ICS 180: Introduction to Cryptography April 13, 2004

Lecture 3: One-Way Encryption, RSA Example

Lecturer: Stanislaw Jarecki

1 LECTURE SUMMARY

We look at a different security property one might require of encryption, namely *one-way security*. The notion is natural and seems like a minimal requirement on an encryption scheme. It makes sense for both symmetric and public-key encryption schemes. To make the discussion more concrete, we look at the so-called "textbook" variant of the RSA encryption, and see how to pick keys in relation to the security parameter so that the best algorithms that invert RSA are either inefficient or have only negligible advantage. We will also see that while the "textbook RSA" can plausibly be one-way secure, it is definitely not secure in the sense of indistinguishability (this security property of encryption schemes was defined in the last class). This shows us that one-wayness is a weaker notion than indistinguishability.

2 One-Way Security for Encryption

In the last lecture we developed the computational version (relaxation) of the *perfect secrecy* security property for encryption schemes, which we called *indistinguishability* of encryption. This notion is pretty strong, and today we'll look at a weaker notion of security for encryption, namely *one-way security*.

In essence, we say that an encryption scheme is one-way secure if it is infeasible to decrypt ciphertexts of random plaintexts (i.e. randomly chosen from a big-enough message space). Here is the formal definition, first for the case of symmetric encryption schemes:

Definition 1 (one-way secure (symmetric) encryption) *We call a (symmetric) encryption scheme* $\Sigma = (KGen, Enc, Dec)$ one-way secure for (family of) message spaces $\{\mathcal{M}_\tau\}_{\tau=1,2,...}$ *if for all PPT algorithms* A*, the following holds:*

$$Adv_A(\tau) \ = Prob[A(c) = m \mid k \leftarrow KGen(1^\tau); m \leftarrow \mathcal{M}_\tau; c \leftarrow Enc(k, m)] \ \leq \mathsf{negl}(\tau)$$

And here is the corresponding definition for public-key encryption schemes. The only real difference is that here the adversary sees the public key used to encrypt messages:

Definition 2 (one-way secure (public key) encryption) *We call a (public-key) encryption scheme* $\Sigma = (KGen, Enc, Dec)$ one-way secure for (family of) message spaces $\{\mathcal{M}_\tau\}_{\tau=1,2,...}$ *if for all PPT algorithms* A*, the following holds:*

$$Adv_A(\tau) \ = Prob[A(PK, c) = m \mid (SK, PK) \leftarrow KGen(1^\tau); m \leftarrow \mathcal{M}_\tau; c \leftarrow Enc(PK, m)] \leq \mathsf{negl}(\tau)$$

Discussion. The one-wayness of encryption seems to be a pretty minimal requirement needed of an encryption scheme. Suppose, on the contrary, that an encryption scheme is not one-way. This would mean that there exists an efficient algorithm A which has a non-negligible chance of success in decrypting an encryption of a random message. Notice that in any application of an encryption scheme, the encryption/decryption keys are going to be picked by a (random) run of the $KGen(1^\tau)$ algorithm, which is just like in the one-wayness game the adversary plays with an encryption scheme in the above definition(s). So if A's advantage in this game is "sizable", i.e. larger than negligible (larger than $1/p(\tau)$ for some polynomial $p(\cdot)$), this could mean one of the following things, each of them pretty bad for any application using such encryption:

- It could be that for every message $m \in \mathcal{M}$ and every $c = Enc(k, m)$, there is about $1/p(\tau)$ chance that A decrypts c as m, where the probability is taken over the random coins of A. This case is the worst, because in this case we could run A over an over on the same input c, and after $\tau * p(\tau)$ trials, the probability that in none of these trials A ends up decrypting c is only

  $$Prob[A \text{ fails in all } \tau * p(\tau) \text{ trials}] \approx (1 - 1/p(\tau))^{\tau * p(\tau)} \approx (1/e)^\tau < (1/2)^\tau = 2^{-\tau}$$

  Note also that the time taken by this new attack is $\tau * p(\tau) * Time_A(\tau)$ which is $poly(\tau)$ because $Time_A(\tau)$ is polynomial in τ.

  So such A leads to an efficient attack which decrypts every message except for a negligible probability $2^{-\tau}$.

- However, it could also be that the above is true only for some messages $m \in \mathcal{M}$. For example, it could be that A decrypts with probability almost 1 on any encryptions of all messages in some subset $\mathcal{M}' \subset \mathcal{M}$ which contains $1/p(\tau)$ fraction of messages, i.e. $|\mathcal{M}'|/|\mathcal{M}| = 1/p(\tau)$, but on encryptions of messages $m \notin \mathcal{M}'$ it fails.

  Why would such an attack be so bad? First of all, for many encryption schemes, an attack which succeeds well on some fraction of messages can be used to succeed on all messages. (We'll see that on the RSA example below!)

  Second, even if there was no way of transforming such attack into decrypting any message, this is still not good because the application would have to make sure that messages in $\mathcal{M}'$ are avoided. Whatever message space $\mathcal{M}'' = \mathcal{M} \setminus \mathcal{M}'$ an application needs to use, it needs to know that the encryption scheme is at least one-way secure for the message space $\mathcal{M}''$.

- The inverter A could work in yet another way. For example, maybe A has a good chance of inverting a particular class of ciphertexts, but fails to invert if the ciphertext does not fall into this class? This type of attack usually leads to an attack against any ciphertext too (similarly to the case above). An example of this in in problem 3 on homework 2. You'll see there that if there is an attack A which inverts ElGamal ciphertext only if they have some special (although common-enough) form, you can use A to construct an attack which actually inverts any ElGamal ciphertext.

3 Example: RSA cryptosystem

To understand the RSA cryptosystem, for now all you need to know is that $\mathbb{Z}_n = \{0, ..., n-1\}$. To understand it more more deeply, you should read the handouts on modular arithmetics (first the

handout on primes and then the one on composites). You'll see there what $\mathbb{Z}_n^*$ is, where did $\phi(n) = (p-1)(q-1)$ come from, and why equation (2) below holds for all $m \in \mathbb{Z}_n^*$ if $ed = 1 \bmod \phi(n)$. But at first you can just assume that $d = e^{-1} \bmod \phi(n)$ is easy to compute and that equation (2) does hold.

The scheme below is called a "textbook RSA" cryptosystem, because it uses the RSA setting in the most basic way possible.

- $KGen(1^\tau)$ generates two prime numbers p, q of length $|p| = |q| \approx \frac{1}{2c^3}\tau^3$ where c is a small constant, $c \approx 8$ (the actual formula for the good prime size is a little more complex but this is a good approximation, and we'll see later today how this is the choice is made!). Then $KGen$ computes a $n = p * q$, picks some small prime number $e \geq 3$ (in most applications $e = 3$ is used because the speed of encryption depends on the size of e, so the smallest possible e is picked), then computes number $d = e^{-1} \bmod \phi(n)$ where $\phi(n) = (p-1)(q-1)$ (computing modular inverses is efficient via the Euclidean algorithm), and outputs $PK = (n, e)$ and $SK = (p, q, n, e, d)$.

- $Enc(PK, m)$ for $m \in \{0,1\}^*$ first divides m into $|m|/(|n|-1)$ blocks $m = [\overline{m}^{(1)}|\overline{m}^{(2)}|...]$ of $|n|-1$ bits each (the last block can be padded with zeroes to make it $|n|-1$ bit long). This way each $m^{(i)}$ is in $\mathbb{Z}_n$, and in fact it is almost always an element of $\mathbb{Z}_n^*$ too.[1] Then for each block $\overline{m}$ (assumed in $\mathbb{Z}_n^*$), Enc computes $\overline{c} = (\overline{m})^e \bmod n$. Then it collects the resulting ciphertext blocks and outputs $c = [\overline{c}^{(1)}|\overline{c}^{(2)}|...]$.

- The decryption $Dec(SK, c)$ goes block-by-block as well, where the block $\overline{m}$ of plaintext is computed from the corresponding ciphertext block $\overline{c}$ as $\overline{m} = (\overline{c})^d \bmod n$.

The reason that RSA encryption can work at all is the following fact, which holds for any n:

$$\forall\, m \in \mathbb{Z}_n^*, \quad m^{\phi(n)} = 1 \bmod n \tag{1}$$

From this equation it follows that for any e, d s.t. $ed = 1 \bmod \phi(n)$ we have:

$$\forall\, m \in \mathbb{Z}_n^*, \quad m^{e*d} = m \bmod n \tag{2}$$

It follows because if $m^{\phi(n)} = 1 \bmod n$ and $ed = i\phi(n) + 1$ for some integer i (because $ed = 1 \bmod \phi(n)$), then $m^{ed} = m^{i\phi(n)+1} = (m^{\phi(n)})^i * m^1 = 1 * m = m \bmod n$.

Equation (2) implies, taking $m \in \mathbb{Z}_n^*$ for notation simplicity, that $Dec(SK, Enc(PK, m)) = ((m^e \bmod n)^d \bmod n) = (m^{e*d} \bmod n) = m$.

3.1 The RSA Assumption: "Textbook RSA" is a One-Way Encryption

The so-called *RSA assumption* is basically exactly what the definition (2) states where $\Sigma = (KGen, Enc, Dec)$ is the above "textbook RSA" scheme and the message space $\mathcal{M}$ is chosen as $\mathbb{Z}_n^*$ (equivalently, one can assume RSA one-wayness on message space $\mathcal{M}_\tau = \{0,1\}^{\tau^3/c^3-1}$). In other words:

[1] We can treat any element in $1, ..., n-1$ as most likely an element of $\mathbb{Z}_n^*$ because finding elements m in $\mathbb{Z}_n \setminus \mathbb{Z}_n^*$ actually leads to factoring of n. The reason is that if $n \notin \mathbb{Z}_n^*$ then it must be that $gcd(n, m) \neq 1$, but since $n = pq$ and $m < n$ then $gcd(n, m)$ is either p or q, so you find a prime factor of n if you find $m \notin \mathbb{Z}_n^*$. Therefore finding such m must be unlikely if the factoring problem is hard.

Definition 3 (RSA Assumption) *It is infeasible, for large enough τ, to compute m s.t. $m^e = c \bmod n$ on inputs $((n,e),c)$, where (n,e) are RSA parameters chosen as above, $c = m^e \bmod n$, and m is picked at random in $\mathbb{Z}_n^*$.*

First, note that the above assumption indeed states exactly that the "textbook RSA" is a one-way secure encryption scheme on message space $\mathcal{M} = \mathbb{Z}_n^*$.

But what rationale do we have to think that the RSA assumption holds, or, equivalently, that the "textbook RSA" encryption is one-way secure? Let's look at some attempts A to break the one-wayness of the RSA encryption, and see that they all either fail to be polynomial time, e.g. $Time_A(\tau) = 2^\tau$, or they have only negligigible advantage in breaking the one-wayness property, e.g. $Adv_A(\tau) = 2^{-\tau}$.

Attempt 1. How about a trivial algorithm A which on input $((n,e),c)$ guesses factor $p \in \{0,1\}^{|n|/2}$ of n, and if the guess is correct, it computes $q = n/p$, $\phi(n) = (p-1)(q-1)$, $d = e^{-1} \bmod \phi(n)$, and then decrypts $m = c^d \bmod n$. While $Time_A$ is good because all these operations are efficient, the advantage Adv_A of A, i.e. the probability that A decrypts m succesfully is only negligible, because A succeeds if and only if it guesses p correctly, and the chances of that are $2^{-|p|} = 2^{-|n|/2} = 2^{-\tau^3/c^3}$, which is a negligible function of the security parameter τ.

Attempt 2. Alternatively, A could search the whole space of possible p's. Then A would succeed for sure, but since there are $2^{|n|/2} = 2^{\tau^3/c^3}$ possible p's, A would be run in exponential time, which is infeasible.

Attempt 3. The best known way to invert RSA encryption of random messages is indeed to first factor n and then proceed like the attackers above. However, the best factoring algorithm does it much faster than by searching the space of possible $|n|/2$-bit factors of n. The fastest algorithm is called a number-field sieve [NSF], and it takes time $Time_{\mathsf{NSF}} = O(2^{c*|n|^{1/3}})$. Therefore, since $|n| = (\tau/c)^3$ and hence $c*|n|^{1/3} = \tau$, and the attack time is dominated by the time it takes to factor n, we have $Time_A(\tau) \approx Time_{\mathsf{NSF}} = O(2^\tau)$. Thus again, the time it takes for A to invert an RSA encryption is exponential in τ.

This analysis explains why the length of the RSA composite n is chosen in the $|n| = (\tau/c)^3$ relation to the security parameter τ. It's so that an RSA cryptosystem chosen with security parameter τ indeed takes at least about 2^τ operations to break.

3.2 "Textbook RSA" is *Not* an Indistinguishably Secure Encryption

It is easy to see that while RSA can be plausibly one-way secure (see the failed attempts 1-3 above to break this assumption), it is *not* secure in the sense of indistinguishability of encryption.

Recall the definition of encryption scheme indistinguishability from lecture 2. To show that the "textbook RSA" encryption scheme fails this security property, we only need to show two messages m_0 and m_1, and an efficient adversary who distinguishes encryptions of m_0 from encryptions of m_1. But that's easy: Take any $\overline{m}, \overline{m'} \in \mathbb{Z}_n^*$ s.t. $\overline{m} \neq \overline{m}'$, and make $m_0 = [\overline{m}, \overline{m}]$ and $m_1 = [\overline{m}, \overline{m'}]$. This way $Enc((n,e), m_0) = [\overline{c}, \overline{c}]$ for some $\overline{c}$ while $Enc((n,e), m_1) = [\overline{c}, \overline{c}']$ where $\overline{c} \neq \overline{c'}$. (This holds because the RSA function $y = RSA_{(n,e)}(x) = m^e \bmod n$ is a *permutation* in $\mathbb{Z}_n^*$, and hence if $x \neq x'$ then $y \neq y'$.) Hence it is trivial to distinguish encryptions of m_0 from encryptions of m_1.

Why is this so bad? Because it means, for example, that in an application in which someone sometimes send the same message twice, which might very well happen with messages like "yes", or "no", or "snafu", the eavesdropper will be able to distinguish, for any two communicated messages, if they are ciphertexts of the same plaintext. This could very well be insecure and is for sure undesirable.

Note that the above argument implies something more general, namely that any *deterministic* encryption scheme cannot be secure in the sense of indistinguishability.

How is RSA used in practice to make it secure then? First of all, randomness is introduced so that to make the encryption non-deterministic. Instead of computing $c = m^e \bmod n$ for message block $m \in \mathbb{Z}_p^*$, the message is divided into smaller blocks m, about $\frac{3}{4} * |n|$-bits each, the encryption procedure picks a random string $r \in \{0,1\}^{\frac{1}{4}|n|}$, and then *pads* the message m using randomness r in quite an elaborate way, to create a preimage $m' = pad(m;r)$. Encryption then outputs $c = (m')^e \bmod n$, while decryption reverses this process, extracting (m,r) from $m' = c^d \bmod n$ and outputing m. We'll see later in classs how this padding is done and why.