "RSA assumption".

$\mathsf{RSA} - \mathsf{Inv}_{\mathcal{A},\mathsf{GenRSA}}(n)$:

- Run GenRSA, get (N, p, q).
- Choose a uniformly random $x \in \mathbb{Z}_n^*$.
- Compute $y = x^e$.
- Let $\mathcal{A}$ have N, e, y. $\mathcal{A}$ outputs x'.
- $\mathcal{A}$ succeeds if and only if $x' = x$.

The RSA assumption is that for some PPT GenRSA, $\forall$ PPT $\mathcal{A}$,

$$\Pr[\mathsf{RSA} - \mathsf{Inv}_{\mathcal{A},\mathsf{GenRSA}}(n) = 1] \leq \mathtt{negl}(n).$$

It is believed that the "vanilla" GenRSA that we have defined in the past is secure in this experiment.

Security of "Plain RSA". (Construction 11.26)

- Not CPA-secure. (Deterministic.)
- RSA assumption only implies that inversion is hard on average, not all the time.
- Often e is set to a small value ($e.g.3$) for fast encryption.
- If the message $< N^{\frac{1}{3}}$, it is easy to find x from x^3, because no modular arithmetic really occurs.

An attack which yields an approximately quadratic improvement over brute force, which does not rely on having a small e. It'll take $\sim N$ time.

A random n-bit number is equal to the product of two αn-bit numbers with good probability, where $\alpha = 0.51$, for example. A constant fraction of xs satisfy this attack.

x^e, where x is an n-bit value.

Assume $x = a \cdot b$, $a, b \leq 0.51n$ bits.

How can we compute x in $\sim 2^{.51(n)}$ time?

For each (r, α_r), compute $\alpha_r = r^e$. (Working over $\mathbb{Z}_N^*$.) This is the brute-force method, but only over the strings with $\leq .51$ bits.

Compute (r, β_r), where $\beta_r = x^e(\alpha_r)^{-1}$.

If $r = a$, then β_r, then $\beta_r = x^e/a^e = (ab)^e/a^e = b^e$.

So (a, b^e) is one of the pairs we compute...

We are going to sort all of the pairs by the second component.

For each $0.51n$-bit number s, check whether $s^e =$ some second component. (We can check this equality with binary searching.)

11.5.2: Padded RSA and PKCS#1 v. 1.5 (1993).

PKCS = Public Key Cryptographic System. It incorporates randomization into the encryption function.

We have approximately $||N||$-bit messages. The high-level idea is that we're going to take shorter messages, prepend a chunk of random bits, then encrypt. Decryption just has to chop off the random bits.

In the method described in the text, we work in bytes. The message length is variable, but this forces a lower bound on the number of random bytes. We will have at least 8 random bytes in this particular construction. (Technicality: To aid in parsing, the random bytes tend to be non-zero. Then, they're followed by a zero-byte, so we know how much to strip off. [There's more confusing stuff in the textbook, which prof didn't fully understand.])

This scheme is no longer used, because 8 is not a sufficient number of random bytes.

They should have required ∼half of the bits to be random. Turns out using 8 bytes ain't CPA-secure, and bwahaha, things go boom.

11.5.5: A CCA-secure KEM in the random oracle model.

Gen, Encaps, Decaps.

- Gen: Use GenRSA to obtain pk, sk.
- Encaps: On input $1^n, pk$, choose a random $r \in \mathbb{Z}_N^*$. Let $c = r^e$, and $k = H(n)$, where H is a random oracle.
- Decaps: Receive c, have sk. Then $r = c^d$, and $k = H(r)$.

CPA-security is easy to argue subject to the RSA assumption.

Theorem 11.38: It's actually CCA-secure.

11.5.3: CPA-secure encryption without random oracles.

Unfortunately, the results here are primarily of theoretical interest, simply because they are too inefficient to calculate and use.

1: CPA-secure encryption scheme for single-bit messages.

2: CPA-secure KEM.

Result 1 is based on Theorem 11.31.

Which, er, is probably important to state. Unfortunately, my group started messaging me about our presentation in an hour, and I stopped paying attention. Oops. It's in the textbook.