CSCI 7000-001 — Lecture Notes
Public Key Algorithms: RSA

September 13, 2001

These notes are derived, with a few modifications, from Section 5.2 and Section 5.3 of "Network Security," C. Kaufman, R. Perlman, and M Speciner. Prentice-Hall.

Most public-key algorithms are based on modular arithmetic.

1 Modular Arithmetic

- non-negative integers $\leq n$
- normal operations such as $+$ and $-$
- result is the reminder of the division by n
- notation: $x \mod n$

Examples:

- let's consider some arithmetic $\mod 10$

$$5 + 5 = 0 \mod 10$$

$$2 + 9 = 1 \mod 10$$

$$6 + 7 = 3 \mod 10$$

- just like in normal arithmetic the *additive inverse* of x, denoted by $-x$ is that added to x gives $0 \mod 10$. So, for example:

$$-3 = 7 \mod 10$$

  because

$$3 + 7 = 0 \mod 10$$

  So, if we were to add $3 \mod 10$ as a form of encryption, we would have to add $7 \mod 10$ to decrypt. In fact, what was that cipher that used $+3 \mod 26$ as an encryption function?

okay, adding is easy. Let's look at multiplication $\mod 10$. Examples:

- $$3 \times 5 = 5 \mod 10$$

- $$4 \times 5 = 0 \mod 10$$

- $$7 \times 3 = 1 \mod 10$$

We can construct a multiplication table:

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9
2	0	2	4	6	8	0	2	4	6	8
3	0	3	6	9	2	5	8	1	4	7
4	0	4	8	2	6	0	4	8	2	6
5	0	5	0	5	0	5	0	5	0	5
6	0	6	2	8	4	0	6	2	8	4
7	0	7	4	1	8	5	2	9	6	3
8	0	8	6	4	2	0	8	6	4	2
9	0	9	8	7	6	5	4	3	2	1

Notice that multiplying by 1, 3, 7, and 9 mod 10 would work as a cipher—because it performs a substitution, with no compression—but multiplying by 2, 4, 5, 6, and 8 mod 10 would not work.

So, let's say we use $\times 3 \mod 10$ as an encryption function:

$$\textit{ciphertext} = \textit{plaintext} \times 3 \mod 10$$

What is the decryption function? We need the *multiplicative inverse* of 3, $3^{-1} \mod 10$, in order to compute (decrypt) the plaintext from the ciphertext:

$$\textit{plaintext} = \textit{ciphehrtext} \times 3^{-1} \mod 10$$

.

In fact, by definition, the multiplicative inverse of x, $x^{-1} \mod 10$ is such that $x \times x^{-1} = 1 \mod 10$. Now, the question is (1) whether $x^{-1} = 1 \mod 10$ exists for some x, and (2) if it exists, how to compute $x^{-1} \mod 10$, given x.

We can answer these questions (in the particular case of $\mod 10$) by looking up the multiplication table. In particular, we can see that the only numbers that have a multiplicative inverse—the only ones that have a 1 somewhere in their row/column—are 1, 3, 7, and 9. The reason why 1, 3, 7, and 9 have a multiplicative inverse mod 10, and the reason why they are the only numbers that produce a valid symmetric cipher in a multiplication mod 10 is that 1, 3, 7, and 9 are *relatively prime* to 10. The answer to question (2) is the *Euclidean algorithm*, which we won't study at this point.

It is useful to know how many numbers less than n are relatively prime to n. That count is so important in number theory that it has been given a special notation. It's called the *totient function*, and it is denoted by $\phi(n)$. So, how do we compute $\phi(n)$? Clearly, $\phi(n) < n$, because we said we are counting the numbers less than n. If n is prime, then $\phi(n) = n - 1$, because every number less than n is relatively prime to n. What is n is not prime? Well, the simplest case is when n is the product of two primes $n = pq$, in that case, there are some numbers, less than n, that are *not* relatively prime to n. Those are the multiples of p or q less than $n = pq$, plus 0 (which of course is not relatively prime to anything). In practice, the non relatively primes are p, $2p$, $3p$, $\ldots (q-1)p$, and q, $2q$, $3q$, $\ldots (p-1)q$, and 0. So we have $q - 1$ (from the first group), $p - 1$, from the second group plus 1 (0). So,

$$\phi(pq) = pq-(q-1)-(p-1)-1 = pq-q-p+1 = q(p-1)-(p-1) = (p-1)(q-1)$$

Why is $\phi(n)$ so important? We will see that when we talk about exponentiation mod n. Let's look at the exponentiation table mod 10:

x^y	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*
0		0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	0	2	4	8	6	2	4	8	6	2	4	8	6
3	0	3	9	7	1	3	9	7	1	3	9	7	1
4	0	4	6	4	6	6	6	4	6	4	6	4	6
5	0	5	5	5	5	5	5	5	5	5	5	5	5
6	0	6	6	6	6	6	6	6	6	6	6	6	6
7	0	7	9	3	1	7	9	3	7	9	3	7	1
8	0	8	4	2	6	8	4	2	6	8	4	2	6
9	0	9	1	9	1	9	1	9	1	9	1	9	1

Again, looking at the table, we can (almost) immediately see that raising $x^3 \mod 10$ is a valid encryption function for for plaintext x, because it scrambles values without compression, whereas an exponent of, say, 4 wouldn't work. We can also notice an interesting property of the exponentiation table (mod 10), that is, columns 1 and 5 are the same, also 2 and 6 are the same, and so are 3 and 7, 4 and 8, and so on. In other words, $x^y = x^{y+4} \mod 10$.

As it turns out, the general property of exponentiation $\mod n$ is that $x^y = x^{y \mod \phi(n)} \mod n$ (for almost every n). In particular, when $y = 1 \mod \phi(n)$, then $x^y = x \mod n$

2 RSA

1. choose two large random prime numbers: p and q, and compute $n = pq$;

2. choose an encryption key (a number) e that is relatively prime to $\phi(n)$ (remember: since $n = pq$, $\phi(n) = (p-1)(q-1)$);

3. compute the multiplicative inverse $d = e^{-1} \mod \phi(n)$.

4. your public key is $\langle e, n \rangle$;

5. your secret key is $\langle d, n \rangle$

2.1 Encryption and Decryption

Here's how someone would encrypt a message $m < n$ with your public key:

$$c = m^e \mod n$$

Here's how you decrypt that ciphertext:

$$m = c^d \mod n$$

Why does that work? Because

$$m = c^d \mod n = (m^e)^d = m^{ed} \mod n$$

but since d is the multiplicative inverse of $e \mod \phi(n)$, that is, $de = 1 \mod \phi(n)$, and given the magic of $x^y = x \mod n$ when $y = 1 \mod \phi(n)$,

$$m^{ed} \mod n = m^{1 \mod \phi(n)} \mod n = m \mod n = m$$

Now, as you can imagine, signing and verifying follow exactly the same process [hint: $(m^e)^d = m^{ed} = m^{de} = (m^d)^e$]

2.2 Why is RSA Secure?

The strength of RSA is the difficulty in factoring large numbers. In fact, if an attacker could factor n (into p and q), then he or she could easily compute d from e. Following the same process you follow when you create your key pair.

There is no proof that factoring is difficult, other than the fact that lots of smart people have not been able to find an easy solution. Also, there might be other ways to figure out d, given e and n, but someone demonstrated that, any other method would be just as difficult as factoring (i.e., factoring can be reduced to breaking RSA).

2.3 Other issues

1. how do you find large primes?

2. are there enough large-enough primes in the universe?

3. how do you compute exponentiations $\mod n$ with such large exponents?

2.4 And of course...

RSA is just one building block for protocols—a very important and very powerful one, of course. So, keep in mind that the strengths of RSA can be lost completely without a careful protocol design.