Designing Secure Cryptography

Cornell CS 6831

1 Public Key Cryptography

1.1 Motivation

So far, we have focused our study on symmetric primitives. That is, primitives where two (or multiple) parties agree on a shared key, which is used for both encryption and decryption, or correspondingly, creating tags and verifying them, etc. While we have made much progress on such techniques in terms of security and efficiency, the fact that keys are shared raises several problems.

For one, the fact that everyone who has the same key can read all messages encrypted using it. For instance, if Alice creates a secret key, shares it with both Bob and Marley, then uses a symmetric encryption scheme to encrypt messages to both Bob and Marley with the same key, Alice and Bob's communication is not private from Marley and vice versa for Alice and Marley's. This means, she now has to create, and store high entropy keys for every person she wants to communicate with.

The above scenario doesn't even consider the fact that the keys need to be shared with every person in a secure way. This completely rules out communication with new entities only via the internet. Or even changing compromised shared keys.

Later, we will study digital signatures which allow a user Alice, to cryptographically sign a document in a way that is verifiable by anyone. Non-digital signatures are verified by looking at previously known signatures of Alice, which are presumably hard to copy. However, symmetric keys are easy to copy once known, so they cannot be used in this scenario either.

These kinds of issues motivate the need for a system in which Alice can have a secret known only to her, and corresponding public data, which allows other users to send her messages but not see them. Or, to verify her cryptographic signature but not forge it. In this sense, these primitives fall under *asymmetric* cryptography.

1.2 Background on Computational Number Theory

1.2.1 Basic Definitions

Asymmetric cryptography is based on underlying problems believed to be computationally hard – particularly interesting ones being number theoretic. Before we can dive into asymmetric primitives, we will need to define pre-requisite number theory concepts.

Definition 1 (Group)**.** *A group is a set G along with an operation $* : G \times G \to G$ such that:*

- *(Closure) Given $a, b \in G$, $a * b \in G$.*
- *(Associativity) For all $a, b, c \in G$ $a * (b * c) = (a * b) * c$.*
- *(Identity) There exists an element, let us call it $e \in G$, such that, for all $a \in G$ $e*a = a = a*e$.*

- *(Inverses) For all $a \in G$, there exists an element, let us denote it $a^{-1} \in G$, such that $a * a^{-1} = e$, where e is the identity.*

While one can denote the group from the definition $(G, *)$, in situations where the operation is understood from context, we will just denote it as G and call it the *group* G. Also, we denote

$$a^k = \underbrace{a * ... * a}_{k \text{ times}}$$

for any group element where $*$ is the group operation.

If you have not encountered this definition before, it would be useful to verify some examples of groups, such as $(\mathbb{Z}, +)$ (where $\mathbb{Z}$ denotes the set of integers), $(\mathbb{Q} \setminus \{0\}, \times)$ (where $\mathbb{Q}$ is the set of rational numbers), the groups defined below and see how they fit the definition.

Next, we define a specific type of group we will be interested in and show examples which will be needed to learn about the RSA and El Gamal crypto-systems.

If N is a positive number and let $gcd(i, N)$ denote the greatest common divisor of i, N. We also denote the remainder when a is divided by N as $a \mod N$, and we say

$$a \equiv b \mod N \text{ if } (a \mod N) = (b \mod N).$$

Note that for the usual addition and multiplication in $\mathbb{Z}$,

$$(a \times b) \mod N = (a \mod N) \times (b \mod N) \mod N$$

and the same for addition. For example, if $N = 5$, $12 \times 13 = 156 \equiv 1 \mod N$, and $12 \equiv 2 \mod 5$, $13 \equiv 3 \mod 5$ and $2 \times 3 = 6 \equiv 1 \mod 5$. You can see that the same holds for addition.

Definition 2 (Multiplicative group of integers modulo N)**.** *Given a positive integer N, the set $\mathbb{Z}_N^* = \{i : gcd(i, N) = 1 \text{ and } 1 \leq i < N\}$ along with the operation $*$ defined as $a * b = a \times b \mod N$ where $\times$ is the usual integer multiplication. We call N the modulus.*

Moving forward, we will just refer to the multiplicative group of integers modulo N as just $\mathbb{Z}_N^*$.

Definition 3 (Euler Totient Function)**.** *The Euler Totient function, denoted $\phi(N)$ is the size of Z_N^*.*

In particular, when we consider Z_p^* for prime p, the number of integers co-prime to it and less than it are $p - 1$, and hence $\phi(p) = p - 1$. In the case of an integer $N = pq$ for primes p and q, $\phi(N) = (p - 1)(q - 1)$.

Theorem 4 (Euler's Theorem)**.** *Given any element $a \in \mathbb{Z}_N^*$, $a^{\phi(N)} \equiv 1 \mod N$.*

We omit the proof for space reasons and urge the reader to attempt it or refer to an online source, such as Wikipedia.

1.2.2 Computational Tools

In order to make use of any of these number theoretic structures, we need efficient algorithms to compute the basic operations. We measure these values for computation in $\mathbb{Z}_N^*$ in terms of the bit length of N (the modulus). The worst case running times of known algorithms are given in Table 1.

Algorithm	**Running Time** ($n = \log N$)
Modular Multiplication ($ab \mod N$)	$\mathcal{O}(n^2)$
Modular Exponentiation ($a^i \mod N$)	$\mathcal{O}(n^3)$
Modular Inverses ($a^{-1} \mod N$)	$\mathcal{O}(n^2)$

Table 1: Algorithmic Complexity for Operations in $\mathbb{Z}_N^*$

In order to get the remainder of a number c modulo N, we can use the simple division algorithm from grade school. The Extended Euclidean Algorithm (EX-EUC) is used to find the inverse of an element in $\mathbb{Z}_N^*$ – if you have not seen this algorithm before, it is worthwhile to look it up and think about why it works to find an inverse. In the worst case, both these algorithms run in $\mathcal{O}(|a| \times |b|)$ where a and b are the inputs (can you show why?). In the case of an RSA modulus N, this means the worst case time for these operations is $\mathcal{O}(n^2)$ where $n = |N| = \log N$.

Multiplication in $\mathbb{Z}_N^*$ is done as we learnt in grade school, and then we get the remainder with respect to N which results in the running time shown in Table 1.

The most interesting operation in $\mathbb{Z}_N^*$ is actually exponentiation. To do it naively as we learnt in school would mean in the worst case, this would take exponential time in n. However, if instead, we observe that a number x in binary form $b_n...b_1$ is actually

$$x = 2^{n-1}b_n + ... + 2^0 b_1$$

then we find that FAST-POW , described in Algorithm 1 correctly implements $a^x \mod N$ for $a \in \mathbb{Z}_N^*$.

Algorithm 1 FAST-POW (a, x)

1: If $x < 0$, $a \longleftarrow a^{-1}$ and $x \longleftarrow -n$.
2: Let $b_k...b_0$ be the binary representation of x and $g \longleftarrow 1$.
3: **for all** $i = k...0$ **do**
4: $\quad g \longleftarrow g^2$
5: $\quad$ If $b_i = 1$, $g \longleftarrow g \times a \mod N$.
6: **end for**
7: **return** g

A way to see that Algorithm 1 indeed returns the correct value of $a^x \mod N$, is as follows. Define the sequence

$$f_j = \sum_{i=j}^{k} 2^{i-j} b_i.$$

Then,

$$f_{j-1} = \sum_{i=(j-1)}^{k} 2^{i-j+1} b_i = 2 \sum_{i=j}^{k} 2^{i-j} b_i + b_{j-1}.$$

Observe that, $x = f_0$ and hence, at the ith iteration in the for loop, we are computing a^{f_i}, down to a^{f_0} in the last iteration. We leave it as an exercise to show that this algorithm indeed completes in $\mathcal{O}(n^3)$ time.

Now that we have all these handy tools, let us revert to the cryptography world and use them to define public key encryption schemes.

1.3 Public Key Encryption

A PKE scheme $\mathsf{AE} = (\mathsf{kg}, \mathsf{enc}, \mathsf{dec})$ is a triple of algorithms. Key generation (kg) is randomized and outputs a key pair (pk, sk). Encryption (enc) takes as input a public key pk and message M and outputs a ciphertext. Decryption (dec) takes as input a secret key sk and ciphertext C and outputs a message or a distinguished error symbol $\perp$.

An ideal candidate for such a scheme would consist of a *trapdoor* function along with a corresponding kg. If f is a trapdoor function, then there exist efficient algorithms to compute $f_{pk}(\cdot)$ and $g_{pk}(\cdot, \cdot)$, such that $g_{pk}(f_{pk}(x), sk) = x$. However, there does not exist (or is assumed not to exist) an algorithm to invery f_{pk} without the corresponding sk.

1.4 Constructing RSA

We first start with a claim, which will help us construct key pairs:

Claim 5. *Let $N = pq$, for some primes p and q. Given d, e, such that $d = e^{-1}$, i.e. $de \equiv 1 \mod \phi(N)$, then $(a^e)^d \equiv a \mod N$ for all $a \in \mathbb{Z}_N^*$.*

Proof. Recall, that Theorem 4 (Euler's theorem) states that $a^{\phi(N)} \equiv 1 \mod N$. Hence, $a^{de} = a^{k\phi(N)+1}$, for some $k \in Z$ and so, $a^{de} = a^{k\phi(N)} \times a \equiv 1 \times a \equiv a \mod N$. □

This shows, that given our $\mathcal{O}(n^3)$ algorithm, given such an $N = pq$ and d, e, it is possible to compute $f_{N,e} = a^e$ in $\mathcal{O}(n^3)$ time. Further, if it is hard to find $c^{1/e}$ for $c \in Z_N^*$, then it is hard to invert $f_{N,e}$. On the other hand, as we have shown, it is easy (in $\mathcal{O}(n^3)$ time) to find $f_{N,e}^{-1}(c)$ when $d = e^{-1}$ is known. Hence, we have defined the RSA trapdoor function.

Finally, we can realize a Public Key Encryption scheme, with kg being the process of generating two primes p, q, $N = pq$ and some e and $d = e^{-1} \mod N$, $pk = (N, e)$ and $sk = (p, q, e)$ (which can be used to compute N, $\phi(N)$ and d). Given a message $M \in Z_N^*$, enc computes $C = M^e \mod N$ and dec computes $C^d \mod N$.

1.4.1 Generating Primes for RSA and Other Such Cryptosystems

RSA and other cryptosystems we will later see require the ability to generate large primes. Fortunately, the prime number theorem tells us that primes are distributed such that given any large integer N, the probability that a number less than N is prime is $1/\log(N)$. Stated differently, the prime number theorem says that given a number N, it is prime with probability $1/\log(N)$. In fact, it is well known that if we want to find an n bit prime, in expectation, we can test polynomially in n many n-bit integers to find one.

Given a prime p, we know $\phi(p) = p-1$, so for any prime p, $a^{p-1} \equiv 1 \mod p$ for any $a < p$. And, in fact, a small number of such *witnesses* of primality would suffice. However, R.D. Carmichael [**carmichael1910note**] found that there were composites which had no witness of non-primality which are known as pseudoprimes. Others [**alfordinfinite**] later showed that there are infinitely many such numbers, hence ruling out such a test.

Other tests include Miller-Rabin, which has a much lower probability of returning true on non-primes. The Miller-Rabin Primality test is based on the following property of primes:

Claim 6. *Given an odd prime, p, write $p - 1 = 2^k q$ for odd q. If $gcd(a, p) = 1$. Then either $a^q \equiv 1 \mod p$, or one of $a^q, a^{2q}, ..., a^{2(k-1)q} \equiv -1 \mod p$.*

Proof. With the same notation as in the statement, if $a^q \equiv 1 \mod p$, we are done. On the other hand, if not, then there must be some element w on the list $a^q, a^{2q}, ..., a^{2(k-1)q}$ such that $w \not\equiv 1 \mod p$ and $w^2 \equiv 1 \mod p$, so it must be that $w \equiv -1 \mod p$. □

Remark 7. On the other hand, for an odd composite number c, it turns out that at least $\frac{3}{4}$ of the positive integers $< c$ do not satisfy the above property. For more details, we defer to other sources, such as Hoffstein et al [**hoffstein2008introduction**].

Hence, checking t numbers less than any candidate prime p ensures with probability at least $1 - \left(\frac{1}{4}\right)^t$ that p is prime, if it none of the tested numbers fail to satisfy the above property. We can choose t appropriately to get very small failure probability. Finally, we can combine the prime number theorem with Miller-Rabin and/or other primality tests to get efficient algorithms for kg.

Note that recent work by Abrecht et al. [**albrecht2018prime**] has shown flaws in many current implementations of primality testing which lend themselves to returning pseudoprimes or numbers which make it hard to efficiently find witnesses of compositeness.

1.4.2 A Brief Overview of Attacks on 'Naive' RSA

Now that we have defined RSA as above and can generate primes, we must be ready to use it and all our woes must have disappeared! Unfortunately, RSA as we have defined it, i.e. with $pk = \langle N, e\rangle$ and $sk = \langle p, q, d\rangle$, has various issues that make it hard to use securely in practice. Here we discuss some of the attacks on this 'naive' RSA. Due to the fact that the underlying mathematics for some of these attacks is out of scope for this text, we will only go into the details of a few of them and include a (partial) list below. More details and pointers to original sources can be found in [**boneh1999twenty**].

The following failures were found in various parts of RSA:

- As we have shown, if $\phi(N)$ is known, it is easy to get the private key. This quickly rules out shared modulus.
- Due to the property of *malleability*, if an RSA key is used for signatures, and Eve asks Bob to sign a nonce for some protocol, she may use it to get his signature for something else. In particular, if Eve wants Bob to sign M, she can pick a random r, and get Bob to sign the "nonce" $r^e M \mod N$. Now, Bob sends Eve $Sig = (r^e M)^d$ and Eve computes $Sig/r = M^d$ to obtain a false signature.
- If a small private exponent is used, that is, $d < 1/3N^{1/4}$, then due to a certain approximation relation, d can be recovered in linear-time. It was found later, that any $d < N^{0.292}$ is also susceptible to attacks.
- Low public exponents also cause vulnerabilities. This is primarily to a theorem by Coppersmith which shows an algorithm for efficiently solving monic polynomials. Due to this theorem, for instance, ciphertexts of messages which are polynomially related can be used to recover the messages themselves. More concretely, this attack renders many simple padding schemes vulnerable to full ciphertext recovery. This theorem also helps to prove that partial private keys can lead to recovery of full keys.
- We have already seen how deterministic encryption is not secure. In order to prevent this, padding was proposed and included in an old standard known as Public Key Cryptography Standard 1 (PKCS 1). The padding they preposed was to include 16 bits containing "02" at the start of a message to indicate padding, then a random pad not containing "00" and then 16 bits containing "00" to indicate the end of the padding as shown below.

02	Random	00	Message

Now, suppose Bob's machine returns an error if a ciphertext was improperly formatted and nothing otherwise. Bleichenbacher mounted an attack on this scheme, where he was able to efficiently decrypt a ciphertext C given only the above functionality. Here's how it started: given a ciphertext C, pick random r and send Bob $rC \mod N$ until Bob doesn't return an error, indicating that the first sixteen bits of rC are "02". Using this functionality alone, Bleichenbacher showed how to decrypt the entire ciphertext.

Chinese Remainder Theorem

Now that low public and private exponents have been ruled out, we would now be left without efficient ways of decrypting on small devices. A solution to this lies in using the Chinese Remainder Theorem (CRT).

Theorem 8 (Chinese Remainder Theorem)**.** *Given a set $m_1, ..., m_n$ of pairwise relatively prime integers, i.e. $gcd(m_i, m_j) = 1$ whenever $i \neq j$, and a set of congruences: $x \equiv r_i \mod m_i$, then we can find a unique solution $x = R \mod (m_1 \times ... \times m_n)$, which satisfies each of these congruences. That is to say, there exists $0 \leq R < (m_1 \times ... \times m_n)$ such that $R \equiv r_j \mod m_j$ for each j and if R' is also a solution, then, $R' \equiv R \mod (m_1 \times ... \times m_n)$.*

Proof. The theorem is clearly true when $n = 1$, i.e. we only consider one m_i. Now, suppose that we can find a solution to the congruences from the theorem statement whenever we have up to k different congruences. That is, suppose that whenever $m_1, ..., m_n$ are integers such that $gcd(m_i, m_j) = 1$ whenever $i \neq j$ and $n \leq k$, there exists a solution R to the congruences $x \equiv r_1 \mod m_1$, ..., $x \equiv r_n \mod m_n$.

Now consider the set $m_1, ..., m_k, m_{k+1}$ of pairwise co-prime integers as well as the congruences $x \equiv r_1 \mod m_1$, ..., $x \equiv r_{k+1} \mod m_{k+1}$. We know from our assumption, that there exists a solution R_k to the first k congruences. Now suppose we write a solution $x = R_k + (m_1 \times ... \times m_k)y$ to $x \equiv r_{k+1} \mod m_{k+1}$. So,

$$R_k + (m_1 \times ... \times m_k)y \equiv r_{k+1} \mod m_{k+1}$$

and hence

$$(m_1 \times ... \times m_k)y \equiv r_{k+1} - R_k \mod m_{k+1}.$$

But, recall that the integers co-prime to m_{k+1} form a group, and hence, we can find an inverse, $w = (m_1 \times ... \times m_k)^{-1}$ in $Z^*_{m_{k+1}}$ and hence, by multiplying w on both sides of the congruence, we get

$$y \equiv w(r_{k+1} - R_k) \mod m_{k+1}.$$

Denote, $w(r_{k+1} - R_k)$ as z. Finally, we replace y above, to get the value of x, and $x \equiv r_{k+1} \mod m_{k+1}$ but also, $x \equiv r_i \mod m_i$ by the assumption above.

To show uniqueness, suppose not, i.e. there exists a solution $R' \not\equiv R \mod m_1, ..., m_{k+1}$. However, clearly $R \equiv r_i \mod m_i$, so $R' \not\equiv r_i \mod m_i$ and is not a solution. □

The process we just used in the proof also gives us an efficient algorithm for finding solutions.

Now, in RSA, if we want to compute $M = C^d \mod N$, where $N = pq$ for primes p and q, we know that this value must satisfy the congruences $M = C^d \mod p$ and $M = C^d \mod q$. We can also reduce the exponents further, and denote $d_p = d \mod \phi(p)$ and $d_q \mod \phi(q)$ to obtain the congrences $M = C^{d_p} \mod p$ and $M = C^{d_q} \mod q$ and hence using the Chinese Remainder Theorem, we can obtain $M = C^d \mod N$. In fact, it is possible to find d such that $d \mod \phi(p)$ and $d \mod \phi(q)$ are small but $d \mod \phi(N)$ is not.

In the exercises, we ask you to construct a few attacks using the Chinese Remainder Theorem.

Side-Channel Attacks and the Montgomery Ladder

Recall that when a user, Bob gets a ciphertext, he can use FAST-POW to obtain the plaintext. Note that FAST-POW has one extra multiplication step whenever a bit of the secret exponent d is non-zero. This means that an adversary $\mathcal{A}$, given a decryption oracle, can recover the bits of the key using the time taken to decrypt each of different messages. Intuitively, this is how the attack would work: start from the most significant bit and offline, measure what the time taken for the first iteration in the for-loop in Algorithm 1 if the first bit were 1. It is the case that if the time taken for the first iteration computation offline was higher than the expected value, then the total time for the decryption of that chosen ciphertext would be higher than the expected value. Conversely, if it is lower than the expected value, then the total time to decrypt would be lower than the expected value. With a guess for the most significant bit, the same strategy can be employed moving to less significant bits. A more formal treatment can be found in [**boneh1999twenty**].

So what does one do instead? On the one hand, it would seem as though the way to work-around this issue is to add in extra computation. On the other hand, the folks working on Elliptic Curve Cryptography lent us a useful tool called the Montgomery Ladder, which was later adapted for general commutative groups by Joye et. al [**joye2002montgomery**]. Again, given $a \in \mathbb{Z}_N^*$, and x, we would like to compute a^x more efficiently than the naive approach. So we rewrite

$$x = 2^k b_k + ... + 2^0 b_0$$

and define the sequence

$$f_j = \sum_{i=j}^{k} 2^{i-j} b_i.$$

Then, if $g_j = f_j + 1$:

$$f_{j-1} = 2f_j + b_{j-1} = f_j + g_j + b_{j-1} - 1 = 2g_j + b_{j-1} - 2$$

and:

$$(f_{j-1}, g_{j-1}) = \begin{cases} (2f_j, f_j + g_j) \text{ if } b_{j-1} = 0 \\ (f_j + g_j, 2g_j) \text{ if } b_{j-1} = 1. \end{cases}$$

This results in Algorithm **??**.

Algorithm 2 MONTGOMERY (a, x)

1: If $x < 0$, $a \longleftarrow a^{-1}$ and $x \longleftarrow -n$.
2: Let $b_k...b_0$ be the binary representation of x and $g_0 \longleftarrow 1$ and $g_1 \longleftarrow a$.
3: **for all** $i = k...0$ **do**
4: If $b_i = 0$, $g_0 \longleftarrow g_0^2$, $g_1 \longleftarrow g_0 g_1$
5: If $b_i = 1$, $g_0 \longleftarrow g_0 g_1$, $g_1 \longleftarrow g_1^2$.
6: **end for**
7: **return** g_0

While by itself, this algorithm is somewhat slower in expectation that FAST-POW , it lends itself to parallelization which makes it significantly faster than FAST-POW and its variants. And since each step consists of the same set of operations, the above side-channel attack is mitigated. See [**joye2002montgomery**] for other benefits of this approach.

RSA and Factoring

We know that if a fast algorithm for integer factoring can be found, the hardness of RSA no longer holds. Much work has been done in this area and thus far, the best candidate has been

something known as a number field sieve. See Table **??** for a comparison of the efficiency of the various algorithms. Again, due to the involved mathematics that goes into showing details, we will limit ourselves to a high level picture of these algorithms.

As one would expect, the naive algorithm for factoring a number N involves checking the divisibility of N with each $i < \sqrt{N}$.

Algorithm	**Running Time** $(n = \log N)$
Naive	$\mathcal{O}(e^{0.5 \log n})$
Quadratic Sieve	$\mathcal{O}\left(e^{c(\log N)^{0.5}(\log\log N)^{0.5}}\right)$
Number Field Sieve	$\mathcal{O}\left(e^{c(\log N)^{1/3}(\log\log N)^{2/3}}\right)$

Table 2: Algorithmic Complexity for Operations in $\mathbb{Z}_N^*$

The sieves start with the following very simple equation:

$$(a+b)(a-b) = a^2 - b^2.$$

Which means, to find factors p and q of N, we want to find a and b such that

$$N + a^2 = b^2$$

and then factors of N would be $a+b$ and $a-b$. Of course by itself, this may be difficult, so instead, we loosen the requirement to finding a and b such that

$$kN + a^2 = b^2$$

for some integer k. In this case, it is likely, that $b^2 - a^2$ has a non-trivial common factor with N. In other words, we are looking for two integers, a and b, such that

$$a^2 \equiv b^2 \mod N.$$

This yields the following steps:

- Find many values $a_i, ..., a_n$, such that $c_i = a_i^2 \mod N$ factors as a product of small primes. This is so that it's more efficient to find the corresponding values for b_i, using linear algebra for the powers.
- Find a subset of c_i's so that the power of every prime is even. That is, let $a = a_{i_1}^2 \times ... \times a_{i_k}^2 \equiv c_{i_1} \times ... \times c_{i_k} \equiv b^2 \mod N$.
- Now, $a^2 \equiv b^2 \mod N$, and we can use the above identity.

The quadratic number sieve makes the first step significantly faster by using a procedure similar to what we learnt in grade school, to find primes (also known as the Sieve of Eratosthenes, an ancient Greek technique), where we go through a list up to some number and cancel out values which are divisible by some small prime. At the same time, a number field (another algebraic structure similar to a group), lends itself to more general techniques, allowing for even more efficient methods for this first step.

These techniques are not included in much detail here but an interested reader is encouraged to consult a number theory source. See [**hoffstein2008introduction**] for a discussion, for instance. Hoffstein et al also note that the fastest known method for factoring large numbers N up to about 2^{350} is the quadratic number sieve and that for larger number, say larger than 2^{450}, number field sieve wins out.

1.5 Security and Definitions for RSA

As we have seen, there are various issues with naive RSA, so we set out to create a different protocol, derived from RSA and to define its security.

Consider the following encryption scheme:

Let $\mathsf{SE} = (\mathsf{kg_s}, \mathsf{enc_s}, \mathsf{dec_s})$ be a one-time secure symmetric encryption scheme where $\mathsf{kg_s}$ outputs random n-bit string. Let $H : \mathcal{M} \to \{0,1\}^n$ be a hash function with a suitable message space. Also, let $pk = (N, e)$ and $sk = (N, d)$ be the usual RSA parameters. Let enc and dec be defined as follows:

$\underline{\mathsf{enc}((N,e),M)}$:
$R \leftarrow\!\!\$\ \mathbb{Z}_N^*$
$C_1 \leftarrow R^e \bmod N$
$K \leftarrow H(R)$
$C_2 \leftarrow\!\!\$\ \mathsf{enc_s}(K, M)$
Ret (C_1, C_2)

$\underline{\mathsf{dec}((N,d),(C_1,C_2))}$:
$R \leftarrow C_1^d \bmod N$
$K \leftarrow H(R)$
$M \leftarrow \mathsf{dec_s}(K, C_2)$
Ret M

We will now define IND-CPA security for a public key encryption scheme and show how the above scheme realizes it. Define the IND-CPA game as follows, for an adversary $\mathcal{A}$, which can make at most q queries to the Enc oracle:

$\underline{\text{IND-CPA}^{\mathcal{A}}_{\mathsf{AE}}}$
$(pk, sk) \leftarrow\!\!\$\ \mathsf{kg}$
$b \leftarrow\!\!\$\ \{0,1\}$
$b' \leftarrow\!\!\$\ \mathcal{A}^{\text{Enc}}(pk)$
Ret $b = b'$

$\underline{\text{Enc}(M_0, M_1)}$
If $|M_0| \neq |M_1|$ then
  Ret $\perp$
$C \leftarrow\!\!\$\ \mathsf{enc}(pk, M_b)$
Ret C

As in the case of symmetric encryption, we define

$$\mathbf{Adv}^{\text{ind-cpa}}_{\mathsf{AE}}(\mathcal{A}) = 2 \cdot \Pr[\text{IND-CPA}^{\mathcal{A}}_{\mathsf{AE}} \implies \mathsf{true}] - 1.$$

Correspondingly, we can also define $\text{IND-CPA0}^{\mathcal{A}}_{\mathsf{AE}}$ and $\text{IND-CPA1}^{\mathcal{A}}_{\mathsf{AE}}$ and then define:

$$\mathbf{Adv}^{\text{ind-cpa}}_{\mathsf{AE}}(\mathcal{A}) = |\Pr[\text{IND-CPA1}^{\mathcal{A}}_{\mathsf{AE}} = 1] - \Pr[\text{IND-CPA0}^{\mathcal{A}}_{\mathsf{AE}} = 1]|.$$

To make things easier moving forward, we prove the following result:

Theorem. *Let AE be a PKE scheme. Let $\mathcal{A}$ be an* IND-CPA$_{\mathsf{AE}}$*-adversary making at most q queries. We give an* IND-CPA$_{\mathsf{AE}}$*-adversary $\mathcal{B}$ making one query such that*

$$\mathbf{Adv}^{\text{ind-cpa}}_{\mathsf{AE}}(\mathcal{A}) \leq q \cdot \mathbf{Adv}^{\text{ind-cpa}}_{\mathsf{AE}}(\mathcal{B})$$

Adversary $\mathcal{B}$ runs in time that of $\mathcal{A}$ plus the time to perform $(q-1)$ encryptions under AE.

Proof. We will use a hybrid approach to prove this theorem. We define the games G_{i^*} for $i^* \in \{0, ..., q\}$ such that:

- G_0 has an encryption oracle that replies to all queries with encryptions of M_1,
  $\vdots$

- G_k has an encryption oracle that replies to the first k queries with encryptions of M_1 and the remaining queries with encryptions of M_0

  $\vdots$

- G_q has an encryption oracle that replies to all queries with the encryptions of M_0.

Hence,

$$G_0 = \text{IND-CPA1}_{\mathsf{AE}}$$
$$G_q = \text{IND-CPA0}_{\mathsf{AE}}.$$

Below is a description of G_{i^*}:

$\underline{\mathrm{G}_{i^*}}$
$b \leftarrow\!\!\$\ \{0,1\}$
$(pk, sk) \leftarrow\!\!\$\ \mathsf{kg}$
$i \leftarrow 1$
$b' \leftarrow\!\!\$\ \mathcal{A}^{\mathrm{EncSim}}(pk)$
Ret b'

$\underline{\mathrm{EncSim}(M_0, M_1)}$
If $|M_0| \neq |M_1|$ then
  Ret $\perp$
If $i > i^*$ then
  $C \leftarrow\!\!\$\ \mathsf{enc}(pk, M_0)$
Else
  $C \leftarrow\!\!\$\ \mathsf{enc}(pk, M_1)$
$i \leftarrow i + 1$
Ret C

Recall that since $i^* = 0$ and $i^* = q$, we get exactly the games IND-CPA0$^{\mathcal{A}}$ and IND-CPA1$^{\mathcal{A}}$ respectively, so we can write:

$$\mathbf{Adv}^{\text{ind-cpa}}_{\mathsf{AE}}(\mathcal{A}) = |\Pr[\,\mathrm{G}_0 \Rightarrow 1\,] - \Pr[\,\mathrm{G}_q \Rightarrow 1\,]|$$

Now, consider the following hybrid $\mathcal{B}^{\mathsf{enc}}_{i*}$ which can run a simulator EncSim as well. This means that B_{i^*} is a single-query IND-CPA adversary that "switches" between G_{i^*-1} and G_{i^*}.

$\underline{\mathcal{B}^{\mathrm{Enc}}_{i^*}(pk)}$
$i \leftarrow 1$
$b' \leftarrow\!\!\$\ \mathcal{A}^{\mathrm{EncSim}}(pk)$
Ret b'

$\underline{\mathrm{EncSim}(M_0, M_1)}$
If $|M_0| \neq |M_1|$ then
  Ret $\perp$
If $i > i^*$ then
  $C \leftarrow\!\!\$\ \mathsf{enc}(pk, M_1)$
Else if $i = i^*$ then
  $C \leftarrow \mathrm{Enc}(M_0, M_1)$
Else
  $C \leftarrow\!\!\$\ \mathsf{enc}(pk, M_0)$
$i \leftarrow i + 1$
Ret C

Suppose $\mathcal{B}$ chooses uniformly at random which $\mathcal{B}_{i^*}$ world to be in. That is, $\mathcal{B}$ runs the game:

$\underline{\mathcal{B}^{\mathrm{Enc}}(pk)}$
$i^* \leftarrow\!\!\$\ \{0,...,q-1\}$
$b' \leftarrow\!\!\$\ \mathcal{B}_{i^*}^{\mathrm{EncSim}}(pk)$
Ret b'

Then,

$$\begin{aligned}
\mathbf{Adv}^{\text{ind-cpa}}_{\mathsf{AE}}(\mathcal{B}) &= \left|\Pr\left[\,\text{IND-CPA1}^{\mathcal{B}} \Rightarrow 1\,\right] - \Pr\left[\,\text{IND-CPA0}^{\mathcal{B}} \Rightarrow 1\,\right]\right| \\
&= \frac{1}{q}\left|\sum_{i^*=1}^{q} \Pr\left[\,\text{IND-CPA1}^{\mathcal{B}} \Rightarrow 1 \mid j = i^*\,\right] - \Pr\left[\,\text{IND-CPA0}^{\mathcal{B}} \Rightarrow 1 \mid j = i^*\,\right]\right| \\
&\qquad \text{(since } \mathcal{B} \text{ picks } i^* \text{ uniformly at random)} \\
&= \frac{1}{q}\left|\sum_{i^*=1}^{q} \Pr\left[\,\text{IND-CPA1}^{\mathcal{B}_{i^*}} \Rightarrow 1\,\right] - \Pr\left[\,\text{IND-CPA0}^{\mathcal{B}_{i^*}} \Rightarrow 1\,\right]\right| \\
&= \frac{1}{q}\sum_{i=0}^{q-1} \left|\Pr\left[\,\mathrm{G}_i \Rightarrow 1\,\right] - \Pr\left[\,\mathrm{G}_{i+1} \Rightarrow 1\,\right]\right| \\
&\geq \frac{1}{q}\left|\Pr\left[\,\mathrm{G}_0 \Rightarrow 1\,\right] - \Pr\left[\,\mathrm{G}_q \Rightarrow 1\,\right]\right| \ \text{(triangle inequality)} \\
&= \frac{1}{q}\mathbf{Adv}^{\text{ind-cpa}}_{\mathsf{AE}}(\mathcal{A}).
\end{aligned}$$

□

Next, let us define a security game for an RSA based scheme with a k bit security parameter RSA-k. Intuitively, the security of RSA is based on difficulty of inversion, so we define the security as the security of a one-way function, where the adversary knows the public parameters:

$\underline{\mathrm{OWF}_{\text{RSA-}k}}$
$((N,e),(N,d)) \leftarrow\!\!\$\ \mathsf{kg}(k)$
$X \leftarrow\!\!\$\ \mathbb{Z}_N^*$
$Y \leftarrow X^e \bmod N$
$X' \leftarrow\!\!\$\ \mathcal{A}(Y,(N,e))$
Ret $(X' = X)$

$$\mathbf{Adv}^{\text{owf}}_{\text{RSA-}k}(\mathcal{A}) = \Pr\left[\,\mathrm{OWF}^{\mathcal{A}}_{\text{RSA-}k} \Rightarrow \mathsf{true}\,\right]$$

Finally, we show:

Theorem. *Let* RSA-k *be the RSA-based scheme using security parameter* k*, hash function* $\mathsf{H}\colon \mathcal{M} \to \{0,1\}^n$ *modeled as a random oracle, and symmetric encryption scheme* SE*. Let* $\mathcal{A}$ *be an* $\text{IND-CPA}_{\text{RSA-}k}$*-adversary making at most* q *queries to* H*. Then we give an* $\mathrm{OWF}_{\text{RSA-}k}$*-adversary* $\mathcal{B}$ *and* $\text{IND-ROR}_{\mathsf{SE}}$*-adversary* $\mathcal{C}$ *such that*

$$\mathbf{Adv}^{\text{ind-cpa}}_{\text{RSA-}k,\mathsf{H}}(\mathcal{A}) \leq 2\cdot\mathbf{Adv}^{\text{owf}}_{\text{RSA-}k}(\mathcal{B}) + 2\cdot\mathbf{Adv}^{\text{ror}}_{\mathsf{SE}}(\mathcal{C})\ .$$

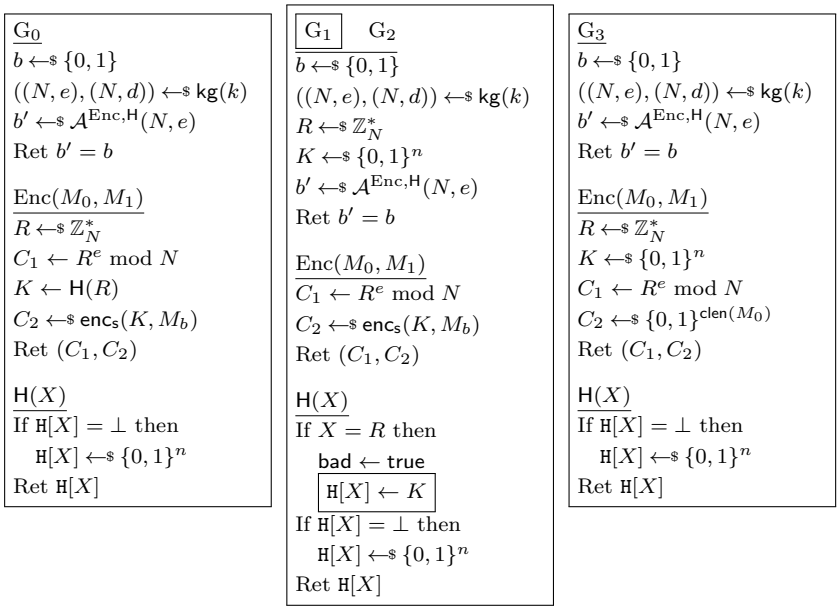
Adversaries $\mathcal{B},\mathcal{C}$ *run in time that of* $\mathcal{A}$ *plus the time to simulate* q *random oracle queries. Adversary* $\mathcal{C}$ *makes a single encryption query.*

Proof. Below is a set of games we will use to prove this theorem. First, note that G_0 simulates exactly, the IND-CPA security game and so,

$$\mathbf{Adv}^{\text{ind-cpa}}_{\text{RSA-}k,\mathsf{H}}(\mathcal{A}) = 2\cdot\Pr\left[\,\mathrm{G}_0 \implies \mathsf{true}\,\right] - 1.$$

Next, observe that G_1 is just G_0 but with the order of generating the symmetric encryption key re-written, so it remains

$$\mathbf{Adv}^{\text{ind-cpa}}_{\text{RSA-}k,\mathsf{H}}(\mathcal{A}) = 2 \cdot \Pr[\,\mathrm{G}_1 \implies \mathsf{true}\,] - 1.$$



Now, between G_1 and G_2, the only difference is when $\mathsf{bad} \leftarrow \mathsf{true}$. This happens whenever $\mathcal{A}$ queries the oracle with R as input. Given that $\mathcal{A}$ has access to C_1, she can run an $\mathrm{OWF}_{\text{RSA-}k}$ adversary $\mathcal{B}$ (as defined above), to attempt to obtain R as a subroutine. Hence $\mathsf{bad} \leftarrow \mathsf{true}$ whenever $\mathcal{B}$ succeeds and by the fundamental lemma of game playing:

$$|\Pr[\,\mathrm{G}_1 \implies \mathsf{true}\,] - \Pr[\,\mathrm{G}_2 \implies \mathsf{true}\,]| \leq \mathbf{Adv}^{\text{owf}}_{\text{RSA-}k}(\mathcal{B})$$

and so, by triangle inequality:

$$\Pr[\,\mathrm{G}_1 \implies \mathsf{true}\,] \leq \mathbf{Adv}^{\text{owf}}_{\text{RSA-}k}(\mathcal{B}) + \Pr[\,\mathrm{G}_2 \implies \mathsf{true}\,].$$

Since the only difference between the encryption schemes in G_2 and G_3 is that the encryption is real in G_2 and random in G_3. Hence, for an IND-ROR adversary $\mathcal{C}$,

$$|\Pr[\,\mathrm{G}_3 \implies \mathsf{true}\,] - \Pr[\,\mathrm{G}_2 \implies \mathsf{true}\,]| = \mathbf{Adv}^{\text{ror}}_{\mathsf{SE}}(\mathcal{C}).$$

Since the adversary is in the random world in G_3,

$$\Pr[\,\mathrm{G}_3 \implies \mathsf{true}\,] = \frac{1}{2}.$$

So,

$$\Pr[\,\mathrm{G}_2 \implies \mathsf{true}\,] \leq \frac{1}{2} + \mathbf{Adv}^{\text{ror}}_{\mathsf{SE}}(\mathcal{C}).$$

And hence,

$$\Pr[\,\mathrm{G}_1 \implies \mathsf{true}\,] \leq \mathbf{Adv}^{\text{owf}}_{\text{RSA-}k}(\mathcal{B}) + \mathbf{Adv}^{\text{ror}}_{\mathsf{SE}}(\mathcal{C}) + \frac{1}{2}.$$

so,

$$\begin{aligned}\mathbf{Adv}^{\text{ind-cpa}}_{\text{RSA-}k,\mathsf{H}}(\mathcal{A}) &= 2\Pr[\,\mathrm{G}_1 \implies \mathsf{true}\,] - 1 \\ &\leq 2\left(\mathbf{Adv}^{\text{owf}}_{\text{RSA-}k}(\mathcal{B}) + \mathbf{Adv}^{\text{ror}}_{\mathsf{SE}}(\mathcal{C}) + \frac{1}{2}\right) - 1 \\ &\leq 2\cdot\mathbf{Adv}^{\text{owf}}_{\text{RSA-}k}(\mathcal{B}) + 2\cdot\mathbf{Adv}^{\text{ror}}_{\mathsf{SE}}(\mathcal{C})\end{aligned}$$

□

Exercises

Exercise 1.1 While we haven't yet covered digital signatures, you have likely heard of the concept. What you really need to know is that to sign a message M using RSA, a signer, Bob with $pk = (N, e)$ and $sk = (p, q, d)$ where $N = pq$, computes

$$S = M^d \mod N$$

and a verifier, Eve, checks if $S^e = M \mod N$. In order to speed up this computation, sometimes the Chinese Remainder Theorem is used. To do this, Bob computes

$$S_p = M^{d \mod (p-1)} \mod p \text{ and } S_q = M^{d \mod (q-1)} \mod q$$

and then

$$S = aS_p + bS_q \mod N.$$

where $a \equiv 1 \mod p$ and $b \equiv 1 \mod q$, $a \equiv 0 \mod q$ and $b \equiv 0 \mod p$. Mount an attack against this signature scheme, if you know that one of the computations for S_p or S_q fails with a non-negligible probability $\pi(n)$ in $n = \log N$ (the length of N). Also, compute the

Exercise 1.2 Suppose Bob publishes 2 public keys (N, e_0) and (N, e_1) and Alice, wanting to be extra sure that Bob gets her message M, sends him $M^{e_0} \mod N$ and $M^{e_1} \mod N$. Bob decrypts whichever message he receives first and returns it. If a second message with the same plaintext but different key arrives later, he ignores it. In other words, describe the scheme as:

$\underline{\mathsf{enc}((N,e_0),(N,e_1),M)}$:
$C_0 \leftarrow M^{e_0} \bmod N$
$C_1 \leftarrow M^{e_1} \bmod N$
Ret (C_0, C_1)

$\underline{\mathsf{dec}((N,d_1),(N,d_2),(C_0,C_1))}$:
$b \leftarrow\!\!\$\ \{0,1\}$
$M \leftarrow C_b^{d_b} \bmod N$
Ret M

Is this scheme IND-CPA secure? If so, show a reduction to the RSA assumption. If not, then provide an attack.