Notes for Lectures 9–11

1 Public Key vs. Symmetric Encryption

In the encryption we've been doing so far, the sender and the recipient needed to preagree on a key. This is traditionally called "symmetric" or "secret-key" encryption.

The idea of public key encryption is that I can walk into a room, announce my key, and everybody in the room can send secret messages to me by simply shouting them out so that everyone can hear them. In other words, we can communicate secretly by publishing messages for everyone to see, even if we never pre-agreed on a shared secret value unknown to others. This sounds impossible, but can actually be done. The idea was first proposed by Diffie and Hellman [DH76].

2 Public Key Encryption Using the Squaring Function

First Attempt

We know that squaring modulo $n = pq$ is easy, while taking square roots is hard (as hard as factoring n). On the other hand, taking square roots modulo p and modulo q is easy, and so is recombining them using the Chinese Remainder Theorem. So this gives us an idea for the public-key encryption scheme.

Let $p \neq q$ be two primes, $n = pq$, $p \equiv q \equiv 3 \pmod 4$. Let the public key PK be n, and the secret key SK be (p, q). For $m \in QR_n$, define the encryption of m as $c = m^2 \bmod n$. Note that if you just see c, n, it's hard to find m, because it's hard to find square roots (as we proved last time). However, if you know p and q, you can take the square root of c modulo p and modulo q (as shown on HW2), and combine them using Chinese Remainder Theorem to get back m. Note that you have to make sure that you take the square roots that are themselves squares, in order to get m and not one of the other three roots of c.

This scheme can only encrypt messages in QR_n (otherwise you don't get unique decryption, because you don't know which of the four square roots to take). While it is believed to be hard to find out whether $m \in QR_n$ without knowing SK, there are various tricks to get around this problem and get unique decryption for all m. We won't discuss them here.

The idea of using modular squaring in this way is due to Michael Rabin [Rab79], and the squaring function modulo n is often called the *Rabin function*.

Why the First Attempt Isn't Secure

Note that if the adversary knows that m is small (less than $\sqrt{n}$), then c is just m^2, and m can be found by simply taking the integer square root of c. In addition, this encryption scheme cannot be used to encrypt the same message twice, because the adversary will be able to tell that two ciphertexts are the same. Nor can it securely encrypt two messages whose ratio is known. These examples (one can come with many more quite easily) suggest that we need to do better in order to achieve secure encryption.

More generally, remember that while we showed that modular squaring is hard to invert, we don't know that *all* bits of m are hard to recover from c. In particular, it is conceivable that much useful information about m can be computed from c, even if m itself cannot be. All we know from the previous lecture is that the last bit of m is well-hidden.

Single-Bit Scheme

As before, let's focus on encrypting a single bit for now. We'll figure out how to encrypt more later. Since we know the last bit of a modular square root is well-hidden, it suggests the following idea. Public key is still n, secret key is still (p, q). To encrypt a bit b, take a random $r \in QR_n$, compute $s = r^2 \bmod n$, let p be the last bit of r, and $c = p \oplus b$. Output (s, c). Essentially, we are encrypting b with the one-time pad that is hiddent by s, but can be recovered by someone who take square roots modulo n (i.e., who knows the secret key).

Intuitively, this seems secure: if one could tell apart the encryptions of 0 and 1, then one could, from c and s, figure out what p is. But it's hard to tell whether p is 0 or 1, as discussed in the previous lecture.

However, to formally prove this scheme secure, we first need to define security for public-key encryption.

3 Defining and Realizing Secure Encryption

3.1 Secure Single-Bit Encryption

Let's see what we could possibly want from secure public-key encryption by focusing on the single-bit case. That is, we want to encrypt either a 0 or a 1, and we don't want the adversary to know. In the information-theoretic symmetric-key case, we said that secure encryption means that for a fixed ciphertext c, $\Pr[\mathrm{Enc}_k(0) = c] = \Pr[\mathrm{Enc}_k(1) = c]$, where the probability is taken over the choice of k and random choices made by Enc.

We can't do the same in the public-key case, because once the public key is fixed, a given ciphertext c cannot be both an encryption of a 0 and a 1 (if some c was an encryption of both, how would you decrypt?!). The main advantage of symmetric encryption is that the key is hidden from the adversary, and only the ciphertext is visible—so it makes sense to talk about a randomly chosen k that the adversary cannot see. You can't do the same in the public-key case. Hence, we need a different definition, one that makes sense even after the adversary sees PK.

Well, since we can't say that encryption of 0 and encryption of 1 are identically distributed, we'll try for something that's almost as good from a computational point of view: we'll say they are indisitnguishable.

Before we can define security, however, we need to formalize the notion of public-key cryptosystem.

Definition 1. A *public-key cryptosystem* is a triple of polynomial-time algorithms $(\mathrm{Gen}, \mathrm{Enc}, \mathrm{Dec})$. $\mathrm{Gen}(1^k)$ is a (randomized) key-generation algorithm that outputs a key pair $(\mathrm{PK}, \mathrm{SK})$ when given a security parameter k as input. Enc is a (randomized) encryption algorithm that, on input PK and message m, outputs ciphertext c. Dec is a (usually deterministic) decryption algorithm, that, on input SK and c, outputs m. For public key PK, a cryptosystem has to specify a set of allowed messages M_{PK} (ultimately the goal will be to have M be all binary strings regardless of PK; however, we have to allow for less general encryption schemes at first). We require that the following holds: if $(\mathrm{PK}, \mathrm{SK})$ is produced by $\mathrm{Gen}(1^k)$, then for all $m \in M_{\mathrm{PK}}$, $m = \mathrm{Dec}_{\mathrm{SK}}(\mathrm{Enc}_{\mathrm{PK}}(m))$ (this requirement can be relaxed to say "with probability $1 - \mathrm{negl}(k)$").

Now we can say what security means for 1-bit crypsosytems. Let $(\mathrm{Gen}, \mathrm{Enc}, \mathrm{Dec})$ be a public-key cryptosystem, with $M_{\mathrm{PK}} = \{0, 1\}$ for all PK. Let D be a polynomial-time algorithm that attempts to distinguish an encryption of 0 from an encryption of 1, and outputs either 0 or 1. Consider the following two experiments:

`exp0`(k)
1. $(\mathrm{PK}, \mathrm{SK}) \leftarrow \mathrm{Gen}(1^k)$
2. $c \leftarrow \mathrm{Enc}_{\mathrm{PK}}(0)$
3. Output $D(1^k, \mathrm{PK}, c)$

`exp1`(k)
1. $(\mathrm{PK}, \mathrm{SK}) \leftarrow \mathrm{Gen}(1^k)$
2. $c \leftarrow \mathrm{Enc}_{\mathrm{PK}}(1)$
3. Output $D(1^k, \mathrm{PK}, c)$

Note that the two experiments differ only in the second line.

Definition 2 ([GM84]). A public-key cryptosystem for 1-bit messages is *polynomially-secure* if for all polynomial time D there exists a negligible function $\mathrm{negl}(k)$ such that

$$|\Pr[\texttt{exp0}(k) \text{ outputs } 1] - \Pr[\texttt{exp1}(k) \text{ outputs } 1]| \leq \mathrm{negl}(k).$$

This definition is often called *GM-security* or *indistinguishability-security.*

This definition, first proposed by Goldwasser and Micali in [GM84][1], is exteremly strong. As mentioned above, c and PK together contain enough information to tell whether 0 or 1 was encrypted. What the definition requires is that even though the information is there, one can't get to it in polynomial time!

Now that we have such a strong definition, can we achieve it? Recall the following claim from Lecture 8:

Claim 1 ([ACGS88, AGS03]). Assume that factoring Blum integers is hard. Then, given the pair $(n, r^2 \bmod n)$ for n a Blum integer and $r \in QR_n$, the least significant bit of r (i.e., $r \bmod 2$) is unpredictable. More formally, for every polynomial-time algorithm P there exists a negligible function $\mathrm{negl}(k)$ such that for all k,

$$\Pr[P(n, r^2 \bmod n) = (r \bmod 2)] \leq 1/2 + \mathrm{negl}(k),$$

where the probability is taken over the random generation of n as a product of two k-bit primes that are 3 modulo 4, a random choice of $r \in QR_n$, and random choices (if any) made by P.

So now, consider the above cryptosystem for one-bit messages. Gen just generates a random Blum integer n. $\mathrm{Enc}_{\mathrm{PK}}(m)$ is as follows: select a random $r \in QR_n$, let $p = r \bmod 2$ be its least significant bit. Compute $(s = r^2 \bmod n, c = m \oplus p)$. $\mathrm{Dec}_{\mathrm{SK}}((s, c))$ finds r from s by taking a square root, takes its least significant bit p and outputs $m = p \oplus c$. We will call it *bit-by-bit Rabin.*

Theorem 1. *Bit-by-bit Rabin is GM-secure under the assumption that factoring Blum integers is hard.*

[1] An early version of the paper appears in the ACM Symposium on the Theory of Computing (STOC), 1982

Proof. Suppose not. Then let D be a distinguisher for it, such that

$$|\Pr[\texttt{exp0}(k) \text{ outputs } 1] - \Pr[\texttt{exp1}(k) \text{ outputs } 1]| \geq \epsilon(k).$$

for not negligible function $\epsilon(k)$.

First, note that we can get rid of absolute value one way or the other for infinitely many k, so we can assume that either

$$\Pr[\texttt{exp0}(k) \text{ outputs } 1] - \Pr[\texttt{exp1}(k) \text{ outputs } 1] \geq \epsilon'(k).$$

or

$$\Pr[\texttt{exp1}(k) \text{ outputs } 1] - \Pr[\texttt{exp0}(k) \text{ outputs } 1] \geq \epsilon'(k).$$

for not negligible function $\epsilon'(k)$. Let's stick with the second one without loss of generality.

Then build a predictor to violate Claim 1 as follows: P on input (n^2, s) chooses a random bit c, runs $D(1^k, n, (s, c))$ (where k is the length of n) to get a bit b and outputs $b \oplus c$. Then probability that P is correct is as follows: in half the cases, namely when $(r \bmod 2) \oplus c = 0$, P will be correct with the same probability as $\Pr[\texttt{exp0}(k) \text{ outputs } 0] = 1 - \Pr[\texttt{exp0}(k) \text{ outputs } 1]$. In the other half, namely when $(r \bmod 2) \oplus c = 1$, P will be correct with the same probability as $\Pr[\texttt{exp1}(k) \text{ outputs } 1] = 1$. Thus, in total, P will be correct with probability

$$\begin{aligned}
&\frac{1}{2}\left(\Pr[\texttt{exp0}(k) \text{ outputs } 0] + \Pr[\texttt{exp1}(k) \text{ outputs } 1]\right) = \\
&\frac{1}{2}\left(1 - \Pr[\texttt{exp0}(k) \text{ outputs } 1] + \Pr[\texttt{exp1}(k) \text{ outputs } 1]\right) \geq 1/2 + \epsilon'(k)/2,
\end{aligned}$$

which is not negligible. □

3.2 Secure Multi-Bit Encryption

Of course, we often want to send more than one bit. We need to define what we mean by security in that case, and construct an encryption scheme.

To define security for a multi-bit case, note, first of all, that as messages get longer, ciphertexts must necessarily get longer. Hence, encryption cannot hide message length. We will require it to hide everything else, though. Namely, we will allow the adversary (distinguisher) to choose any two messages, m_0 and m_1 after seeing the public key, then encrypt either one of them, and let the distinguisher guess which one. More precisely, D now runs in two stages. In the first stage, on input PK, D outputs (m_0, m_1). In the second stage, on input c, which is an encryption of either m_0 or m_1, D tries to guess which one it is by outputting either 0 or 1. D is allowed to keep state between stages (so you need not give it PK the second time, or give it back m_0 and m_1). The experiments are now as follows:

`exp-`m_0(k)
1. $(\mathrm{PK}, \mathrm{SK}) \leftarrow \mathrm{Gen}(1^k)$
2. $(m_0, m_1) \leftarrow D(1^k, \mathrm{PK})$; if $|m_0| \neq |m_1|$, abort
3. $c \leftarrow \mathrm{Enc}_{\mathrm{PK}}(m_0)$
4. Output $D(c)$

$\texttt{exp-}m_1(\text{k})$
1. $(\text{PK}, \text{SK}) \leftarrow \text{Gen}(1^k)$
2. $(m_0, m_1) \leftarrow D(1^k, \text{PK})$; if $|m_0| \neq |m_1|$, abort
3. $c \leftarrow \text{Enc}_{\text{PK}}(m_1)$
4. Output $D(c)$

Note that the experiments differ only in the third line. The definition is still the same:

Definition 3 ([GM84]). A public-key cryptosystem is *polynomially-secure* if for all polynomial time D there exists a negligible function $\text{negl}(k)$ such that

$$|\Pr[\texttt{exp-}m_0(k) \text{ outputs } 1] - \Pr[\texttt{exp-}m_1(k) \text{ outputs } 1]| \leq \text{negl}(k).$$

Advanced Comment: The experiments require D to output m_0 and m_1. Because D is polynomial-time and gets only PK as input, m_0 and m_1 can only be things that are computable from PK as input. In particular, therefore, this definition says nothing about encrypting SK or anything derived from it. Not that it should matter, because people who encrypt shouldn't know SK, anyway.

How do we build such a public-key cryptosystem? Intuitively, the public-key case is quite different here from the information-theoretically secure encryption we considered earlier. In that case, if you encrypt two bits with the same one-bit key, you clearly give some more information to the adversary (in particular, if the adversary knows the second bit, he deduce the key and hence the first bit). This is not the case in public-key encryption, because the adversary knows the encryption key, and hence can himself encrypt as many bits as he wants. If this helps him to decrypt the first bit, then the one-bit encryption of the first bit must insecure. Therefore, it seems that simply repeating the encryption bit-by-bit with the same key will work in the public-key case.

Indeed, let $(\text{Gen}, \text{Enc}, \text{Dec})$ be a single-bit cryptosystem. Then let $(\text{Gen}', \text{Enc}', \text{Dec}')$ be the following: $\text{Gen}' = \text{Gen}$; for an l-bit message $m = m^1 m^2 m^3 \ldots m^l$, $Enc'_{\text{PK}}(m) = \left(\text{Enc}_{\text{PK}}(m^1), \text{Enc}_{\text{PK}}(m^2), \ldots, \text{Enc}_{\text{PK}}(m^l)\right)$; and for ciphertext $c = (c^1, c^2, \ldots, c^l)$, $\text{Dec}'_{\text{SK}}(c) = \text{Dec}_{\text{SK}}(c^1) \circ \text{Dec}_{\text{SK}}(c^2) \circ \ldots \circ \text{Dec}_{\text{SK}}(c^l)$, where $\circ$ denotes concatenation.

Theorem 2. *If* $(\text{Gen}, \text{Enc}, \text{Dec})$ *is polynomially-secure, then* $(\text{Gen}', \text{Enc}', \text{Dec}')$ *is also polynomially secure.*

Proof. Let D be a distinguisher for $(\text{Gen}', \text{Enc}', \text{Dec}')$ and ϵ be some funciton such that

$$|\Pr[\texttt{exp-}m_0(k) \text{ outputs } 1] - \Pr[\texttt{exp-}m_1(k) \text{ outputs } 1]| \geq \epsilon(k).$$

The proof is by hybrid argument. Let $\texttt{exp}^i$ be the same as $\texttt{exp-}m_0$ and $\texttt{exp-}m_1$ for steps 1, 2 and 4, but different in step 3 as follows: encrypt neither m_0 nor m_1, but a message consisting of the first i bits of m_0 followed by the last $l - i$ bits of m_1. Let

$$D_i(k) = \left|\Pr[\texttt{exp}^{i-1}(k) \text{ outputs } 1] - \Pr[\texttt{exp}^i(k) \text{ outputs } 1]\right|.$$

Then, by the usual triangle inequality, $\sum_{i=1}^{l} D_i(k) \geq \epsilon(k)$.

Now consider yet another family of experiments. Let $\texttt{exp}^i_0$ be the same as $\texttt{exp}^i$, except the i-th bit of the plaintext is replaced by 0 before encryption. Let $\texttt{exp}^i_1$ be the same as $\texttt{exp}^i$, except the i-th bit of the plaintext is replaced by 1 before encryption. Let

$$D'_i = \left|\Pr[\texttt{exp}^i_0(k) \text{ outputs } 1] - \Pr[\texttt{exp}^i_1(k) \text{ outputs } 1]\right|.$$

Note that $D'_i \geq D_i$, because in D'_i the i-th bit is guaranteed to differ, but D_i it merely *may* differ (only if m_0 and m_1 differ in the i-th bit)[2].

Now, finally, we build the following distinguisher $\tilde{D}$ for the original 1-bit encryption scheme. On input $\tilde{c}$, which is a ciphertext of either 0 or 1, run D on $(1^k, \mathrm{PK})$ to get m_0, m_1, pick i at random between 1 and l, and construct c by encrypting the first $i-1$ bits of m_0, followed by the input ciphertext $\tilde{c}$, followed by encryptions of $l-i$ bits of m_1. Run $D(c)$ and output whatever it does. Now if plug in this $\tilde{D}$ into `exp0` and `exp1`, we get

$$|\Pr[\texttt{exp0}(k) \text{ outputs } 1] - \Pr[\texttt{exp1}(k) \text{ outputs } 1]| = \frac{1}{l}\sum_{i=1}^{l} D'_i \geq \frac{1}{l}\sum_{i=1}^{l} D_i \geq \epsilon(k)/l.$$

Hence, if ϵ is not negligible, the original one-bit encryption scheme (Gen, Enc, Dec) is insecure, which is a contradiction. □

Thus, we have a way to encrypt arbitrary messages. However, it's not very efficient in terms of time, because each bit needs to be encrypted and decrypted separately (in the case of Rabin, this means one modular squaring per bit to encrypt, and essentially one modular exponentiation per bit to decrypt). In addition, it's very inefficient in terms of space, because each bit must get expanded in the ciphertext (because the one-bit encryption scheme cannot be deterministic if it is to be secure). Thus, for example, if we are using Rabin with 1024-bit keys, bit-by-bit encryption will expand the message by a multiplicative factor of 1024. We'd like to improve this.

4 Blum-Goldwasser Encryption

Recall the Blum-Blum-Shub pseudorandom generator. Let $n = pq$ be a product of two primes, each 3 modulo 4, and let $x \in QR_n$ be a square modulo n. Let $x_0 = x, x_1 = x_0^2 \bmod n, x_2 = x_1^2 \bmod n \ldots, x_l = x_{l-1}^2 \bmod n$. From each x_i, take the least significant bit b_i to get a pseudorandom string.

If you go through the proof of pseudorandomness we did for the Blum-Micali generator and a similar one for Blum-Blum-Shub, you'll notice that one can prove that each b_{i-1} is unpredictable given not only $b_i, b_{i+1}, \ldots, b_l$, but, in fact, given $x_i, x_{i+1}, \ldots, x_l$. Moreover, it's not hard to show from there that the pair $(b_0 b_1 \ldots b_{l-1}, x_l)$ is indistinguishable from the pair $(r_0 r_1 \ldots r_{l-1}, x_l)$, where r_i are completely random bits.

Thus, the idea of Blum-Goldwasser [BG84] encryption is as follows. The public key is n, as above. The secret key is (p, q). To encrypt m of length l, choose a random $x \in QR_n$ (by choosing a random element in $\mathbb{Z}_n^*$ and squaring it), generate $b_0 b_1 \ldots b_{l-1}$ and $x_l = x^{2^l}$ as above, and use $b_0 b_1 \ldots b_{l-1}$ as a one-time pad to exlusive-or with m to get c. Output (c, x_l) as the ciphertext. To decrypt, use p and q and Chinese Remainder Theorem to compute l square roots of x_l to get back x_0; reproduce the pad $b_0 b_1 \ldots b_{l-1}$; and exlusive-or it with c to get m.

Why is this secure? We won't do a full proof, but the intution is that encryption of any message m is indistiguishable from simply $(r_0 r_1 \ldots r_{l-1}, x_l)$, where r_i are random bits. Therefore, encryption of m_0 and encryption of m_1 are indistinguishable, by triangle inequality with $(r_0 r_1 \ldots r_{l-1}, x_l)$ as the midpoint. Notice that security is under the factoring assumption (rather than the stronger RSA assumption).

[2] To prove this formally, write $\Pr[\texttt{exp}^i(k) \text{ outputs } 1] = \sum_{\mathrm{PK}} \Pr[\mathrm{Gen}(1^k) \to \mathrm{PK}] \sum_{(m_0,m_1)} \Pr[D(1^k, \mathrm{PK}) \to (m_0, m_1)] \Pr[D(\mathrm{Enc}_{\mathrm{PK}}(m_0^1 m_0^2 \ldots m_0^i m_1^{i+1} m_1^{i+2} \ldots m_1^l)) \to 1]$. Similarly for $\Pr[\texttt{exp}^{i-1}(k) \text{ outputs } 1]$. Subtract the two and split the sum into four cases, depending on the i-th bits of m_0 and m_1. Two of the cases (00 and 11) will add nothing to the sum, and other two will get upperbounded by D'_i.

4.1 Efficiency

A opposed to bit-by-bit RSA encryption, the ciphertext is longer than the message by only an additive amount, the length k of n. This is much better than the multiplicative expansion of bit-by-bit encryption, of course. If we are using a 1024-bit modulus, for example, then no matter how long the plaintext is, the ciphertext will be longer by only 128 bytes.

Encryption takes one modular squaring per bit encrypted, or $O(lk^2)$ operations (because a modular squaring or multiplication takes $O(k^2)$). Decryption is trickier to figure out: it calls for taking square roots l times. On HW 4 you figured out how to take the root of degree 2^l in time $O(k^3 + k^2 \log l)$. Once you do this to get x_0, you can just re-run the Blum-Blum-Shub generator to get the one-time pad back in time $O(lk^2)$, for total decryption running time $O(k^3 + k^2 \log l)$.

Thus, in terms of time, this is as efficient for $l < k$ as "plain" (insecure) RSA that is often used in practice (described below). It beats plain RSA for $l > k$, because plain RSA cannot encrypt messages length greater than k.

5 RSA

5.1 The RSA function

RSA encryption [RSA78] is essentially the same thing as the Rabin function, but replaces squaring with raising to another power. Let's consider first raising to the power e modulo a prime p. Suppose $d = e^{-1} \bmod (p-1)$. Let $a \in \mathbb{Z}_p$. Since $ed \equiv 1 \pmod{p-1}$, by HW2 (where we proved exponents work modulo $p-1$), $a^{ed} \equiv a^1 \equiv a \pmod{p}$.

Now doing this modulo n, suppose $ed \equiv 1 \pmod{p-1}$ and $ed \equiv 1 \pmod{q-1}$. Then if we let $a \in \mathbb{Z}_n$, a^{ed} is a both modulo p and modulo q, and hence is a modulo n, by Chinese Remainder Theorem.

So pick two primes $p \neq q$ (not necessarily 3 modulo 4 anymore), let $n = pq$, and let e, d be such that $ed \equiv 1 \pmod{p-1}$ and $\pmod{q-1}$. Note that because e has an inverse modulo $p-1$ and $q-1$, it must be relatively prime with $p-1$ and $q-1$; in particular, e must be odd.

Let (n, e) be the public key and (n, d) be the corresponding secret key. To "encrypt" $m \in Z_n$, compute $c = m^e \bmod n$, and to decrypt c, compute $c^d \bmod n = (m^e)^d \bmod n = m^{ed} \bmod n = m$. We put "encrypt" in quotes because we haven't yet said anything about its security.

Common notation: $\phi(n) = |\mathbb{Z}_n^*| = (p-1)(q-1)$ ($\phi(n)$ is known as the ϕ function). $\lambda(n)$ is the least common multiple of $p-1$ and $q-1$. Most textbooks describe d as $e^{-1} \bmod \phi(n)$; however, $e^{-1} \bmod \lambda(n)$ suffices (and is sligtly smaller since $\lambda(n)|\phi(n)$ and hence $\lambda(n) < \phi(n)$).

RSA can be used on any message in Z_n, not just in QR_p as the Rabin function. However, we give up the equivalence to factoring: it is not known whether taking e-th roots modulo n is as hard as factoring for odd e. To be precise, we know that if taking e-th roots is hard, then factoring is hard (because if factoring were easy, then we could take e-th roots by taking them modulo p and q and combining them using CRT). The other direction is not known. We do know, however, that finding d from e is as hard as factoring (no proof in these notes). So, if there is a way to find e-th roots without factoring, it must not find d.

5.2 RSA Assumption

We haven't addressed the problem of generating RSA keys. That depends on whether you want a fixed e or a random e. People often use fixed small e (such as 3, 17, or $2^{16} + 1 = 65537$), because exponentiation is particularly fast. (There have been some questions raised, however, about the security of this approach—it may be possible that roots of small degree are easier to take than roots of a random degree. No one knows.) The fixed e case is as follows:

Generate-Fixed-Exponent-RSA
1. On input k in unary and e, generate random $k/2$-bit prime p repeatedly until $\gcd(p-1, e) = 1$; similarly for q.
2. Let $l = \text{lcm}(p-1, q-1)$
3. Let $d = e^{-1} \bmod l$ (this can be done using Euclid's extended GCD algorithm).
4. Output $\text{PK} = (n, e)$ and $\text{SK} = (n, d)$

The random e case is a follows:

Generate-Random-Exponent-RSA
1. On input k in unary, generate random $k/2$-bit primes p and q.
2. Let $l = \text{lcm}(p-1, q-1)$.
3. Select a random e between 1 and l repeatedly until $\gcd(e, l) = 1$.
4. Let $d = e^{-1} \bmod l$ (this can be done using Euclid's extended GCD algorithm).
5. Output $\text{PK} = (n, e)$ and $\text{SK} = (n, d)$.

Depending on which key generation procedure you use, you get a different security assumption.

Assumption 1. (Fixed-Exponent RSA for exponent e.) For any poly-time algorithm F, there exists a negligible function negl such that, if you generate RSA public key (n, e) according to the procedure **Generate-Fixed-Exponent-RSA**, then pick a random $m \in \mathbb{Z}_n^*$, $\Pr[F(n, e, m^e \bmod n) = m] \le \text{negl}(k)$.

Assumption 2. (Random-Exponent RSA.) For any poly-time algorithm F, there exists a negligible function negl such that, if you generate RSA public key (n, e) according to the procedure **Generate-Random-Exponent-RSA**, then pick a random $m \in \mathbb{Z}_n^*$, $\Pr[F(n, e, m^e \bmod n) = m] \le \text{negl}(k)$.

5.3 Secure RSA Encryption

We need the following claim (notice that we already saw a similar claim for modular squaring).

Claim 2 ([ACGS88, AGS03]). If the assumption Random-Exponent-RSA above holds, then given the triple $(n, e, x^e \bmod n)$, the least significant bit of x (i.e., $x \bmod 2$) is unpredictable. I.e., for every polynomial-time algorithm P there exists a negligible function $\text{negl}(k)$ such that for all k,

$$\Pr[P(n, e, x^e \bmod n) = (x \bmod 2)] \le 1/2 + \text{negl}(k),$$

where the probability is taken over the random generation of (n, e) by the algorithm **Generate-Random-Exponent-RSA** above on input k, a random choice of $x \in Z_n^*$, and random choices (if any) made by P.

We do not prove the claim here. Note that the same claim also holds if we replace random exponent RSA by fixed exponent RSA (and the discussion below does as well).

So now, consider the following cryptosystem for one-bit messages. Gen is just **Generate-Random-Exponent-RSA**. $\text{Enc}_{\text{PK}}(m)$ is as follows: select a random $x \in Z_n$ with last bit equal to m, and compute $c = x^e \bmod n$. $\text{Dec}_{\text{SK}}(c)$ simply outputs $(c^d \bmod n) \bmod 2$. We will call it *bit-by-bit RSA*.

Theorem 3. *Bit-by-bit RSA is GM-secure under the Random-Exponent-RSA assumption.*

The proof is essentially the same as for bit-by-bit Rabin, and we do not repeat it here.

5.4 RSA Encryption in Practice

You can view the bit-by-bit RSA as follows. To encrypt a one-bit plaintext m, pad m with $k-1$ random bits to get $x \in Z_n$. Then apply the RSA function.

When RSA was invented in 1978, the modern notions of secure encryption did not exist. However, people understood the dangers of deterministic encryption that we already described. Hence, they suggested using random padding. Of course, the idea of encrypting just one bit per modular exponentiation is not practical. The actual RSA encryption scheme used in practice has a lot more plaintext bits and a lot less padding.

In fact, the most common way to use RSA until recently has been a standard known as PKCS #1 version 1.5 [RSA93]. To encrypt a message m, it specifies that one should pad it to the length of the modulus by prepending a zero byte, byte of value 2, at least eight (and as many as needed) random non-zero bytes, followed by another zero byte to separate the pad from the message itself. The resulting bit string gets exponentiated to the public exponent e modulo n.

There is little one can prove about this scheme, although recently Jonsson and Kaliski [JK02] proved its security in certain applications under a relatively strong assumption. At some point it was believed to be not only polynomially secure, but, in fact, secure even against chosen-ciphertext attacks (attacks where adversary gets to see decryptions of a few ciphertexts, which we will discuss in more detail later). However, Bleichenbacher [Ble98] found a reasonably practical chosen-ciphertext attack against it. At that time, version 2.0 of PKCS #1 was in the works; currently the most recent version is 2.1. Both 2.0 and 2.1 can be proven not only semantically secure, but also secure against chosen-ciphertext attacks, in a special (unrealistic) model known as "random oracle model." Whether a proof in such a model is actually meaningful is a matter of some debate; we'll consider this subject later in the course.

References

[ACGS88] W. Alexi, B. Chor, O. Goldreich, and C. Schnorr. RSA and Rabin functions: Certain parts are as hard as the whole. *SIAM Journal on Computing*, 17(2):194–209, April 1988.

[AGS03] Adi Akavia, Shafi Goldwasser, and Muli Safra. Proving hardcore predicates using list decoding. In *44th Annual Symposium on Foundations of Computer Science*, Cambridge, Massachusetts, October 2003. IEEE.

[BG84] Manuel Blum and Shafi Goldwasser. An efficient probabilistic public key encryption scheme which hides all partial information. In G. R. Blakley and David Chaum, editors, *Advances in Cryptology: Proceedings of CRYPTO 84*, volume 196 of *Lecture Notes in Computer Science*, pages 289–302. Springer-Verlag, 1985, 19–22 August 1984.

[Ble98] Daniel Bleichenbacher. Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1. In Hugo Krawczyk, editor, *Advances in Cryptology—CRYPTO '98*, volume 1462 of *Lecture Notes in Computer Science*, pages 1–12. Springer-Verlag, 23–27 August 1998.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.

[GM84] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, April 1984.

[JK02] Jakob Jonsson and Burton S. Kaliski, Jr. On the security of RSA encryption in TLS. In Moti Yung, editor, *Advances in Cryptology—CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 127–142. Springer-Verlag, 18–22 August 2002.

[Rab79] Michael O. Rabin. Digitalized signatures and public-key functions as intractable as factorization. Technical Report MIT/LCS/TR-212, Massachusetts Institute of Technology, Cambridge, MA, January 1979.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[RSA93] PKCS #1: RSA encryption standard. Version 1.5, November 1993. Available from `http://www.rsaisecurity.com/rsalabs/pkcs/`.