YALE UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

CPSC 467a: Cryptography and Computer Security

Notes 8 (rev. 1)
Professor M. J. Fischer *September 27, 2005*

Lecture Notes 8

1 RSA

Probably the most commonly used asymmetric cryptosystem in use today is *RSA*, named from the initials of its three inventors, Rivest, Shamir, and Adelman. Unlike the symmetric systems we have been talking about so far, RSA is based not on substitution and transposition but on arithmetic involving very large integers, numbers that are hundreds or even thousands of bits long. To understand how RSA works requires knowing a bit of number theory, which we will be presenting in the next few lectures. However, the basic ideas can be presented quite simply, which I will do now.

RSA assumes the message space, ciphertext space, and key space are the set of integers between 0 and $n - 1$, where n is a large integer. For now, think of n as a number so large that its binary representation is 1024 bits long. To use RSA in the usual case where we are interested in sending bit strings, Alice must first convert her message to an integer, and Bob must convert the integer he gets from decryption back to a bit string. Many such encodings are possible, but perhaps the simplest is to prepend a "1" to the bit string x and regard the result as a binary integer m. To decode m to a bit string, write out m in binary and then delete the initial "1" bit. To ensure that $m < n$ as required, we will limit the length of our binary messages to 1022 bits.

Here's how RSA works. Alice chooses two sufficiently large prime numbers p and q and computes $n = pq$. For security, p and q should be about the same length (when written in binary). She then computes two numbers e and d with a certain number-theoretic relationship. The public key is the pair (e, n) The private key is the pair (d, n). The primes p and q are no longer needed and should be discarded.

To encrypt, Alice computes $c = m^e \bmod n$.[1] To decrypt, Bob computes $m = c^d \pmod{n}$. Here, $a \bmod n$ means the remainder when a is divided by n. That's all there is to it, once the keys have been found. It turns out that most of the complexity in implementing RSA has to do with key generation, which fortunately is done only infrequently.

You should already be asking yourself the following questions:

- How does one find p, q, e, d with the desired properties?
- What are the desired properties that make RSA work? A priori, it seems pretty unlikely that $D_d(E_e(m)) = (m^e \bmod n)^d \bmod n = m$.
- Why is RSA believed to be secure?
- How can one implement RSA on a computer when most computers only support arithmetic on 32-bit integers, and how long does it take?
- How can one possibly compute $m^e \bmod n$ for 1024 bit numbers. m^e, before taking the remainder, is a number that is roughly 2^{1024} bits long. No computer has enough memory to store that number, and no computer is fast enough to compute it.

[1]In the remainder of this discussion, messages and ciphertexts will refer to integers in the range 0 to $n - 1$, not to bit strings.

To answer these questions will require the study of clever algorithms for primality testing, fast exponentiation, and modular inverse computation. It will also require some theory of Z_n, the integers modulo n, and some properties of numbers n that are the product of two primes. In particular, the security of RSA is based on the premise that the *factoring problem* on large integers is infeasible, that is, given n that is known to be the the product of two primes p and q, to find p and q.

2 Number Theory Review

We next review some number theory that is needed for understanding RSA. These lecture notes only provide a high-level overview. Further details are contained in course handouts 4–6 and in Chapter 3 of the textbook.

2.1 Divisibility properties

Let a, b be integers and assume $b > 0$. The *division theorem* asserts that there are unique integers q (the *quotient*) and r (the *remainder*) such that $a = bq + r$ and $0 \leq r < b$. In case $r = 0$ we say that b *divides* a (exactly) and write $b \,|\, a$.

Fact If $d \,|\, (a + b)$, then either d divides both a and b, or d divides neither of them.

To see this, suppose $d \,|\, (a + b)$ and $d \,|\, a$. Then by the division theorem, $a + b = dq_1$ and $a = dq_2$ for some integers q_1 and q_2. Subsituting for a and solving for b, we get

$$b = dq_1 - dq_2 = d(q_1 - q_2).$$

But this implies $d \,|\, b$, again by the division theorem.

2.2 Greatest common divisor

The *greatest common divisor* of two integers a and b, written $\gcd(a, b)$, is the largest integer d such that $d \,|\, a$ and $d \,|\, b$. The $\gcd$ is always defined since 1 is a divisor of every integer, and the divisor of a number cannot be larger (in absolute value) than the number itself.

The gcd of a and b is easily found if a and b are already given in factored form. Namely, let p_i be the i^{th} prime and write $a = \prod p_i^{e_i}$ and $b = \prod p_i^{f_i}$. Then $\gcd(a, b) = \prod p_i^{\min(e_i, f_i)}$. However, factoring is believed to be a hard problem, and no polynomial-time factorization algorithm is currently known. Indeed, if it were, then Eve could use it to break RSA, and RSA would be of no interest as a cryptosystem.

Fortunately, $\gcd(a, b)$ can be computed efficiently without the need to factor a and b. Here's a sketch of the ideas that lead to the famous *Euclidean algorithm*.

The $\gcd$ function satisfies several identities. In the following, assume $a \geq b \geq 0$:

$$\begin{aligned}
\gcd(a, b) &= \gcd(b, a) && (1)\\
\gcd(a, 0) &= a && (2)\\
\gcd(a, b) &= \gcd(a - b, b) && (3)
\end{aligned}$$

Identity 3 follows from the Fact above. A simple inductive proof shows that identity 3 can be strengthened to

$$\gcd(a, b) = \gcd(a \bmod b, b) \qquad (4)$$

where $a \bmod b$ is the remainder of a divided by b. The Euclidean algorithm uses identities 1, 2, and 4 recursively to compute $\gcd(a, b)$ in $O(n)$ stages, where n is the sum of the lengths of a and b when written in binary notation, and each stage requires at most one remainder computation. We will return to this topic in lecture 10.

2.3 Basic definitions and notation

The set

$$\mathbf{Z}_n = \{0, 1, \ldots, n-1\}$$

contains the non-negative integers less than n. If one defines a binary "addition" operation on $\mathbf{Z}_n$ by

$$a \oplus b \overset{\text{df}}{=} (a + b) \bmod n$$

then $\mathbf{Z}_n$ can be regarded as an Abelian group under addition ($\oplus$).

The set

$$\mathbf{Z}_n^* = \{x \in \mathbf{Z}_n \mid \gcd(x, n) = 1\}$$

contains the non-negative integers less than n that are *relatively prime* to n, that is, which do not share any non-trivial common factor with n. If one defines a binary "multiplication" operation on $\mathbf{Z}_n^*$ by

$$a \otimes b \overset{\text{df}}{=} (a \cdot b) \bmod n$$

then it can be shown that $\mathbf{Z}_n^*$ is an Abelian group under multiplication ($\otimes$).

Euler's totient (ϕ) function is defined to be the cardinality of $\mathbf{Z}_n^*$:

$$\phi(n) = |\mathbf{Z}_n^*|$$

Properties of $\phi(n)$:

1. If p is prime, then $\phi(p) = p - 1$.
2. More generally, if p is prime and $k \geq 1$, then $\phi(p^k) = p^k - p^{k-1} = (p-1)p^{k-1}$.
3. If $\gcd(m, n) = 1$, then $\phi(mn) = \phi(m)\phi(n)$.

These properties enable one to compute $\phi(n)$ for all $n \geq 1$ provided one knows the factorization of n. For example,

$$\phi(126) = \phi(2)\phi(3^2)\phi(7) = (2-1)(3-1)(3^{2-1})(7-1) = 1 \cdot 2 \cdot 3 \cdot 6 = 36.$$

The 36 elements of $\mathbf{Z}_{126}^*$ are: 1, 5, 11, 13, 17, 19, 23, 25, 29, 31, 37, 41, 43, 47, 53, 55, 59, 61, 65, 67, 71, 73, 79, 83, 85, 89, 95, 97, 101, 103, 107, 109, 113, 115, 121, 125.